Input & Output de Arquivos

Vamos ver um pouco sobre arquivos e as operações que podemos executar em Python: abrir arquivos, ler conteúdo de arquivos, escrever em arquivos e como fechá-los corretamente.

O que é um Arquivo?

Arquivo é uma localização em disco com um nome, que tem como objetivo guardar informação. É usado para guardarmos dados na memória não-volátil (disco-rígido), uma vez que a memória de acesso randômico (RAM) é volátil e perde todos os dados quando desligamos o nosso computador, para isso trabalhamos com os arquivos, de forma a fazer com que nossos dados persistam em disco.

Em sua essência, um arquivo é um conjunto contíguo de bytes usado para armazenar dados. Esses dados são organizados em um formato específico e podem ser tão simples como um arquivo de texto ou tão complicados como um executável de programa. No final, esses arquivos de bytes são então traduzidos para os binários 1 e 0 para facilitar o processamento pelo computador.

Os arquivos na maioria dos sistemas de arquivos modernos são compostos de três partes principais:

1. **Header:** Metadados sobre o conteúdo do arquivo (nome do arquivo, tamanho, tipo e assim por diante)
2. **Dados:** Conteúdo do arquivo assim escrito pelo criador ou editor dele
3. **End of File (EOF):** Caractere especial que indica o fim do arquivo

O que esses dados representam depende da especificação de formato usada, que normalmente é representada por uma extensão. Por exemplo, um arquivo que possui uma extensão .gif provavelmente está em conformidade com a especificação do Graphics Interchange Format.

Caminho dos Arquivos (Path)

Quando acessamos um arquivo em um sistema operacional, um caminho de arquivo é necessário. O caminho do arquivo é uma string que representa a localização de um arquivo. Ele é dividido em três partes principais:

Caminho da pasta/diretório

A localização da pasta de arquivos no sistema de arquivos onde as pastas subsequentes são separadas por uma barra / (Unix) ou barra invertida \\ (Windows)

Nome do Arquivo

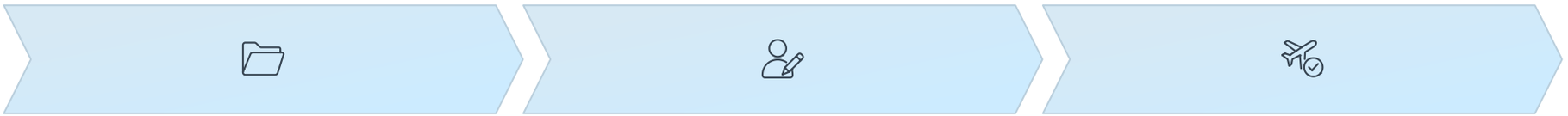
Representa o nome do arquivo

Extensão

O final do caminho do arquivo prefixado com um ponto (.) usado para indicar o tipo de arquivo

Arquivos em Python

Quando nós queremos ler um arquivo ou escrever nele, precisamos abrí-lo antes, quando estivermos finalizado, precisamos fechá-lo, para que assim os recursos que estão ligados com o arquivo, sejam liberados, sendo assim, a operação com arquivos funciona da seguinte forma:



Abrir o arquivo

Ler ou escrever (operação desejada)

Fechar o arquivo

Abrindo Arquivos

Python tem uma função construída que se chama `open()`, ela abre arquivos. Essa função retorna um file object que nos possibilita manipular o arquivo de várias maneiras, por exemplo:

```
f = open("teste.txt") # Abrindo arquivo no diretório atual

f = open("/home/usuario/teste.txt") # Abrindo o arquivo especificando o seu caminho completo (full path)
```

Podemos especificar o modo quando abrindo o arquivo, o modo nos permite especificar se queremos ler 'r', escrever 'w' ou append 'a', que seria o ato de adicionar textos no arquivo sem destruir o que já está previamente nele. É possível também especificar se queremos abrir o arquivo no modo texto ou no modo binário. O padrão é o modo leitura, nesse modo nós recebemos strings ao ler do arquivo. No caso do modo binário ele nos retorna bytes, esse modo é interessante caso queiramos lidar com arquivos que não sejam textos, como por exemplo imagens e arquivos executáveis.

Modos de Abrir Arquivos

A tabela a seguir nos apresenta os modos de abrir arquivos e suas descrições:

Modo	Nome	Descrição
"r"	Ler	Padrão, abre um arquivo para leitura, ocorrerá um erro caso o arquivo não exista
"a"	Append	Abre um arquivo para appending, cria ele caso este não exista
"w"	Escreve	Abre um arquivo para escrita, cria ele caso este não exista
"x"	Cria	Cria o arquivo especificado, retorna um erro se o arquivo já existir
"+"	Atualização	Abre o arquivo para atualização (ler ou escrever)

Além disso poderemos especificar se lidaremos em modo binário ou texto:

Modo	Nome	Descrição
"t"	Texto	Modo padrão
"b"	Binário	Modo binário

Exemplos

```
f = open("teste.txt") # O mesmo que 'r' ou 'rt'

f = open("teste.txt", "w") # escrever no modo texto

f = open("img.png", 'r+b') # ler e escrever no modo binário
```

Fechando e Escrevendo em Arquivos

Fechando um Arquivo usando Python

Uma vez que terminamos todas as operações no arquivo, é necessário que fechemos ele corretamente, fechando ele teremos recursos liberados, e para isso usaremos o método `close()`. É importante lembrarmos que Python tem um garbage collector (coletor de lixo) que limpa objetos não referenciados, mas mesmo assim é importante que fechemos os nossos arquivos por questões de boas práticas e segurança

```
f = open("teste.txt", enconding = 'utf-8')

f.close()
```

Perceba que especificamos o modo de encoding para 'utf-8' para evitarmos conflitos, uma vez que cada sistema tem um modo padrão de codificar, sendo windows o 'cp1252' e linux 'utf-8'. o método `close()` fecha o arquivo.

Dessa forma não é tão seguro, pois caso ocorra algum problema durante a operação no arquivo, o código sai sem que o arquivo seja fechado de maneira adequada. Para isso, temos uma forma mais segura, usando as palavras-chave `try` e `finally`.

```
try:
    f = open("teste.txt", encoding = 'utf-8')
finally:
    f.close()
```

Dessa forma nós garantimos que o arquivo seja fechado de maneira adequada, mesmo que algum erro ocorra. Outra maneira de contornarmos essa situação seria utilizando a palavra-chave `with`, que nos assegura que o arquivo é fechado quando o bloco de código dentro de `with` é finalizado, dessa forma não precisamos mais usar `close()` explicitamente, pois é feito internamente por nós.

```
with open("teste.txt", encoding = 'utf-8') as f:
```

Escrevendo em Arquivos

Para escrevermos em um arquivo é muito simples, só precisamos especificar o modo, seja ele 'w' para escrever, 'a' para append ou 'x' para criação exclusiva, fique atento ao usar o modo 'w', pois ele pode sobrescrever um arquivo caso ele já exista. Para escrevermos uma string ou sequencias de bytes (para arquivos binários) usamos o método `write()`, este que retorna o número de caracteres escritos no arquivo.

Vamos então criar um arquivo de nome teste.txt e escrever dados nele:

```
with open("teste.txt", 'w', encoding = 'utf-8') as f:
    f.write("Estamos escrevendo no arquivo\\n")
    f.write("Outra linha escrita no arquivo\\n")
```

Esse programa irá criar um novo arquivo de nome teste.txt caso ele não exista, se já existir será sobrescrito. Perceba que colocamos o \\n para que possamos obter novas linhas.

Lendo Arquivos

Para lermos um arquivo nós devemos abrí-lo no modo de leitura, existem diversos métodos para isso, como o `read(size)`, onde size é o tamanho, caso não especificarmos size, ele lerá e retornará até o fim do arquivo.

```
f = open("teste.txt", 'r', encoding = 'utf-8')

print(f) # <_io.TextIOWrapper name='teste.txt' mode='r' encoding='utf-8'>

f.read(4) # Lê os primeiro 4 dados ('Esta')

f.read(4) # Lê mais 4 dados ('mos ')

f.read() # Lê o resto do arquivo
# 'escrevendo no arquivo\\nOutra linha escrita no arquivo\\n'
```

Podemos ver que o método `read()` retorna as novas linhas como \\n. Uma vez que o final do arquivo é alcançado, nós recebemos uma string vazia se continuarmos tentando ler. Nós podemos alterar o cursor (posição) do arquivo atual usando o método `seek()`, além disso, o método `tell()` retorna a nossa posição atual (em números de bytes).

Outra forma de lermos arquivos é utilizando o método `readline()`, que lerá cada linha individualmente. Esse método lerá até a nova linha, incluindo o caracter de nova linha \\n:

```
f.readline() # 'Estamos escrevendo no arquivo\\n'

f.readline() # 'Outra linha escrita no arquivo\\n'
```

Por fim temos o método `readlines()` que retorna uma lista das linhas restantes de todo o arquivo. Todos esses métodos de leitura retornam válores vazios quando alcançamos o fim do arquivo (EOF).

```
f.readlines()
# ['Estamos escrevendo no arquivo\\n', 'Outra linha escrita no arquivo\\n']
```

Anexando e Deletando Arquivos

Anexando a um Arquivo

Às vezes, podemos querer anexar a um arquivo ou começar a escrever no final de um arquivo já preenchido. Isso é feito facilmente usando o caractere 'a' para o argumento de modo, por exemplo:

```
with open("teste.txt", "a") as writer:
    writer.write('Anexando ao arquivo')
```

Se formos examinar nosso arquivo teste.txt, veremos que o início do arquivo permanece inalterado, porém se olharmos para o fim dele, nosso conteúdo foi adicionado:

```
with open("teste.txt", "r") as reader:
    print(reader.read())
```

Nos será impresso o seguinte conteúdo:

```
Estamos escrevendo no arquivo
Outra linha escrita no arquivo
Anexando ao arquivo
```

Deletando um Arquivo

Digamos que queremos usar o Python para remover um arquivo de nosso computador, para isso, podemos contar com a ajuda do módulo **os** que nos traz funcionalidades do sistema operacional.

O método `remove()` é capaz de deletar um arquivo, para usá-lo precisamos lembrar de importar o módulo os:

```
import os

os.remove("arquivo.txt")
```

Para evitarmos erros, podemos checar se o arquivo exista antes de deletá-lo:

```
import os

if os.path.exists("arquivo.txt"):
    os.remove("arquivo.txt")
else:
    print("Arquivo não existe")
```

Se eventualmente quisermos remover um diretório, podemos usar o método `rmdir()`:

```
import os

os.rmdir("documentos")
```

Perceba que você pode apenas remover diretórios vazios.

Métodos de Arquivos em Python

A tabela a seguir nos apresenta uma lista completa dos métodos de arquivos disponíveis em Python, bem como suas descrições.

Métodos	Descrição
close()	Fecha um Arquivo
detach()	Retorna o fluxo bruto separado do buffer
fileno()	Retorna um número que representa o fluxo, através da perspectiva do Sistema Operacional
flush()	Libera o buffer interno
isatty()	Retorna se o fluxo do arquivo é interactivo ou não
read()	Retorna o conteúdo do Arquivo
readable()	Retorna se o fluxo do arquivo pode ser lido ou não
readline()	Retorna uma linha do Arquivo
readlines()	Retorna uma lista de linhas do Arquivo
seek()	Muda a posição do Arquivo
seekable()	Retorna se o Arquivo nos permite mudar a posição do Arquivo
tell()	Retorna a posição do Arquivo atual
truncate()	Altera o tamanho do Arquivo para um tamanho especificado
writable()	Retorna se o Arquivo pode escrito ou não
write()	Escreve a string especificada em um Arquivo
writelines()	Escreve uma lista de strings em um Arquivo

Arquivos CSV

Um arquivo **CSV (Comma Separated Values)** é um tipo de arquivo de texto simples que usa uma estrutura específica para organizar dados tabulares. Por ser um arquivo de texto simples, ele pode conter apenas dados de texto reais, em outras palavras, caracteres ASCII ou Unicode imprimíveis.

A estrutura de um arquivo CSV é dada pelo seu nome. Normalmente, arquivos CSV utilizam vírgula para separar cada valor específico de dados. Vejamos o exemplo do arquivo livros.csv abaixo para entendermos melhor:

```
Title,Author,Genre,Height,Publisher
Fundamentals of Wavelets,"Goswami, Jaideva",signal_processing,228,Wiley
Data Smart,"Foreman, John",data_science,235,Wiley
God Created the Integers,"Hawking, Stephen",mathematics,197,Penguin
Superfreakonomics,"Dubner, Stephen",economics,179,HarperCollins
Orientalism,"Said, Edward",history,197,Penguin
"Nature of Statistical Learning Theory, The","Vapnik, Vladimir",data_science,230,Springer
Integration of the Indian States,"Menon, V P",history,217,Orient Blackswan
"Drunkard's Walk, The","Mlodinow, Leonard",science,197,Penguin
Image Processing & Mathematical Morphology,"Shih, Frank",signal_processing,241,CRC
```

Perceba que cada valor de dados é separado por uma vírgula. Normalmente a primeira linha identifica cada valor dos dados, em outras palavras, o nome da coluna dos dados. De modo geral, o caracter separador é chamado de delimitador e a vírgula não é o único utilizado. Outros delimitadores incluem os caracteres tab (\\t), dois pontos (:) e ponto e vírgula (;). Para acessarmos os dados de um arquivo CSV de forma correta é necessário que saibamos com qual delimitador estamos trabalhando.

Lendo Arquivos CSV

A **biblioteca csv** nos fornece a funcionalidade de lermos arquivos CSV e também escrever. Desenvolvida para trabalharmos diretamente com arquivos CSV gerados por Excel, pode ser facilmente adaptável para trabalharmos com uma variedade de formatos CSV.

Método reader

O Método `reader()` nos retorna um objeto _csv.reader iterável que podemos percorrer através de um for loop:

```
import csv

with open('livros.csv', newline='') as csv_file:
    csv_reader = csv.reader(csv_file)
    for linha in csv_reader:
        print(f"{linha[0]},{linha[1]}")
```

Nos é retornado o resultado:

```
Title,Author
Fundamentals of Wavelets,Goswami, Jaideva
Data Smart,Foreman, John
God Created the Integers,Hawking, Stephen
Superfreakonomics,Dubner, Stephen
Orientalism,Said, Edward
Nature of Statistical Learning Theory, The,Vapnik, Vladimir
Integration of the Indian States,Menon, V P
Drunkard's Walk, The,Mlodinow, Leonard
Image Processing & Mathematical Morphology,Shih, Frank
```

Perceba que estamos imprimindo apenas as duas primeiras colunas (Title e Autor).

Método DictReader

O Método `DictReader()` nos retorna um objeto csv.DictReader que podemos iterar e percorrer os valores através de um for loop:

```
import csv

with open('book.csv', newline='') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    for linha in csv_reader:
        print(f"{linha['Title']} | {linha['Author']} | {linha['Genre']} | {linha['Publisher']}")
```

Nos é retornado o seguinte resultado:

```
Fundamentals of Wavelets | Goswami, Jaideva | signal_processing | Wiley
Data Smart | Foreman, John | data_science | Wiley
God Created the Integers | Hawking, Stephen | mathematics | Penguin
Superfreakonomics | Dubner, Stephen | economics | HarperCollins
Orientalism | Said, Edward | history | Penguin
Nature of Statistical Learning Theory, The | Vapnik, Vladimir | data_science | Springer
Integration of the Indian States | Menon, V P | history | Orient Blackswan
Drunkard's Walk, The | Mlodinow, Leonard | science | Penguin
Image Processing & Mathematical Morphology | Shih, Frank | signal_processing | CRC
```

Escrevendo em Arquivos CSV

Nós podemos escrever em um arquivo CSV utilizando o objeto writer ou DictWriter e o método `write_row()`.

Vejamos um exemplo.

```
import csv

with open('livros.csv', newline='') as csv_file:
    csv_reader = csv.DictReader(csv_file)

    with open('livros_novos.csv', 'w') as novo_arquivo:
        fieldnames = ['Title', 'Author', 'Genre']
        csv_writer = csv.DictWriter(novo_arquivo, fieldnames=fieldnames, delimiter=';')
        csv_writer.writeheader()

        for linha in csv_reader:
            del linha['Height']
            del linha['Publisher']
            csv_writer.writerow(linha)
```

- A variável fieldnames especifica as colunas que vamos utilizar em nosso novo arquivo
- O construtor DictWriter recebe como parâmetro nosso novo_arquivo, os fieldnames e o delimitador que nesse caso estamos utilizando (;).
- O método `writeheader()` escreve no arquivo livros_novos.csv o nosso novo cabeçalho
- Através do for nós percorremos os dados do arquivo livros.csv e deletamos as colunas Height e Publisher usando a palavra-chave del
- Finalizamos escrevendo no novo arquivo livros_novos.csv através do método `writerow()`

Por fim, obtivemos um novo arquivo livros_novos.csv com o seguinte conteúdo:

```
Title;Author;Genre
Fundamentals of Wavelets;Goswami, Jaideva;signal_processing
Data Smart;Foreman, John;data_science
.....
```