Universidade Federal do Vale do São Francisco

Minicurso de Linguagem Python

Edjair Aguiar Gomes Filho – Representante Discente pelo Centro Acadêmico de Engenharia da Computação

Aula 5 – Manipulando arquivos texto com Python

Sumário

1) INTRODUÇÃO

É muito comum que nos nossos estudos iniciais trabalhemos com dados inseridos diretamente no código ou digitados pelo usuário à medida que são lidos. Entretanto, quanto mais avançamos no desenvolvimento de aplicações, percebemos que em determinado momento nos é exigido a manipulação de dados que já estejam *armazenados em algum lugar*.

O que isso significa? É provável que você tenha pensado em algo mais elaborado, como um banco de dados, mas é muito mais comum trabalharmos com um tipo de armazenamento de dados mais "primitivo": os *arquivos*. E quando falamos de arquivos, estamos falando de diversos tipos de arquivos: uma imagem usada para identificação facial, um vídeo, o *captcha* que nos aparece quando votamos no BBB...

Para os nossos propósitos, vamos trabalhar com arquivos de dados que contém textos. Os arquivos de texto são a forma mais primária e manipulável para armazenarmos e recuperarmos dados com facilidade e integridade de informações. E os arquivos texto são justamente aqueles que mais estão presentes no nosso cotidiano: escrevemos programa, escrevemos dados extraídos por um sensor, baixamos dados de uma página na internet, baixamos até mesmo uma página da internet – isso tudo utilizando arquivos de texto. Não importa como os arquivos são criados, o Python nos permite manipular seu conteúdo.

2) ABRINDO E FECHANDO UM ARQUIVO

Podemos afirmar que abrir e fechar são as operações mais importantes que aplicaremos a um arquivo. Beleza, entendi, mas por quê?

Quando você abre um arquivo no Word, seu computador imediatamente puxa o arquivo que está salvo no seu disco rígido. Porém imagine que o computador deixe o arquivo aberto o tempo inteiro no Word e, durante todo, ao mesmo tempo que você navega pelo Reddit, usa o Twitter, escuta uma música e baixa algum jogo para relaxar depois. Com todos esses arquivos abertos e sendo processados o tempo inteiro, a chance dos dados se misturarem acaba sendo muito grande.

E como funciona? O sistema operacional *precisa* garantir que todos os seus arquivos estarão íntegros, cada qual com seu conteúdo. Para isso acontecer, cada aplicativo precisa dizer quais são suas intenções com os arquivos que estão no HD: quando você põe uma música para tocar, seu player avisa que ele quer apenas *ler* o arquivo de música. Quando você abre um arquivo no Word para finalizar seu trabalho, o Word avisa que vai *escrever* no arquivo. Vê só o que acontece: o arquivo é aberto, lido, copiado para o Word e fechado imediatamente – agora ele só será aberto novamente quando você for salvar. Dessa forma, o computador garante que nenhum dado vai se misturar ou corromper.

Da mesma forma faremos em Python. Devemos **abrir** nossos arquivos antes de usá-los e **fechá-lo** depois que já tivermos feito o que queríamos fazer. Para abrir e fechar um arquivo, utilizados os métodos *open()* e *close()*.

Como todo método, precisamos passar o parâmetro que vamos utilizar. Quando queremos fechar um arquivo, apenas dizemos qual o nome do arquivo. Mas quando vamos abrir, precisamos atribuí-lo a uma variável, passar o nome do arquivo e especificarmos o que queremos fazer com ele. Essa especificação é feita com as letras “r”, “w” e “a”.

Especificação	Significado	O que faz
r	read (ler)	Usa o arquivo *apenas* para leitura.
w	write (escrever)	Usa o arquivo para leitura e escrita. Porém quando usamos write, estamos criando um arquivo limpo, novo, do zero. Se o arquivo já tinha algo escrito, é tudo apagado e substituído.
a	append (anexar)	É semelhante ao w, porém tudo o que você escreve é *anexado* ao arquivo – ou seja, você pode escrever, tudo que é novo vai para o final e tudo que estava antes será conservado.

Quando falamos que o arquivo deve ser atribuído a uma variável, é porque além de facilitar a manipulação no código, depois de aberto o arquivo passa a ser um objeto Python como qualquer outro. A Figura 1 mostra um exemplo de código que abre e fecha um arquivo qualquer.

```
x = open(nome_do_arquivo, 'r') #Abrimos o arquivo para leitura e o armazenamos na variável x

#Manipulamos o arquivo ao sabor do vento

x.close #Acabou, fecha o arquivo
```

Figura 1 – Exemplo simples: abrir e fechar um arquivo

Podemos também escrever isso de maneira genérica:

variavel = open(‘arquivo’, ‘r’)	Abre um arquivo apenas de leitura e o armazena na variável chamada *variavel*.
variavel = open(‘arquivo’, ‘w’)	Abre um arquivo para escrita e o armazena na variável chamada *variavel*.
variavel.close	Fecha o arquivo que está armazenado na variável chamada *variavel*.

3) ENCONTRANDO UM ARQUIVO

Como você já deve ter imaginado, para abrir um arquivo você e o Python precisam fazer um acordo sobre onde está o seu arquivo. Poderíamos no prolongar sobre esse assunto, mas na verdade ele é muito simples. Basta que especifiquemos o caminho do arquivo, porém utilizando as barras tradicionais (/) Ao invés das barras invertidas (\). Por exemplo, se tivermos um arquivo cujo caminho é **C:\Documents\Minicurso Python\summereletrohits.txt**, escrevemos o open da seguinte forma:

open(‘/Documents/Minicurso Python/summereletrohits.txt’, ‘r’)

4) LENDO ARQUIVOS TEXTO

Vamos supor que temos um arquivo de texto chamado *linhas.txt*, salvo na mesma pasta do nosso código Python. O conteúdo do arquivo é o seguinte:

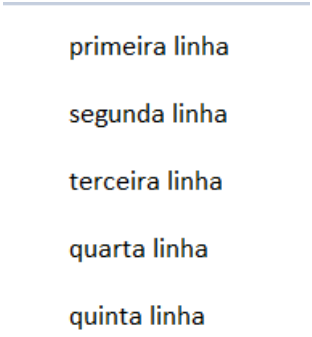

Figura 2 – Criamos um arquivo chamado linhas.txt

Uma **linha** de um arquivo é definida como uma sequência de caracteres que vai até um caractere especial, chamado de **newline**. Lembra do *\n* que a gente coloca nas strings para quebrar a linha? Pois é, esse é o newline. Toda vez que uma linha termina, ele vai estar lá, marcando presença.

4.1) Função read()

Podemos abrir um arquivo e ler todo o conteúdo do arquivo em uma única string através da função read(). Suponhamos o programa Python da Figura 3, que chamaremos de *read.py*.

```
x = open('/Users/USUARIO/Documents/linhas.txt', 'r')
unica_string = x.read()
x.close()

print(unica_string)
```

Figura 3 – read.py

Quando executarmos esse código, obteremos como resultado:

```
C:\Users\USUARIO>C:\Users\USUARIO\Documents\read.py
primeira linha
segunda linha
terceira linha
quarta linha
quinta linha
```

Figura 4 – Resultado da execução do código read.py

4.2) Função readline()

A função *readline()* lê linha a linha do arquivo. É importante frisar que ao chamarmos a readline() pela segunda vez, ela já lê imediatamente a próxima linha. Ela descobre que o arquivo acabou quando encontra o caractere especial eof – end of file – e quando o encontra devolve o vazio, o nada, o nadinha.

```
x = open('/Users/USUARIO/Documents/linhas.txt', 'r')

unica_string = x.readline()
print(unica_string)
unica_string = x.readline()
print(unica_string)
unica_string = x.readline()
print(unica_string)
unica_string = x.readline()
print(unica_string)
unica_string = x.readline()
print(unica_string)
unica_string = x.readline()
print(unica_string)

x.close()
```

Figura 5 – read.py porém agora utilizando o readline()

Quando executarmos esse código, obteremos como resultado:

```
C:\Users\USUARIO>C:\Users\USUARIO\Documents\read.py
primeira linha

segunda linha

terceira linha

quarta linha

quinta linha
```

Figura 6 – Resultado da execução do novo código read.py

5) ESCREVENDO EM ARQUIVOS TEXTO

Para abrirmos um arquivo para leitura usando o modo r, precisamos garantir que o arquivo existe; entretanto quando criamos um arquivo para escrita usando o modo w, mesmo que o arquivo não exista, ele será criado. Vamos compreender isso direitinho.

5.1) Criando um arquivo vazio

Começaremos escrevendo um código em Python que solicita a abertura de um arquivo no modo de escrita. Chamaremos de *novo_arquivo.txt* e eu quero que ele seja criado na minha pasta Documents. Isso é mostrado na Figura 7 abaixo.

```
x = open('/Users/USUARIO/Documents/novo-arquivo.txt', 'w')
x.close
```

Figura 7 – Solicitamos a abertura de um novo arquivo no modo w

Chamamos esse código de *cria.py* e o executamos normalmente.

```
C:\Users\USUARIO>C:\Users\USUARIO\Documents\cria.py
```

Figura 8 – Executamos o código cria.py normalmente

Nesse ponto eu friso que esse arquivo não existia no meu computador. Porém, quando eu executo esse código Python, um novo arquivo em branco foi criado na pasta especificada.

Figura 9 - O Python criou um novo arquivo no meu computador. Como não escrevi nada nele, vemos que ele está em branco, ressaltado pelo seu tamaho de 0 KB.

[!] OBSERVAÇÃO: Se na Figura 7 nós tivéssemos especificado um caminho para um arquivo que já existia, o conteúdo do arquivo teria sido completamente apagado. Isso aconteceria porque, como já dito, o modo w reescreve o arquivo do zero. Quando abrimos um arquivo já existente no modo w e não escrevemos nada, seu conteúdo é substituído por esse nada.

5.2) Função write() e writelines()

Agora que nosso arquivo já está criado, vamos escrever nele. Para isso, utilizamos a função write(), passando como parâmetro o que nós queremos escrever. Vamos chamar esse programa de *escreve.py*.

```
x = open('/Users/USUARIO/Documents/novo-arquivo.txt', 'w')
x.write('Fica em casa, namoral')
x.close
```

Figura 10 – Nosso programa escreve.py

Executamos nosso código normalmente:

```
C:\Users\USUARIO>C:\Users\USUARIO\Documents\escreve.py
```

Figura 11 – Executando escreve.py

Quando abrirmos o nosso arquivo *novo-arquivo.txt*, veremos que uma nova linha foi escrita nele.

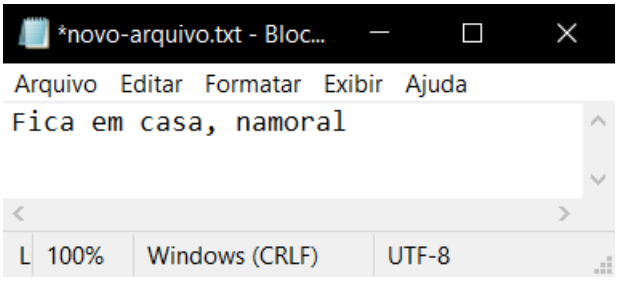

Figura 12 – Nosso arquivo foi escrito!

A diferença da função write() para a writelines() é que a writelines() permite a quebra de linha entre várias escritas seguidas.

[!] OBSERVAÇÃO: Lembre-se sempre: se o arquivo já existisse, todo o conteúdo dele seria substituído pelo que escrevemos. Experimente isso tentando agora escrever novas coisas no arquivo novo-arquivo.txt e observe como todas vezes o conteúdo anterior é completamente substituído pelo novo conteúdo.

5.3) Função append()

E se eu não quiser substituir o conteúdo inteiro?

Podemos utilizar a função append() para acrescentar uma nova linha ao nosso arquivo sem perder o que foi escrito anteriormente. Vamos criar um programa Python chamado *acrescenta.py* igual ao da Figura 13.

```
x = open('/Users/USUARIO/Documents/novo-arquivo.txt', 'r')
conteudo = x.readlines()
x.close

conteudo.append('NAQDWAJOIFJAFA')

x = open('/Users/USUARIO/Documents/novo-arquivo.txt', 'w')
x.writelines(conteudo)
x.close
```

Figura 13 – Programa em Python acrescenta.py

Vamos avaliar o que esse programa faz, linha por linha:

i) Abre o arquivo em modo de leitura e o armazena na variável x;
ii) Criamos outra variável, conteudo, e armazenamos nela tudo o que estava escrito no arquivo anteriormente;
iii) Fechamos o arquivo de leitura;
iv) Usamos a função append() para anexar uma nova linha ao conteúdo salvo;
v) Abrimos o arquivo novamente, dessa vez em modo de escrita;
vi) Usamos o write() para escrever o conteúdo no arquivo.
vii) Fechamos o arquivo após seu uso.

Executamos nosso código normalmente.

```
C:\Users\USUARIO>C:\Users\USUARIO\Documents\acrescenta.py
```

Figura 14 – Executando acrescenta.py

Quando abrirmos nosso arquivo, veremos que nosso anexo foi acrescentado, preservando o conteúdo original.

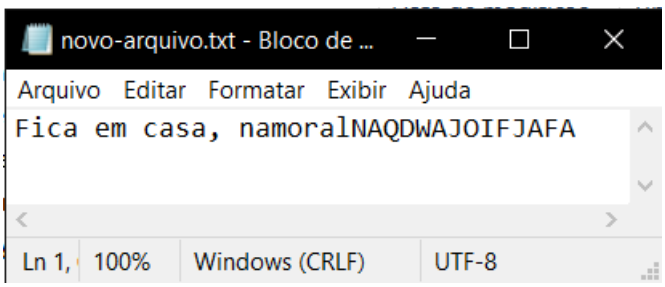

Figura 15 – Nosso arquivo, com conteúdo preservado e novas lindas palavras