FileSystem

Um dos principais usos das linguagens de programação desde o começo da ciência da computação é trabalhar com arquivos. A escrita e leitura em disco é bastante simples de ser feita em python, também podemos listar pastas, subpastas, e os arquivos presentes nestas pastas, nesta aula vamos focar em como trabalhar com o sistema de arquivos do sistema operacional, e como podemos utiliza-lo para escrever nossos programas.


Importante! Quando estivermos trabalhando com arquivos e pastas em python, é importante lembrar que todos os caminhos de diretórios consideram o diretório que está sendo executado no terminal, tenha isso em mente para possíveis erros. Ex: se o meu prompt de comando está aberto em c:/minha_pasta, os arquivos que serão pesquisados nos métodos abaixo deverão estar dentro de c:/minha_pasta.

Para abrir um arquivo dentro de um script python é bastante simples, podemos utilizar a expressão with open(caminho, modo), onde caminho é o caminho até o arquivo desejado, e o modo é apenas leitura ("r") ou leitura, gravação ("w") ou adição ("a").

```
with open('data.txt', 'r') as file:
        data = file.read()
        print(data)
```

O script acima está abrindo um arquivo com o nome data.txt, em formato somente leitura. Em seguida, é realizada a leitura do conteúdo do arquivo utilizando o método read(), que retorna para a variável data, o conteúdo, neste caso textual, do arquivo file. O print dessa variável vai exibir o conteúdo textual do nosso arquivo.

Se quisermos abrir um arquivo e editar o conteúdo dele, podemos utilizar uma sintaxe bem semelhante:

```
with open('data.txt', 'w') as file:
        text = "Meu novo texto."
        file.write(text)
```

Ok, mas ainda não ficou claro porque precisamos utilizar a palavra chave with ao abrir um arquivo, não podemos simplesmente utilizar open() e salvar o conteúdo em uma variável. Existe um motivo, e ele é bastante simples: o escopo, ao utilizar with, nós garantimos que estamos trabalhando com os dados daquele arquivo dentro daquele bloco, e assim que o bloco é fechado, o coletor de lixo da linguagem pode esvaziar a memória alocada para os dados do arquivo, evitando assim também problemas de gravação e etc.

Lendo um poema

Imagine o arquivo poema.txt com o seguinte conteúdo:

```
Em um poema era uma vez
Na linha um existia o número 1
Na linha dois existia o número 2
E Na linha três existia o número 3
```

O poema acima está salvo em um arquivo de texto onde as linhas estão muito bem delimitadas, seria interessante ler este arquivo linha por linha para que o poema fosse melhor analisado. Felizmente em python é bastante simples separar conteúdo de arquivos linha por linha, vamos aprender como.

Para ler uma linha de um arquivo, podemos utilizar o método readline(linha):

```
with open("poema.txt") as poema:
        line = poema.readline()
        print(line) # Em um poema era uma vez
        next_line = poema.readline()
        print(next_line) # Na linha um existia o número 1
```

Por padrão, o método readline retorna sempre a primeira linha do arquivo, no entanto podemos fornecer como argumento uma linha específica que desejamos ler. É bastante comum também ler todas as linhas de um arquivo de uma vez só, para isso temos um outro método bastante semelhante, o readlines(), que retorna uma lista com todas as linhas do documento.

```
with open("poema.txt") as poema:
        lines = poema.readlines()
        print(lines)
        # [ Em um poema era uma vez, Na linha um existia o número 1,
        # Na linha dois existia o número 2, E Na linha três existia o número 3]
```

No entanto é mais fácil percorrer todas as linhas do documento utilizando um for como no exemplo abaixo:

```
with open("poema.txt") as poema:
        for line in poema:
                print(line)
```

Trabalhando com pastas

Para obter a lista de pastas no diretório atual basta utilizar o método scandir() do objeto os. É necessário importar os para acessar este método. O uso é bastante simples, podemos utilizar o gerenciador de contexto with, juntamente com o método scandir() como uma coleção de entradas de diretórios e arquivos da pasta, com isso basta iterar utilizando um for para obter informações de cada arquivo.

```
import os
with os.scandir() as itens:
        for item in itens:
                print(item.name)
```

Ok, mas queremos saber se o item em questão é um arquivo ou uma pasta, para atingir este objetivo vamos precisar utilizar o método is_file(), que retorna verdadeiro caso o item em questão seja um arquivo, e falso caso seja uma pasta.

```
import os
with os.scandir() as itens:
        for item in itens:
                if item.is_file():
                        print(item.name)
```

Se por algum motivo você prefere verificar se a entrada é uma pasta, basta usar o método is_dir():

```
import os
with os.scandir() as itens:
        for item in itens:
                if item.is_dir():
                        print(item.name)
```

Existe ainda uma gama de métodos de **os** que podem ser utilizados para trabalhar com o sistema de arquivos do seu computador, nesta aula focamos apenas nos principais, mas recomendamos que você não limite a sua curiosidade, e busque aprender os demais métodos presentes nesta biblioteca.