# Uso de Arquivos em Python

Professor Dr. Ricardo Ferreira de Oliveira

Estudaremos neste texto a criação, o uso e modificação de arquivos na linguagem de programação Python. Iniciamos nosso estudo apresentando a definição de arquivos ilustrando alguns dos tipos mais comuns de arquivos. Apresentamos a sintaxe e as formas de manipulação de arquivos em Python. Passamos a apresentar as formas de manipulação de arquivos exemplificando com casos práticos concretos apresentando três tipos de arquivos para trabalhar em Python.

Desta forma, iniciaremos com algoritmos simples de manipulação e caminhamento de arquivos partindo para exemplos de usos de arquivos mais sofisticados.

## Arquivos

Comumente, arquivos consistem de informações organizadas, nomeadas e armazenadas em alguma mídia de dados. Os arquivos podem ter vários tipos de extensão indicando o tipo de arquivos a que se refere. Neste roteiro iremos lidar com três tipos de extensão para exemplificação: ‘.txt’, ‘.csv’ e ‘.pdb’.

Os arquivos com extensão ‘.txt’ são normalmente arquivos contento texto sem formatação. São usados frequentemente para troca de dados simples, envio de parâmetros ou onde a indexação não se faz necessária. Os arquivos com extensão ‘.csv’ são arquivos onde os seus campos são separados por vírgula ou ponto-e-vírgula. CSV é uma sigla que significa ‘*Comma-separated values*’ e é um arquivo de formato simples bastante utilizado em planilhas eletrônicas e SGBD disponibilizados no mercado. Os arquivos ‘.pdb’ são arquivos usados em biologia molecular e bioinformática e ilustram um tipo de arquivo com informações estruturadas a partir de sua posição dentro de cada registro.

As operações típicas com arquivos consistem de leitura ou escrita, caminhamento e fechamento.

## Arquivos em Python

Comumente, arquivos consistem de informações organizadas, nomeadas e armazenadas em alguma mídia de dados. Existem vários tipos de formatos de arquivos. Estes formatos são definidos para que aplicações possam reconhecer os dados que estão armazenados nos arquivos. Os formatos de arquivos digitais podem ser arquivos de texto, arquivos de imagem, arquivos de som, arquivos de vídeo, arquivos de bancos de dados, arquivos com código executável, etc. Alguns formatos são considerados padrões de mercado. Exemplo deste tipo de arquivo são os arquivos:

* ‘.csv’- valores separados por vírgula ou ponto-e-vírgula.
* ‘.xlx’ – arquivos de planilhas Excel, mas lido por outros softwares compatíveis
* ‘.doc’ – arquivos para documentos de texto
* ‘.txt’ – arquivos com texto comum
* ‘.pdf’ – arquivos com formato Acrobat Adobe
* ‘.pdb’ – arquivos com átomos de proteínas do Protein Data Bank

Os arquivos podem ser abertos de quatro formas distintas em Python:

* r – para leitura (forma padrão)
* w – para escrita
* a – para adição
* b – em formato binário

## 

## Abrindo arquivos em Python

Podemos abrir um arquivo para leitura ou escrita em Python. Para isso, utilizamos uma variável que será usada como *handle* do arquivo, ou seja, um objeto que conterá, entre outros atributos, o nome do arquivo para referência durante as operações de manipulação. Além do nome do *handle*, necessitamos o caminho onde o arquivo se encontra e também a especificação de que se está abrindo o arquivo para leitura. Muitas vezes é necessário especificar também o encoding (tipo de conjunto de caracteres) como por exemplo: ‘utf-8’, ‘latin-1’ ou ‘iso-8859-1’.

A forma mais simples de abertura de arquivos em Python é:

|  |
| --- |
| handle = open( “arquivo.txt”, “r”) |

onde handle é a variável que será usada para manusear o arquivo, ‘arquivo.txt’ é o nome do arquivo no sistema operacional que pode incluir o seu caminho e ‘r’ um parâmetro que indica que o arquivo está sendo aberto para leitura.

A Tabela 1 ilustra algumas das principais formas de sintaxe para abertura de arquivos no modo de leitura.

**Tabela 1 – Formas de sintaxe para abertura de arquivos para leitura**

|  |  |
| --- | --- |
| Forma | Descrição |
| handle = open( “arquivo.txt”, “r” ) | Abertura simples em modo leitura |
| handle = open( “c:/diretorio/arquivo.txt”, “r” ) | Abertura de arquivo para leitura especificando caminho completo. Caso seja usada a outra barra, é necessário usar duas barras por vez: = open( “c:\\diretorio\\arquivo.txt”, “r” ) |
|  |  |
| handle = open( “arquivo.txt”, “r”, encoding=”utf-8” ) | Abertura de arquivo no modo leitura usando encoding utf-8. |
| handle = open( “arquivo.txt”, “rb” ) | Abertura de arquivo binário em modo leitura |

**Fonte: próprio autor**

Para abrir um arquivo no modo escrita usamos a seguinte sintaxe:

|  |
| --- |
| handle = open( “arquivo.txt”, “w”) |

Esta forma irá criar o arquivo com nome “arquivo.txt” e prepará-lo para que conteúdo seja escrito nele. Caso já exista um arquivo com este nome, ele será sobrescrito por esta operação. Muitas vezes queremos abrir um arquivo para escrita e continuar a escrever no arquivo sem apagar o conteúdo anterior.

Neste caso a forma de abertura do arquivo é como se segue:

|  |
| --- |
| handle = open( “arquivo.txt”, “a”) |

Abrindo com a opção “a” (append) podemos continuar a escrever no arquivo sem perder o conteúdo já gravado. A Tabela 2 ilustra algumas das principais formas de sintaxe para abertura de arquivos no modo de leitura.

**Tabela 2 – Formas de sintaxe para abertura de arquivos para escrita**

|  |  |
| --- | --- |
| Forma | Descrição |
| handle = open( “arquivo.txt”, “w” ) | Abertura simples em modo escrita |
| handle = open( “c:/diretorio/arquivo.txt”, “w” ) | Abertura de arquivo para escrita especificando caminho completo. Caso seja usada a outra barra, é necessário usar duas barras por vez: = open( “c:\\diretorio\\arquivo.txt”, “w” ) |
|  |  |
| handle = open( “arquivo.txt”, “w”, encoding=”utf-8” ) | Abertura de arquivo no modo escrita usando encoding utf-8. |
| handle = open( “arquivo.txt”, “a” ) | Abertura de arquivo para continuar escrevendo no arquivo (o conteúdo anterior não será apagado). |
| handle = open( “arquivo.txt”, “wb” ) | Abertura de arquivo binário em modo escrita |

**Fonte: próprio autor**

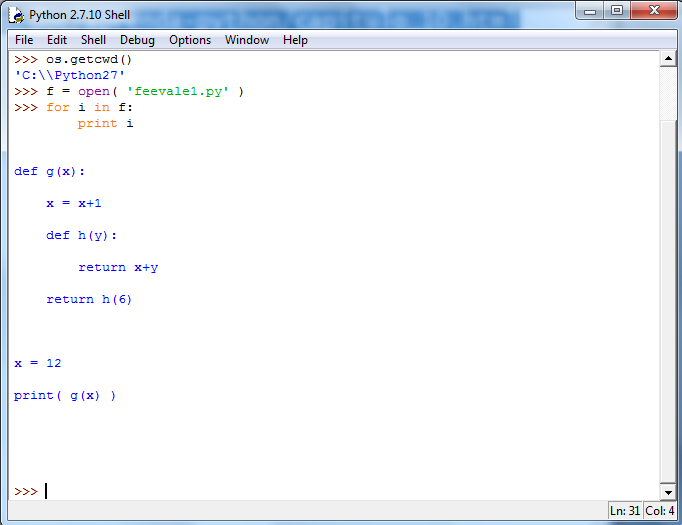
Uma vez aberto o arquivo, operações de escrita ou leitura podem ser realizadas e ao fim do trabalho com o arquivo, o mesmo deve ser fechado invocando o método close() a partir do handle usado para aquele arquivo:

|  |
| --- |
| handle.close() |

## Percorrendo um arquivo

Existem várias formas de navegar nos arquivos em Python. Uma das formas mais comuns é usar o comando de iteração for para percorrer o arquivo linha a linha. A figura 1 ilustra um pequeno código onde um fonte (‘feevale.py’) é aberto para leitura (modo default) e é percorrido com o comando for. Repare que a cada volta, a variável de iteração recebe uma linha do arquivo.

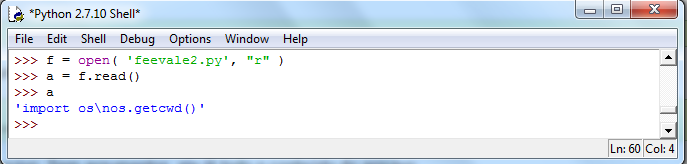
**Figura 1 – Percorrendo um fonte Python linha a linha com for**



**Fonte: próprio autor**

O Python possibilita que todo o arquivo seja lido de uma vez só guardando o conteúdo em uma variável. Esta variável armazenará uma string contendo todo o conteúdo do arquivo. Fazemos isto chamando a função read() sem nenhum argumento. Observe que, caso se percorra esta variável com um comando for, o arquivo será percorrido caracter a caracter e não linha a linha. A figura 2 ilustra um pequeno código onde um arquivo (‘feevale2.py’) é lido todo de uma vez só. Observe o ‘\n’ mostrando uma quebra de linha.

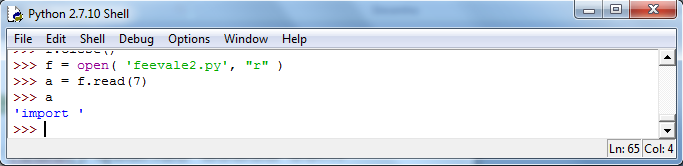
**Figura 2 – Lendo todo o arquivo de uma vez só para uma string**



**Fonte: próprio autor**

Podemos fornecer valores como argumento para a função read(). Neste caso, o valor do argumento é o número de bytes lidos do arquivo. A leitura é interrompida naquele ponto de modo que, caso sejam lidos mais bytes a leitura parte dali. A figura 3 ilustra um pequeno código onde somente 7 bytes de um arquivo (‘feevale2.py’) são lidos.

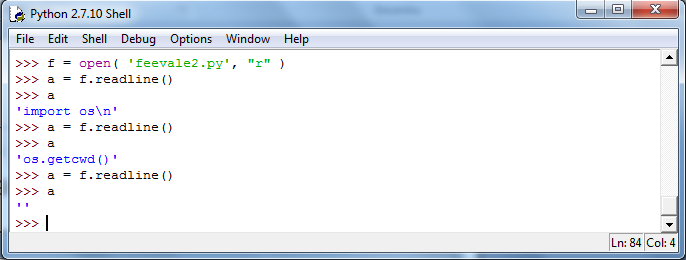
**Figura 3 – Lendo 7 bytes de um arquivo**



**Fonte: próprio autor**

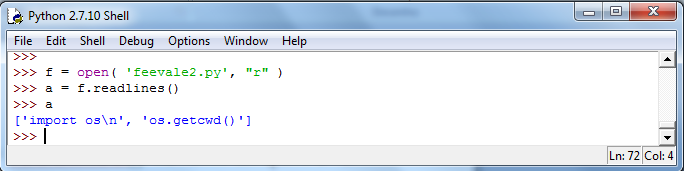
A função readline() lê uma linha do arquivo e se posiciona para ler a proxima linha. Podemos ler um arquivo linha a linha invocando a cada linha a função readline(). A figura 4 ilustra a leitura de um arquivo linha a linha usando a função readline(). Já a função readlines() lê todas a linhas de um arquivo e cria uma lista contendo cada linha como elemento. A figura 5 ilustra a leitura das linhas de um arquivo usando a função readlines(). Podemos então, navegar pelas linhas do arquivo bastando para isso, percorrer a lista elemento por elemento usando, por exemplo, um comando for.

**Figura 4 – Lendo um arquivo linha a linha com readline()**



**Fonte: próprio autor**

**Figura 5 – Criando uma lista de linhas com a função readlines()**



**Fonte: próprio autor**

Nos próximos materiais:

* “TrabalhandoComArquivosTextoEmPython.pptx”
* “**LendoEescrevendoCSVEmPython.pptx**”

iremos exemplificar, na prática, exemplos de uso de arquivos comumente usados. O primeiro exemplo tratará do percurso e manipulação de um arquivo texto. O segundo exemplo irá trabalhar a leitura e escrita em arquivos CSV. Até lá!

Referências

BARRY, Paul. Use a cabeça: Python. Rio de Janeiro, RJ: Alta Books, 2012. xxxiv, 457 p. (Use a cabeça) ISBN 9788576087434

LUTZ, Mark; ASCHER, David. Aprendendo Python. [2. ed.] Porto Alegre, RS: Bookman, 2008. xvii, 566 p. ISBN 9788577800131.

MCKINNEY, Wes. Python para análise de dados: tratamento de dados com pandas, numpy e ipython. São Paulo, SP: Novatec, c2018. 615 p. ISBN 9788575226476.

PERKOVIC, Ljubomir. Introdução à computação usando Python um foco no desenvolvimento de aplicações. Rio de Janeiro LTC 2016 1 recurso online ISBN 9788521630937.