Arquivos

Arquivos – conceitos iniciais

Um arquivo de computador é um recurso de armazenamento de dados que está disponível em todo tipo de dispositivo computacional, seja um computador, dispositivo móvel, uma câmera fotográfica, entre outros. Os arquivos são utilizados para armazenar dados de maneira permanente, e para isso devem ser gravados em algum equipamento de hardware que não necessite estar ligado permanentemente em uma fonte de energia. Nos tempos atuais, os equipamentos mais utilizados para esse fim são os discos magnéticos e os chips SSD.

O gerenciamento do armazenamento fica a cargo do sistema operacional (SO) instalado no dispositivo computacional. Isso permite maiores segurança, padronização e organização das unidades de armazenamento.

As linguagens de programação, por sua vez, contam com comandos e recursos próprios para realizar as operações de gravação e leitura dos arquivos. No entanto, não é a linguagem, qualquer que seja, que fará o acesso físico ao hardware. O que os comandos da linguagem fazem para manipular arquivos em disco é colocar em execução um conjunto de funções que fazem parte do sistema operacional. Assim sendo, o programador, ao utilizar os comandos necessários para efetuar operações de gravação e leitura, está indiretamente acessando serviços do SO.

Um grande benefício dessa abordagem para o programador é que ela permite separar, de um lado, os detalhes de implementação do sistema de arquivos que o SO utiliza e, de outro, os comandos da linguagem que o programador deve utilizar para executar as operações.

Uma vez gravado em disco, o arquivo pode ser entendido como um conjunto de bytes ao qual é atribuído um nome. Quando esse arquivo voltar a ser lido, haverá duas possíveis e distintas maneiras de interpretar seus bytes: o modo texto ou o modo binário. É possível abrir o mesmo arquivo tanto de um modo como de outro, porém, a interpretação que se terá dos bytes lidos difere, e é preciso haver coerência entre o que foi gravado e o que será lido.

- **Arquivos texto**: ao usar a linguagem Python, ao abrir um arquivo no modo texto e efetuar sua leitura, seus bytes serão lidos, decodificados e interpretados segundo uma tabela de caracteres, e o conjunto resultante será retornado como um objeto string. Por outro

lado, ao realizar uma operação de gravação ocorrerá o processo inverso, no qual os caracteres de um string serão codificados, transformados em bytes e gravados no disco.

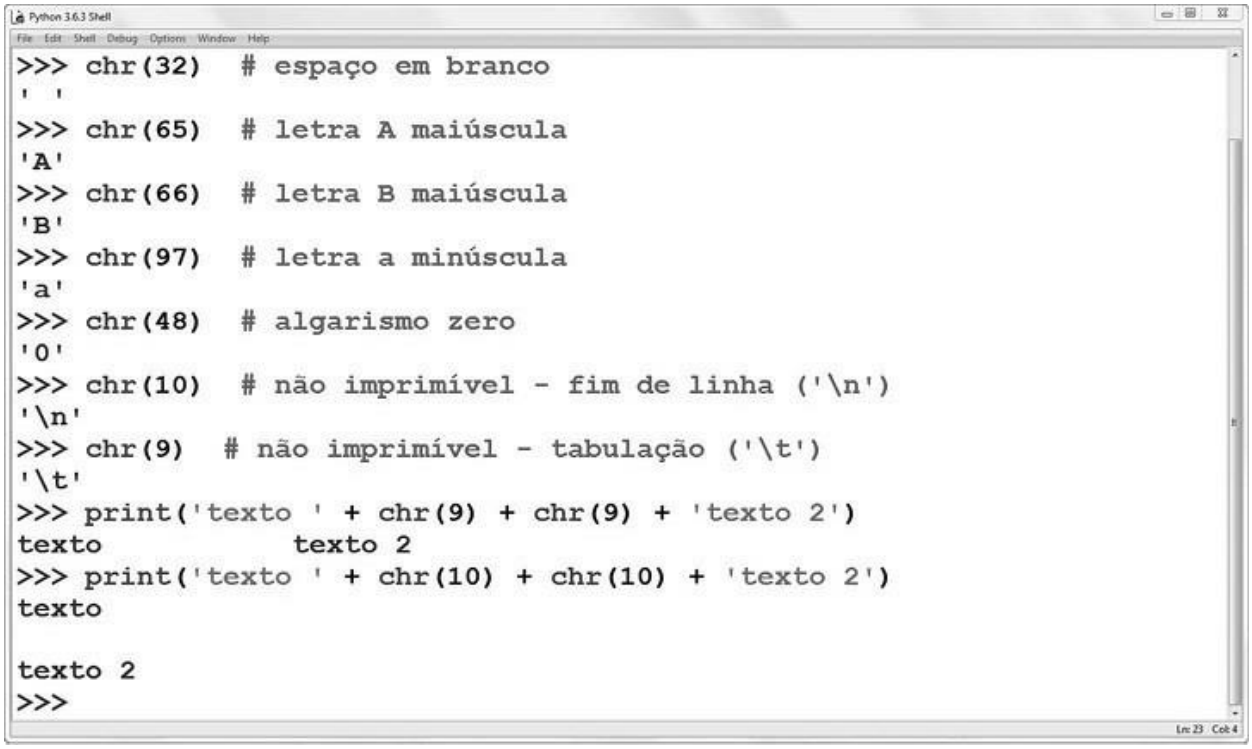
- Como exemplo de arquivos texto encontrados frequentemente tem-se: código-fonte de programas, arquivos HTML, CSS, JavaScript usados na web, arquivos XML e CSV (comma separated values) usados para intercâmbio de dados entre sistemas, arquivos TXT em geral.

- **Arquivos binários:** ao abrir um arquivo no modo binário, seus bytes são lidos e trazidos para a memória sem qualquer interpretação ou decodificação. São bytes em estado bruto, e caberá ao programador escrever o programa de maneira apropriada a fim de interpretar corretamente tais bytes.

- Como exemplo de arquivos binários comuns tem-se: arquivos de imagens, áudio e vídeo; arquivos de bancos de dados; arquivo executável de um programa compilado; arquivos compactados por meio de um algoritmo de compressão, entre outros.

O padrão ASCII (abreviação de *American Standard Code for Information Interchange*), criado na década de 1960, utiliza 7 bits para representar um caractere que é armazenado em 1 byte de 8 bits, sendo que oitavo bit não é efetivamente utilizado na codificação dos caracteres. Com esses 7 bits, é possível formar 128 números inteiros na faixa de valores de 0 a 127, sendo que cada um desses números equivale a um caractere diferente segundo a tabela padrão. Tais caracteres são letras, com diferenciação para maiúsculas e minúsculas, algarismos, pontuação e outros. São ao todo 95 sinais gráficos, conhecidos como *printables* (ou "*imprimíveis*"), e 33 sinais de controle que não têm uma aparência gráfica e, por isso, são conhecidos como *non-printables* ("*não imprimíveis*"), sendo usados em dispositivos de comunicação e transferência de arquivos, bem como elementos que afetam o processamento do texto, como caractere de fim de linha ("\n") ou tabulação ("\t").

Em Python pode-se usar a função chr para converter um número inteiro para o caractere corresponde e a função ord para fazer a operação inversa. A Figura 7.1 ilustra alguns casos, mostrando, por exemplo, que o número 32 equivale a um espaço em branco e 65 equivale à letra "A" maiúscula. Os caracteres 9 (tabulação) e 10 (fim de linha) também são exemplificados.



Exemplo de caracteres da tabela ASCII.

A tabela ASCII estruturada dessa maneira sempre foi muito apropriada para os textos no idioma inglês. No entanto, com o passar do tempo e o aumento da penetração dos computadores em todo o mundo, a tabela ASCII mostrou-se insuficiente para acomodar todos idiomas existentes e alternativas começaram a ser buscadas.

Visando solucionar as novas demandas, o primeiro e natural passo dado foi utilizar o oitavo bit para ampliar a faixa de possibilidades, incorporando-se, assim, os códigos de 128 até 255. Este oitavo bit passou a ser usado de diversas formas distintas, com certo prejuízo de padronização: em alguns casos, era empregado para informar a paridade em transmissão assíncrona de dados; a Microsoft utilizou-o para criar seu sistema de páginas de codificação (*Windows Code Page*) entre os anos de 1980 e 1990 etc.

Um dos usos dados a esse oitavo bit foi sua incorporação à codificação, passando-se a denominá-la tabela ASCII estendida e tornando possível acomodar os caracteres acentuados típicos dos idiomas da Europa Ocidental e Américas do Sul e Central. Como esse uso foi muito intenso e relevante, erroneamente se difundiu a ideia de que a tabela ASCII foi ampliada para

utilizar 8 bits. Fato este que, ao menos oficialmente, nunca ocorreu.

A busca de uma solução oficial para as limitações da tabela ASCII levou ao desenvolvimento do sistema de codificação Unicode, mantido pelo *Unicode Consortium* (ver referência UNICODE, 2017). Esse sistema permite a representação e a manipulação de texto de maneira consistente em qualquer sistema de escrita existente. Apenas 8 bits não eram suficientes para a representação de todos os caracteres de muitos idiomas, de modo que o Unicode trabalha com a opção de codificação usando 1 ou mais bytes por caractere. Em razão disso, cadeias de texto construídas utilizando-se a codificação Unicode são conhecidas no mundo da computação como *wide- character strings* ou *wide-strings*.

Para armazenar e manipular corretamente esse tipo de string são necessárias operações de codificação e decodificação que devem ser conhecidas pelos programadores. Assim, têm-se as definições:

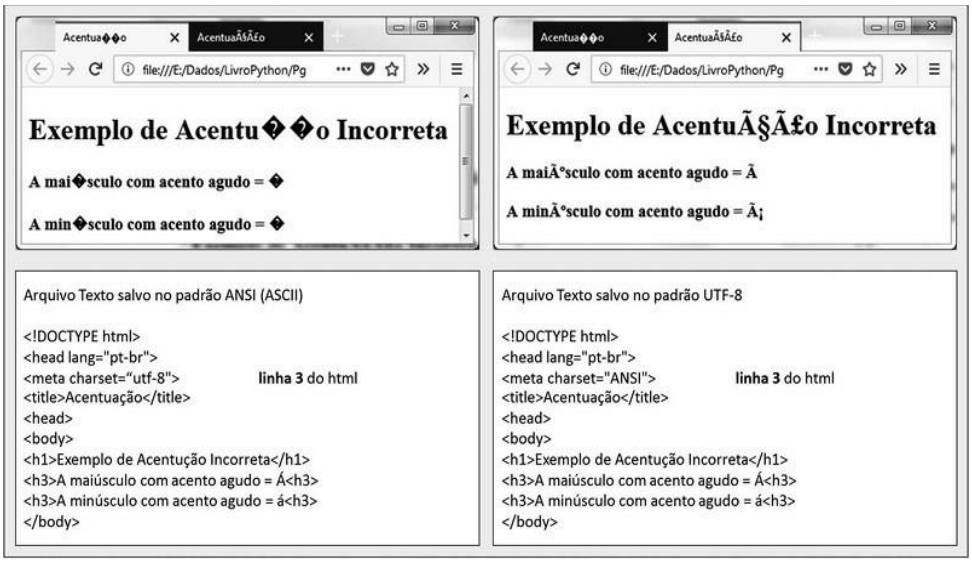
- **Codificação de um string**: é a conversão de cada caractere do string para os bytes (de 1 a 4 bytes por caractere) que o compõem, segundo o tipo de codificação desejada.
- **Decodificação de um string**: é a conversão dos bytes que representam o caractere (de 1 a 4 bytes por caractere), gerando o caractere em si segundo o tipo de codificação desejada.

A codificação supramencionada refere-se a qual subconjunto de caracteres Unicode se deseja utilizar. No mundo ocidental, as codificações ASCII (sim, ela continua a existir e é um subconjunto do Unicode), Latin-1 e UTF-8 são as mais amplamente utilizadas.

Conhecer esses conceitos é importante para o programador, uma vez que, com frequência, precisa-se desse conhecimento para não incorrer em erros comuns. A título de exemplo, considere-se a Figura 7.2. Ela mostra a exibição de uma página html desenvolvida em duas situações de incoerência, com diferentes resultados errôneos. Do lado esquerdo, o código html especifica que a codificação usada é UTF-8, ou seja, Unicode e o arquivo texto foi salvo com codificação ANSI, que usa a tabela ASCII. Do lado direito, foi criada a situação inversa: ou seja, o html especifica que a codificação usada é “ANSI”, mas o arquivo foi com codificação UTF-8.

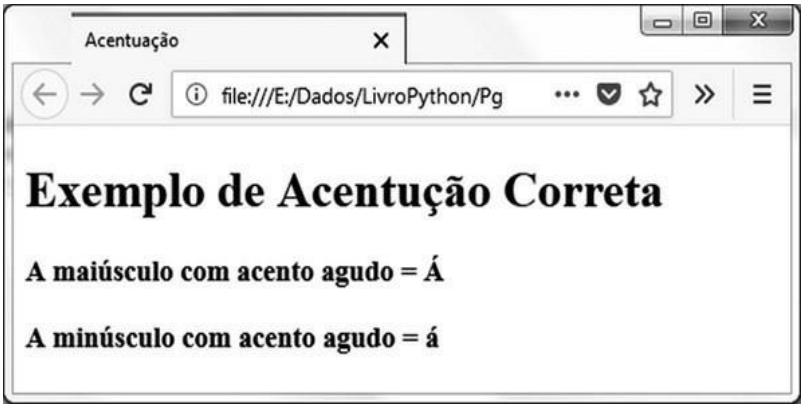
Ao utilizar os programas editores de texto, como *Notepad*++, no Windows, ou *Kwrite* e *Notepadqq*, no Linux, é possível especificar qual a codificação com que se quer gravar o arquivo. E, para evitar a ocorrência de erros assim, é necessário que o programador esteja atento e

sempre tome o cuidado de manter a coerência entre a codificação do arquivo salvo em disco com a maneira como são interpretados os bytes lidos pelo programa que se está escrevendo.



Erros frequentes relativos à codificação Unicode.

Se houver coerência entre o conteúdo do arquivo e a interpretação de seu conteúdo, então, tudo fica correto, como na Figura 7.3.



Coerência entre a codificação do arquivo e a interpretação de seu conteúdo.

A linguagem Python 3.0 oferece amplo suporte à codificação Unicode. Esse suporte está implementado na classe “str”, ou seja, nos strings já descritos no Capítulo 4. Naquele momento, nada foi dito sobre isso, pois faltavam as informações sobre codificação supra-apresentadas para tornar clara essa conceituação.

Arquivos em Python 3

Antes de começar, recomenda-se alguns cuidados. Tudo o que será tratado neste capítulo diz respeito à versão 3.x da linguagem Python. Nas versões anteriores (2.x), a forma de implementação é diferente em muitos casos. Na internet há muito material disponível sobre Python 2.x em fóruns, FAQs etc. Assim, se estiver fazendo buscas na internet com o propósito de complementar os conceitos e conteúdos aqui passados, verifique com atenção a versão de Python que está sendo discutida em cada material encontrado, pois há diferenças significativas entre Python 2.x e 3.x.

O Python conta com recursos voltados à gravação e à leitura de arquivos, sejam eles binários ou texto. Os arquivos binários não serão abordados neste livro, uma vez que demandam conhecimentos específicos de certos formatos de dados, tais como imagens ou áudio, que estão fora do escopo pretendido aqui. Todo este capítulo, daqui por diante, é dedicado aos arquivos texto.

Considere-se o Exemplo 7.1. Esse programa tem duas partes. Na primeira, grava-se o arquivo “Exemplo7_1.txt”, e na segunda parte o mesmo arquivo é lido.

A primeira providência é abrir o arquivo com o comando open. Esse comando recebe dois parâmetros: o nome do arquivo e o caractere indicador de modo de abertura. Para gravação, utiliza-se o caractere “w” *(write)*. Neste caso, se o arquivo não existir, será criado, e caso exista, terá seu conteúdo zerado (é preciso ter cuidado com isso, pois os dados de um eventual arquivo preexistente serão perdidos).

Como o nome do arquivo foi fornecido sem qualquer indicação de pasta, então, ele será gravado na mesma pasta onde está salvo o programa.

O comando open cria um objeto em memória e retorna seu id, que deve ser atribuído a um identificador. No caso do Exemplo 7.1, esse identificador foi denominado arq. A partir daí são empregados os métodos do objeto arquivo para executar as operações. O método write foi usado para executar a gravação do string G no arquivo e o método close foi uado para fechar o arquivo.

Na segunda parte do programa o mesmo arquivo foi aberto no modo de leitura, ou seja, passando-se “r” para o segundo parâmetro e utilizou-se o método readline para efetuar a leitura da primeira (e, neste caso, única) linha do arquivo.

Exemplo 7.1 Gravação e leitura de um arquivo texto print(“Inicio do Programa”)

Parte 1 - Gravação do arquivo

G = “Gravação e Leitura de Arquivo Texto” # carrega o string G arq = open(“Exemplo7_1.txt”, “w”) # abre o arquivo p/ gravar arq.write(G) # executa a gravação

arq.close() # fecha o arquivo

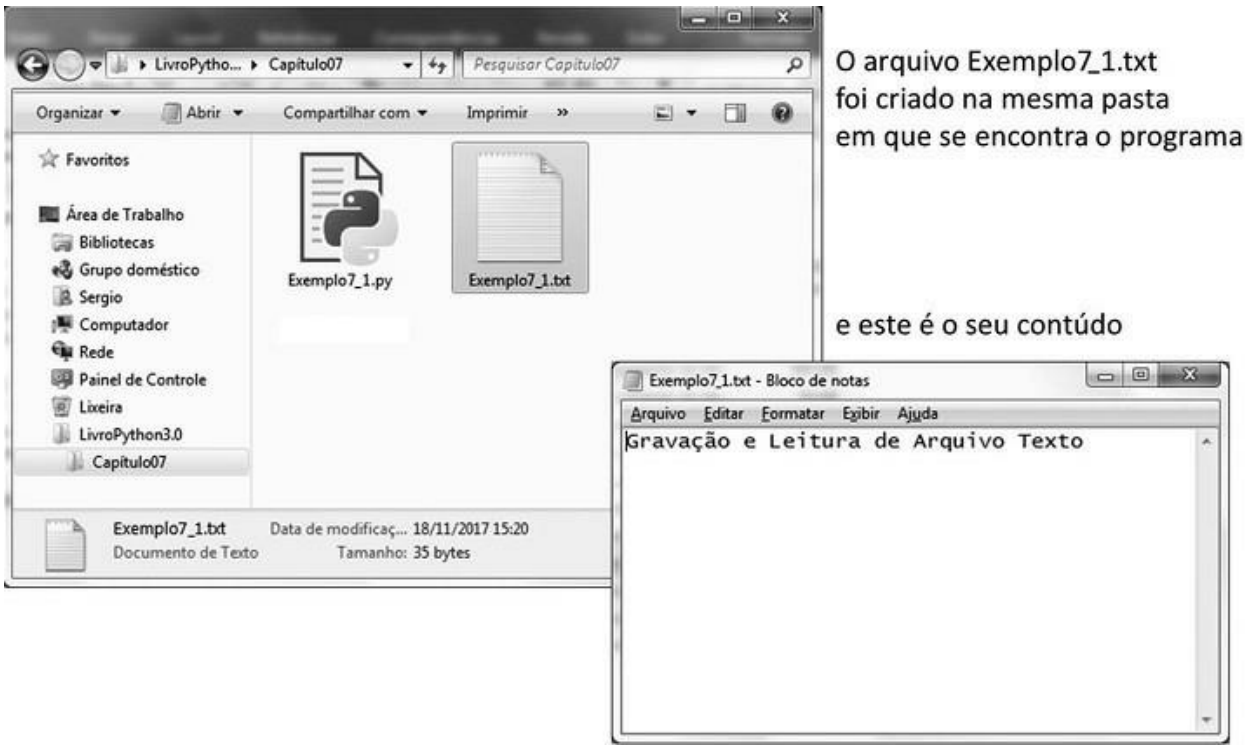
Parte 2 – Leitura do arquivo gravado na Parte 1

arq = open(“Exemplo7_1.txt”, “r”) # abre o arquivo p/ ler

L = arq.readline() # executa a leitura arq.close() # fecha o arquivo

print(“String lido = {}”.format(L)) # exibe o string lido

print(“Fim do Programa”)



Abertura dos arquivos no Python 3

A forma mais simples de uso do comando open envolve o fornecimento apenas do primeiro parâmetro, o nome do arquivo. Neste caso, o interpretador Python assume que o arquivo será aberto para leitura e todos os demais parâmetros assumirão valores padrão.

No entanto, a abertura de arquivos na linguagem Python apresenta alguns detalhes

relevantes que precisam ser conhecidos pelo programador. A forma completa do comando open é mostrada a seguir:

open(file, mode="r", buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

file

O primeiro parâmetro é o nome do arquivo que será lido e/ou gravado. Caso o programador queira criar o arquivo em outra pasta, então, deverá fornecer um nome qualificado, indicando a pasta de destino junto com o nome. O fornecimento do nome qualificado deve seguir as regras válidas para o sistema operacional em uso.

mode

Os modos de abertura de arquivos texto existentes em Python 3 não se limitam a "w" e "r", vistos há pouco. O Quadro 7.1 ilustra as possibilidades para arquivos texto. Note-se que existe também o modo "b", para arquivos binários, que não está presente neste quadro pois, como esclarecido anteriormente, arquivos binários não serão assunto deste livro.

Quadro 7.1 Modos de abertura de arquivos em Python 3.

Operações permitidas	r	r+	w	w+	a	a+	x	x+
Leitura de dados do arquivo	✓	✓		✓		✓		✓
Gravação de dados no arquivo		✓	✓	✓	✓	✓	✓	✓
Criação do arquivo			✓	✓	✓	✓	N_1	N_1
Zera o conteúdo de arquivo existente			✓	✓				
Posiciona cursor no início do arquivo (N_2)	✓	✓	✓	✓			✓	✓
Posiciona cursor no fim do arquivo ($N_{3\,e\,4}$)					✓	✓		

N_1 = Permite a criação do arquivo exclusivamente se este não existir. Caso exista, levanta a exceção "FileExistsError".

N$_2$ = O cursor de um arquivo é um controle posicional que indica (ou aponta) o próximo byte a ser lido. Quando um arquivo é aberto, normalmente o cursor é posicionado em seu primeiro byte.

N$_3$ = Os modos **a** e **a+** de Python posicionam esse cursor no final do arquivo, pois seu pressuposto é que o arquivo foi aberto para acréscimos no final (append).

N$_4$ = O modo **a+** permite a leitura, porém, na abertura do arquivo o cursor estará posicionado no final. Caso o programador efetue a leitura nessas condições, obterá um resultado vazio. Para conseguir efetuar a leitura, deve, antes, reposicionar o cursor do arquivo com o método seek.

buffering

Especifica as características de buferização do arquivo. As opções são: 0, então, não será usado buffer (permitido apenas para arquivos binários); 1, só se aplica a arquivos texto e o buffer conterá uma linha do arquivo; número inteiro maior que 1, indica um buffer de tamanho fixo com o valor indicado. Caso não seja especificado, o valor −1 é assumido e o interpretador adotará um esquema--padrão de buffer.

encoding

Este parâmetro só se aplica a arquivos texto e diz respeito à codificação descrita na primeira parte deste capítulo. Existem muitas opções para uso, mas as mais frequentes no mundo ocidental são “ansi” e “utf-8”.

Os demais parâmetros – erros, newline, closed, opener – fogem ao escopo deste livro, de modo que basta dizer que seus valores-padrão atendem às necessidades dos exercícios e projetos que serão aqui desenvolvidos.

Métodos relativos à manipulação de arquivos em Python 3

O Quadro 7.2 apresenta e explica os comandos e métodos relativos à manipulação de arquivos.

Método	Descrição
close()	Fecha o arquivo que foi aberto com open. Se o arquivo foi aberto para gravação, primeiro descarrega seu buffer.
flush()	Descarrega o buffer de arquivo aberto para gravação, sem fechá-lo.
s = read()	Lê o arquivo inteiro e retorna-o como um único string. Se o arquivo contiver várias linhas, insere um caractere "\n" para cada quebra de linha.
S = readline()	Lê uma linha do arquivo e avança o cursor para o início da próxima. Retorna um string com o conteúdo da linha, incluindo o caractere "\n" se este estiver presente.
L = readlines()	Lê todas as linhas do arquivo e retorna-as como uma lista de strings, incluindo o "\n" no final de cada uma.
write(S)	Grava no arquivo um string de caracteres. O objeto "S" deve ser do tipo string.
writelines(L)	Grava no arquivo todos os strings contidos na lista L.
seek(N)	Altera a posição do cursor do arquivo, posicionando-o no N-ésimo caractere, contado a partir do início do arquivo.

Quadro 7.2 Comandos e métodos relativos a arquivos em Python 3.

Sequência de números reais gravada em arquivo

Escreva um programa que permaneça em laço lendo números reais até que seja digitado 0. Todos os valores digitados, exceto o zero, devem ser gravados em um arquivo em disco, um por linha, com três casas decimais.

Serão criadas duas soluções para esse exercício. Na primeira será utilizado o método write, e na segunda será utilizado o método writelines.

Na linha 2 o arquivo é aberto. Na linha 5 é empregado o método write para efetuar a gravação. Como esse método exige um string, então, foi escrito o string "{0:.3f}\n", no qual 0:.3f garante que o valor x seja formatado com três casas decimais, e o caractere de final de linha, "\n" garante que cada valor esteja salvo em uma linha diferente. Para verificar o resultado, pode-se abrir o arquivo gravado usando o Bloco de Notas.

Ler um arquivo do tipo CSV

Escreva um programa que leia um arquivo texto que contém diversas linhas que representam uma lista de compras. Em cada linha há três informações: nome de um produto, quantidade e preço unitário, separados pelo caractere “;”. Pede-se que cada item da lista seja exibido na tela, incluindo o valor total do item. Ao final, exiba o total da compra.

Os valores devem ser exibidos com duas casas decimais. Um exemplo de arquivo de entrada é mostrado a seguir:

```
Leite,12,3.8 Maçã,100,4.4
Café,9,16.35
Pão de Forma,41,5.9
```

Antes de chegar à solução final para esse problema, é necessário apresentar alguns recursos que serão utilizados. Por questões didáticas, o processo será apresentado e utilizado passo a passo. Para tanto, considere-se o Exemplo 7.2.

Na primeira linha é atribuído um string típico desse problema ao objeto

S. Esse string contém um caractere “\n” no final, e a primeira tarefa é removê-lo. Isso poderia ser feito com o fatiamento S[:-1] (que significa todos os caracteres menos o último), porém, se o string não terminar em “\n”, haverá a remoção indevida do último caractere. Normalmente isso ocorre na última linha do arquivo, na qual pode não haver o caractere de fim de linha.

A solução, então, é usar o método rstrip do string, o qual remove caracteres à direita. Os caracteres a serem removidos devem ser especificados como parâmetro e, caso sejam omitidos, são removidos os espaços em branco e o “\n”.

Exemplo 7.2 Tratamento de um string CSV

```
>>> S = “prodA,12,3.8\n” # string S como virá do arquivo

>>> S[:-1] # remover o “\n” com fatiamento não é ‘prodA,12,3.8’ # a melhor opção
>>> S = “prodA,12,3.8” # se o “\n” não estiver presente, então
```

```
>>> S[:-1] # o último caractere será removido 'prodA,12,3.' # indevidamente
>>> S = "prodA,12,3.8\n"

>>> S.rstrip() # essa solução é melhor 'prodA,12,3.8' # o "\n" foi removido
>>> L = S.split(",") # separa S em uma lista de strings

>>> L # usando "," como delimitador ['prodA', '12', '3.8'] # lista produzida
>>> L[1], L[2] = int(L[1]), float(L[2]) # converte L[1] e L[2]
>>> L # para int e float,

['prodA', 12, 3.8] # respectivamente
```

O passo seguinte é usar o método split, que retorna uma lista de strings separados a partir de S. O parâmetro passado é um substring empregado como delimitador para a separação.

O último passo é converter o elemento L[1] para número inteiro e o elemento L[2] para número real.

A partir daí, os dados estão prontos para processamento.

Gerador de dados em arquivo do tipo CSV

Escreva um programa que leia um número inteiro N (10 < N < 10.000) e grave um arquivo com N linhas com os dados listados na tabela seguinte. O arquivo deve ter o nome "Estoque.csv" e deve usar o caractere ";" (ponto e vírgula) como delimitador. Não é necessário que o arquivo esteja ordenado.

Campo	Descrição
Código do produto	Número inteiro entre 10000 e 50000. Não pode haver repetição desse código, e pede-se que não sejam sequenciais (aleatórios).
Quantidade em estoque	Número inteiro entre 1 e 3800. Gerar aleatórios.

Preço unitário compra	Número real entre 1.80 e 435.90. Gerar aleatórios.
Alíquota do ICMS	Alíquota do imposto ICMS. Essa alíquota deve ser 7%, 12% ou 18%. (Não colocar o caractere “%” no arquivo).

RA.TXT	RASENHA.TXT
330019	330019;318A89P
414061	414061;E87H14M
109229	109229;019MKX9
827392	827392;313G093

etc...