Capítulo 14

Arquivos

Estamos muito acostumados a lidar com arquivos. Seja um documento de texto, uma planilha, uma figura ou um programa Python, tudo fica armazenado em um arquivo no nosso disco rígido ou outra mídia. Nossos programa também podem interagir com os arquivos, lendo dados a partir deles ou escrevendo dados neles.

Em particular, um tipo de arquivo que vamos utilizar é o "arquivo texto" ou "arquivo de texto". Como o nome sugere, nele armazenamos apenas texto, ou seja, uma sequência de caracteres. Podemos pensar no arquivo como um string, que pode ser muito grande, e que está armazenado no disco do nosso computador. Em geral, usamos para esse tipo de arquivo a extensão ".txt". Mas outras categorias de arquivos também são simplesmente texto como, por exemplo, os arquivos ".py" que temos usado.

Para criar ou editar um arquivo texto podemos usar vários programas como o Bloco de Notas do Windows ou mesmo o Geany ou outro ambiente de programação. Nas próximas seções vamos ver algumas das operações que podemos fazer sobre os arquivos texto. Daqui pra frente vamos nos referir a eles apenas como "arquivos".

14.1 Abrindo e fechando

Para identificar um arquivo armazenado no nosso computador precisamos saber seu nome e precisamos saber onde (qual pasta) ele está. Assim, todo arquivo tem um 'nome absoluto' que indica as esses duas coisas. Por exemplo, usando o nosso Bloco de Notas do Windows (Figura 14.1), vamos criar um arquivo "Poema.txt" e vamos armazená-lo na pasta "C:\Users\Eng\Documents". O arquivo contém parte do poema "O Engenheiro" de João Cabral de Melo Neto.

Para acessar esse arquivo dentro de nosso programa Python usamos a função `open` da biblioteca padrão. Para essa função devemos passar o nome do arquivo que queremos acessar e ela devolve um objeto que vai identificar, dentro do programa, esse arquivo. Por exemplo:

Programa 14.1 Abrindo um arquivo

```
1 f = open('C:\Users\Eng\Documents\Poema.txt')
```

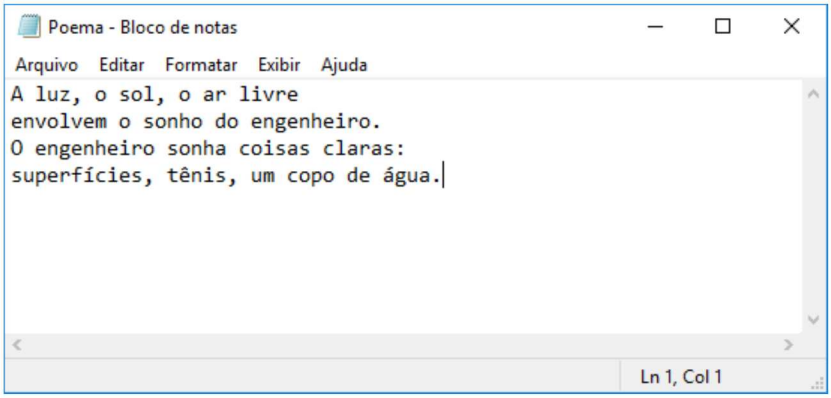

Figura 14.1: Arquivo texto editado no Bloco de Notas

A partir desse ponto, qualquer uso da variável `f` irá referenciar o arquivo *Poema.txt*. Costumamos dizer que o arquivo está "aberto" para podemos trabalhar com ele.

Podemos, também, identificar o nome do arquivo em relação à localização do nosso programa. Por exemplo, se o arquivo estiver na mesma pasta que o nosso programa, podemos simplificar e escrever:

Programa 14.2 Abrindo um arquivo, usando seu nome relativo ao programa

```
1 f = open('Poema.txt')
```

Quando terminarmos de usar o arquivo, precisamos fazer a operação contrária: devemos fechar o arquivo. Isso significa que deixamos de ter acesso àquele arquivo por meio da variável que usamos com a função `open`. Essa operação é importante pois ela avisa o sistema operacional que nosso programa deixará de acessar o arquivo. Ela evita, também, que nosso arquivo fique desatualizado ou seja corrompido. A seguir, o esquema geral de como utilizamos um arquivo.

Programa 14.3 Abrindo e fechando um arquivo

```
1 f = open('Poema.txt') #abre o arquivo
2
3 # aqui nosso programa acessa o arquivo
4
5 f.close() #fecha o arquivo
```

É importante notar que depois de fechado, não podemos mais usar a variável `f`. Se tentarmos fazer isso, o interpretador Python irá indicar um erro.

14.2 Lendo dados

Vamos admitir que o nosso arquivo *Poema.txt* foi criado e reside na mesma pasta do nosso arquivo. Agora, queremos ler o seu conteúdo, ou seja, usar aquilo que está no arquivo, dentro do nosso programa. Da mesma forma, existem funções específicas para se fazer isso.

A primeira forma de enxergarmos o arquivo é como um grande string. Assim, podemos simplesmente "ler" esse string do arquivo para dentro do nosso programa e, então, manipulá-lo como quisermos. Isso é feito pela função `read`.

Programa 14.4 Lendo dados de um arquivo

```
1 f = open('Poema.txt') #abre o arquivo
2 s = f.read()
3 print(s)
4 f.close() #fecha o arquivo
```

Ao ler o arquivo na linha 3, transferimos todo o seu conteúdo para a variável `s`. O comando `print` a seguir, vai apresentar esse conteúdo, como mostrado a seguir. Uma coisa a ser destacada é que nesse string existem alguns caracteres de final de linha, o tal de "\n" que vimos anteriormente. Por isso, a saída exibida aparece em quatro linhas.

```
> python poema.py
A luz, o sol, o ar livre
envolvem o sonho do engenheiro.
O engenheiro sonha coisas claras:
superfícies, tênis, um copo de água.
```

Uma outra forma de enxergar o arquivo texto é como uma sequência de linhas. Assim, podemos, também, ler o arquivo separando as suas linhas em uma lista usando a função `readlines`. Nesse caso, o retorno da chamada dessa função é uma lista na qual cada elemento é um string que corresponde a uma linha do arquivo.

Programa 14.5 Lendo linhas de um arquivo

```
1 f = open('Poema.txt') #abre o arquivo
2 linha = f.readlines()
3 print(linhas)
4 f.close() #fecha o arquivo
```

A saída desse programa é mostrada a seguir. A lista `linhas` tem 4 elementos, cada um é um string que contém uma linha do arquivo.

```
> python poema2.py
['A luz, o sol, o ar livre\n', 'envolvem o sonho do
engenheiro.\n', 'O engenheiro sonha coisas claras:\n',
'superfícies, tênis, um copo de água.\n']
```

Uma função, que não está diretamente ligada com arquivos, mas que é muito útil quando lemos dados deles é a `split`, que é aplicada um um objeto do tipo string. Ela quebra o string, colocando cada "palavra" em uma posição de uma lista. Por exemplo:

Programa 14.6 Lendo e quebrando as palavras de um arquivo

```
1 f = open('Poema.txt') #abre o arquivo
2 s = f.read()
3 palavras = s.split()
4 print(palavras)
5 f.close() #fecha o arquivo
```

Esse programa vai criar uma lista, que é mostrada na saída abaixo. Veja que cada espaço em branco do string é usado para separar as palavras. A saída é mostra a seguir.

```
> python poema3.py
['A', 'luz,', 'o', 'sol,', 'o', 'ar', 'livre', 'envolvem',
'o', 'sonho', 'do', 'engenheiro.', 'O', 'engenheiro',
'sonha', 'coisas', 'claras:', 'superfícies,', 'tênis,', 'um',
'copo', 'de', 'água.']
```

Essa função é útil, por exemplo, se tivermos um arquivo que contém os dados de consumo de cerveja de uma turma de estudantes de engenharia. O arquivo poderia ser representado da seguinte maneira:

```
 1.5  2  2  4
0  2.5  3.5
2  6  3.5
```

Então, para ler esses dados com o o intuito de calcular, por exemplo, as estatísticas que vimos anteriormente, bastaria implementar a seguinte função. Na linha 3 o arquivo é lido e o string retornado é quebrado e colocado em uma lista. Em seguida, cria-se uma nova lista, na qual os elementos, transformados para

`float`, são inseridos. Isso é necessários pois queremos uma lista com números e não com strings.

Programa 14.7 Lendo números em uma lista

```
1 def le_dados(arquivo):
2     f = open(arquivo)
3     s = f.read().split()
4     f.close()
5     r = []
6     for c in s:
7         r.append(float(c))
8     return r
```

14.3 Lendo com o comando `for`

Podemos ter arquivos muito grandes, com milhares de linhas. Ao usar a função `read` todo o conteúdo do arquivo é transferido para o nosso programa, ou seja, para a memória do computador. Isso pode ser um problema pois podemos não ter tanta memória disponível. Por isso, podemos adotar a estratégia de ler e processar apenas uma linha por vez.

No programa abaixo, a função `readline` é usada para ler uma linha do arquivo. Para saber até quando devemos ler, ou seja, para descobrir quando o arquivo lido terminou, usamos o tamanho do string devolvido pela função `readline`. Quando esse string for vazio, ou seja, tiver tamanho zero, então, terminamos de ler o arquivo.

Programa 14.8 Lendo linhas com `readline`

```
1 f = open('numeros.txt')
2 s = f.readline()
3 while len(s) > 0:
4     # usa string em s
5     s = f.readline()
6 f.close()
```

Uma forma mais simples de fazer isso é utilizando o comando `for`. Assim como fazemos com listas, podemos usar esse comando para percorrer cada elemento – no caso, cada linha – do arquivo. No programa a seguir fazemos isso.

Programa 14.9 Lendo linhas com o comando `for`

```
1 f = open('numeros.txt')
2 for s in f:
3     # usa string em s
4 f.close()
```

14.4 Escrevendo dados em um arquivo

Nosso programa pode, também, usar arquivos para armazenar dados que foram gerados. Para isso, algumas diferenças existem em relação à leitura de arquivos. A primeira, é que precisamos, no nosso programa, avisar que não vamos ler dados do arquivo, mas sim, vamos colocar dados nele. Isso é feito ao abrirmos o arquivo.

Programa 14.10 Abrindo o arquivo para escrever

```
1 f = open('Poema2.txt', 'w')
2 f.write('O lápis, o esquadro, o papel;\n')
3 f.write('o desenho, o projeto, o número:\n')
4 f.write('o engenheiro pensa o mundo justo,\n')
5 f.write('mundo que nenhum véu encobre.\n')
6 f.close()
```

No programa acima, a variável `f` refere-se a um arquivo no qual podemos gravar dados. O segundo parâmetro do comando `open`, ou seja, o string "w" indica justamente que o arquivo vai ser usado para escrever dados, não para ler. Isso permite que, nas linhas seguintes possamos usar o comando `write` para escrever as quatros linhas seguintes do poema de João Cabral de Melo Neto.

De certa forma, o comando `write` se assemelha ao comando `print`, que já conhecemos muito bem. Algumas diferenças, porém, precisam ser apontadas:

- o comando `a`ceita um único parâmetro;
- esse parâmetro tem que ser, necessariamente, um string;
- o comando `write` não insere automaticamente um caractere de final de linha como é o comportamento padrão do `print`. Por isso, nos comandos do nosso programa temos um "\n" no final de cada string que escrevemos no arquivo;
- o string, no caso do `write` vai aparecer no arquivo correspondente e não na saída do nosso programa, como no caso do `print`.

O Programa 14.10 poderia utilizar apenas um comando `write` em vez de quatro. Bastaria termos um único string com as quebras de linha posicionadas corretamente e o resultado seria o mesmo. Mostramos esse programa a seguir.

Programa 14.11 Usando um único `write`

```
1 f = open('Poema2.txt', 'w')
2 f.write('O lápis, o esquadro, o papel;\no desenho, \
3 o projeto, o número:\no engenheiro pensa o mundo \
4 justo,\nmundo que nenhum véu encobre.\n')
5 f.close()
```

No caso de querermos escrever outros tipos de dados que não sejam strings precisamos transforma-los em strings antes. No exemplo a seguir, vamos tomar uma lista de números do tipo `float` que representam, por exemplo, o consumo de cerveja dos alunos de uma turma de engenharia e vamos escrevê-los em uma arquivo. Para complicar um pouco, vamos sempre escrever 3 valores em cada linha do arquivo e cada valor formatado com duas casas decimais.

Programa 14.12 Escrevendo números de uma lista

```
1 def grava_dados(arquivo,lista):
2     f = open(arquivo, 'w')
3     i = 0
4     for c in lista:
5         f.write('{:7.2f}'.format(c))
6         i += 1
7         if i % 3 == 0:
8             f.write('\n')
9     f.close()
```

Para uma lista como $[1.5, 2, 2, 4, 0, 2.5, 3.5, 2, 6, 3.5]$ teríamos a saída, gravada no arquivo cujo nome é passado por parâmetro, mostrada a seguir.

```
 1.50   2.00   2.00
 4.00   0.00   2.50
 3.50   2.00   6.00
 3.50
```

14.5 Modos de abertura

O segundo parâmetro usado na função `open` indica como queremos ou para que queremos abrir o arqui. Por isso é chamado de "mode de abertura".

Existem várias formas de abrir um arquivo. A seguir descrevemos aquelas mais usuais e que mais nos interessam.

Modo "r": indica que o arquivo vai ser usado para ler dados. Nesse caso, o arquvo precisa existir no sistema de arquivos. Caso ele não exista, o interpretador vai abortar a execução do programa indicando um erro de execução, como vemos a seguir.

```
>>>f = open('arquivo_nao_existe.txt', 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory:
'arquivo_nao_existe.txt'
```

Esse modo de abertura é o modo padrão. Ou seja, se passarmos apenas um parâmetro para a função `open` assume-se que esse será o modo de abertura do arquivo.

Modo "w": indica que queremos ler dados do arquivo. Nesse caso, se arquivo não existe não é um problema pois o arquivo é criado, sem nenhum dado nele. Se o arquivo existe, ele é descartado e um novo arquivo, com o mesmo nome é criado, sem nenhum dado nele. Isso quer dizer que se executarmos duas vezes o Programa 14.10 vamos, na primeira vez, criar o arquivo e nele escrever o poema. Na segunda vez, o arquivo é deletado, em seguida criado e por fim escrevemos o poema nele. Ou seja, no final, teremos o arquivo contendo as quatro linhas do poema, exatamente como se tivessemos executado o programa uma única vez.

Ainda assim, podemos ter um erro ao executarmos a função `open`, se tentarmos criar o arquivo em uma pasta que não existe ou em uma pasta que não temos autorização para modificar. No exemplo abaixo, tentamos criar o arquivo na pasta "inexiste", que não existe.

```
>>>f = open('inexiste/Poema.txt', 'w')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory:
'inexiste/Poema.txt'
```

Modo "a": é o modo para adicionar coisas no final do arquivo. Se o arquivo não existe, ele é criado, desde que seja em uma pasta válida. Se o arquivo já existe, o seu conteúdo não é descartado e tudo que nosso programa escreve no arquivo vai para o final do conteúdo que já existe. Então, se executarmos duas vezes o Programa 14.10 mas com o modo de abertura "a" em vez do "w", teríamos o seguinte conteúdo no arquivo *Poema2.txt*:

```
O lápis, o esquadro, o papel;
o desenho, o projeto, o número:
o engenheiro pensa o mundo justo,
mundo que nenhum véu encobre.
O lápis, o esquadro, o papel;
o desenho, o projeto, o número:
o engenheiro pensa o mundo justo,
mundo que nenhum véu encobre.
```

Na primeira execução, são gravadas as quatro primeiras linhas do arquivo. Na segunda execução, nosso programa iria abrir o arquivo e começar a escrever no seu final, ou seja, a segunda cópia do poema, que vemos acima.

14.5.1 Exercícios

1. Faça um programa que crie um arquivo texto, com o nome "dados.txt", e escreva neste arquivo em disco uma contagem que vá de 1 até 100, com um número em cada linha. Abra este arquivo em um editor de textos, como por exemplo o Notepad ou o Geany do Windows.

2. Faça um programa que leia um arquivo texto do disco, lendo linha a linha, e exibindo cada uma das linhas numeradas na tela. A ideia é podermos pegar um arquivo texto qualquer (pode ser inclusive o arquivo do programa fonte – arquivo ".py") e mostrar na tela com as linhas numeradas. As primeiras linhas do arquivo iriam ser exibidas na tela da seguinte forma:

```
001:  def h_peso_ideal(altura):
002:      peso = altura * 72.7 - 58.0
003:   return peso
004:
005:  def m_peso_ideal(altura):
006:      peso = altura * 62.1 - 44.7
007:      return peso
```

3. Complemente o programa anterior para que ele grave o resultado em um outro arquivo.

4. Suponha arquivo "mega-sena.txt" tem os números sorteados em todos os concursos da mega sena, um concurso em cada linha do arquivo. Escreva um programa que leia esse arquivo e descubra qual a dezena que mais apareceu e qual a que menos apareceu.

5. Escreva um programa que lê um arquivo texto e exibe: o número de linhas, o número total de caracteres, o número de espaços em branco e o número de caracteres não brancos encontrados. O nome do arquivo de entrada deve ser digitado pelo pelo usuário. Sugestão: use a função isspace para verificar se um caractere é um espaço.

6. O seu professor gostaria de avaliar o desempenho de uma turma, em relação às notas de uma avaliação. Para isso, criou o arquivo "notas.txt", com o seguinte formato, representando o número de matrícula dos alunos e as notas obtidas:

```
9654819 5.2
7788950 5.2
9560923 9.7
6095479 0.6
8006694 5.5
9569242 8.7
9889433 9.9
```

Escreva um programa que mostre o seguinte relatório:

```
------------------------------
Matrícula      Nota   Difer.
9654819        5.20    -1.20
7788950        5.20    -1.20
9560923        9.70     3.30
6095479        0.60    -5.80
8006694        5.50    -0.90
9569242        8.70     2.30
9889433        9.90     3.50
------------------------------
Média: 6.40
Acima da média: 3
Abaixo da média: 4
```

A terceira coluna representa a diferença entre a nota do aluno e a média. Notas maiores ou iguais à médias contam como"Acima da média". O relatório deve ser exibido na tela do computador e também gravado em um arquivo "relatório.txt" Use funções para: 1) calcular a média; 2) calcular a terceira coluna do relatório; 3) mostrar os relatórios.

7. Um linguista muito famoso está fazendo a análise dos nomes dos alunos da sua universidade e para isso, criou um arquivo "nomes.txt" com o seguite formato:

```
Alecio Navarro
Ályson Buarque
Amanda de Mello
Ana Flávia Correia
Ana Julia Picard
Ana Luiza Severiano
Bruna Nordsveri
```

ou seja, um nome por linha. Agora ele quer saber a frequência com que cada nome aparece. Para isso, você deve criar um programa que gera o seguinte relatório:

```
---------------------------------
Nome              Ocorr.  % total
Alecio                 1    14.29%
Ályson                 1    14.29%
Amanda                 1    14.29%
Ana                    3    42.86%
Bruna                  1    14.29%
---------------------------------
Média ocorrencias: 1.40
Média percentagens: 20.00
```

A terceira coluna representa a porcentagem de vezes que o nome apareceu, no total de alunos analisados. O relatório deve ser exibido na tela do computador e também gravado em um arquivo "relatório.txt". Use funções para: 1) contar quantas vezes cada nome aparece; 2) mostrar os relatórios; 3) outras que você achar necessário.

14.6 Manipulação externa de arquivos

A biblioteca Python possui um módulo chamado `os` que permite que comandos do sistema operacional sejam executados de dentro dos nossos programas, utilizando-se funções pré-definidas. A descrição desse módulo pode ser encontrada em `https://docs.python.org/3/library/os.html`.

Um conjunto particularmente interessante é das funções que manipulam pastas e arquivos. A sua descrição pode ser encontrada em `https://docs.python.org/3/library/os.html#os-file-dir`. Vamos a seguir descrever algumas dessas funções.

Quando executamos um programa Python usando o interpretador, essa execução possui uma pasta corrente. Quando executamos um programa de dentro de um ambiente de programação, esse ambiente acerta as coisas de modo que a pasta corrente da execução seja a pasta onde nosso programa está. Então, executando o seguinte programa

Programa 14.13 Exibindo a pasta corrente

```
1 import os
2
3 print(os.getcwd())
```

obtemos a saída mostrada a seguir, supondo que o arquivo do nosso programa está na pasta `/home/user/PythoParaEng/`. Note que a responsável por retornar um string que contém a pasta corrente é a função `os.getcwd`.

```
/home/user/PythonParaEng
```

De modo diverso, se fizermos a chamada do programa a partir do console do sistema operacional, o valor retornado pela função `os.getcwd`, ou seja, a pasta corrente do programa é a pasta corrente do console que fez a chamada. Por exemplo:

```
> pwd
/home/user
> python PythonParaEng/testa_getcwd.py
/home/user
```

Note que nesse exemplo, a pasta corrente do console é `/home/user` e executando-se o programa *testa_ getcwd.py* que está dentro da pasta *PythonParaEng*, obtemos como resposta não a pasta onde está o arquivo mas sim a pasta corrente.

A função `os.chdir` serve para mudar o diretório corrente. Ela recebe como parâmetro um string que representa a pasta que deve passar a ser a pasta corrente. Esse string pode ser especificado em relação à pasta corrente do programa – se não inicia com uma "/" – ou como um caminho absoluto, caso contrário. No exemplo a seguir, vemos os dois caso. Na linha 3, altera-se a pasta corrente para *PythonParaEng/dados*, que está dentro da pasta corrente. No segundo caso, linha 4, a pasta corrente passa a ser */home/user/PythonParaEng*, qualquer que seja a pasta corrente. Nas linhas 5 e 6, mostramos que os os nomes especiais de pasta ".." e "." também podem ser usados com essa função. O primeiro faz com que a pasta corrente seja aquela "acima" da pasta corrente o segundo simplesmente não altera a pasta corrente.

Programa 14.14 Alterando a pasta corrente

```
1 import os
2
3 os.chdir('PythonParaEng/dados')
4 os.chdir('/home/delamaro')
5 os.chdir('..')
6 os.chdir('.')
```

No caso em que a mudança de pasta não possa ser feita – por exemplo por não existir nova pasta ou por não ter o programa acesso a ela – um erro será apontado pelo interpretador Python.

Uma função muito útil é `os.listdir`. Ela retorna uma lista que contém todas as entradas – arquivos e subpastas – presentes em uma determinada pasta. Essa pasta deve ser passada como parâmetro ou, caso não seja, a função devolve o resultado relativo à pasta corrente do programa. No exemplo a seguir, nosso programa obtém a lista de entradas da pasta corrente e, em seguida, mostra o nome de cada uma delas, indicando se é um arquivo ou uma subpasta. Para isso, usamos as funções `os.path.isdir` e `os.path.isfile`.

Programa 14.15 Identificando as entradas de uma pasta

```
1 import os
2 import os.path
3
4 entradas = os.listdir()
5 for nome in entradas:
6     if os.path.isdir(nome):
7         print('{} é uma pasta'.format(nome))
8     elif os.path.isfile(nome):
9         print('{} é um arquivo'.format(nome))
```

Outras funções para manipulação de arquivos e pastas estão sumarizadas na Tabela 14.1.

Nome da função	**Significado**
`os.mkdir(pasta)`	Cria uma nova pasta.
`os.remove(arquivo`	Deleta um arquivo
`os.rename(arq1,arq2)`	Traca o nome do arquivo `arq1` para `arq2`
`os.rmdir(pasta)`	Deleta uma pasta, que tem que estar vazia
`os.unlink(arquivo)`	O mesmo que `os.remove(arquivo)`

Tabela 14.1: Algumas funções para manipulação externa de arquivos e pastas

14.6.1 Exercícios