Introdução à leitura/gravação de arquivos em *Python*

Em *Computação*, todo programa e todo conteúdo de programas, devem ser arquivados, o que é feito no denominado **sistema de arquivos**. Do ponto de vista do **sistema operacional**, cada **arquivo** pode na verdade estar registrado em vários espaços não contíguos do **disco**, como ilustrado na figura 1.

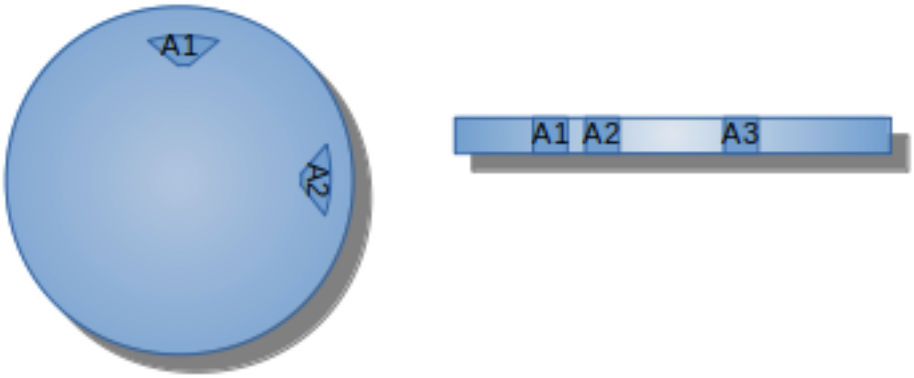

Fig. 1. Ilustração de um programa arquivado em disco rígido ou sólido (SSD).

1. Comandos para leitura de arquivo.

Apesar do arquivo poder estar espalhado pelo *disco rígido*, ele é tratado pelo *sistema operacional* como uma unidade. Assim, cada *linguagem de programação* dispõe de comandos específicos para leitura e gravação de arquivos.

Em **Python** é necessário declarar um *apontador para arquivo* (digamos `apont_arq`), então associá-lo com o arquivo propriamente dito a partir da função `open` (*open file*). Por exemplo, se o arquivo for preparado apenas para leitura, deve-se usar o parâmetro "`r`" (*read only*): `apont_arq = open(arquivo1, "r")`. Ao final, para evitar danificar o arquivo, usar sempre o comando para "fechar" o arquivo: `apont_arq.close()`.

Quanto á leitura dos dados (para variáveis internas) ou para impressão, a linguagem **Python** dispõe de alguns métodos especiais, como `readline` para leitura e `write` para gravação, por exemplo: `apont_arq.readline()` e `apont_arq.write("%d : " % i)`

1.1 Leitura de arquivo com texto em *Python*.

O código 1 ilustra a leitura de arquivo com conteúdo na forma de texto.

```
# Leitura do arquivo linha por linha (como "strings"): usando 'for'
def leitura_linha_for (nome_arq) :
  print("Imprime linha por linha (usando 'for'): ");
  conta = 0;
  try :
    apont_arq = open(nome_arq, 'r'); # aponta para arquivo – o parametro 'r'
=> "read", ler
```

```
    linha = apont_arq.readline();
    print(" %2d: (%3d) %s" % (conta, len(linha), linha), end=""); # a linha
finaliza com um \n => nao deixar quebra extra!
    # Py3 while ( linha is not "" and conta<10) # usar "" funciona como
finalizador no Python 3, mas nao no 2

    for linha in apont_arq : # funciona no Python 2 e  Python 3
      conta += 1;
      print(" %2d: (%3d) %s" % (conta, len(linha), linha), end=""); # a linha
finaliza com um \n => nao deixar quebra extra!
    apont_arq.close();
  except ValueError:
    print("Erro ao tentar ler arquivo %s: tipo do erro %s" % (argv[1],
ValueError));
  print();
```

Cód. 1. Código ilustrando a leitura de um arquivo com texto em Python.

1.2 Leitura de arquivo com dados numéricos em *Python*.

É possível ler conteúdo numérico, por exemplo, o código 2 ilustra a leitura de um arquivo com dados para uma matriz, sendo que em sua primeira linha estão os valores inteiros *m* e *n*, respectivamente, o número de linhas e de colunas da matriz, em seguida *m* linhas, cada uma com *n* valores "flutuantes".

```
def imprime_listas_vetores (mat, m, n) :
  for i in range(m) :
    for j in range(n) :
      print("%5.2f " % mat[i][j], end=""); # Imprime com 2 casas decimais,
ajustando 5 casas `a direita
    print();

# Leitura do arquivo linha por linha (como "strings"): usando 'for'
# Tentando interpretar todos como linhas de inteiros
# Gera uma matriz com linhas de diferentes tamanhos.
# Quem invoca precisa declarar o parametro "mat" (para passa-lo via
referencia):  mat = []
# Devolver a matriz lida
def leitura_linha_for_numeros (nome_arq, mat) :
  print("Imprime linha por linha (usando 'for'): ");
  conta = 0;
  apont_arq = open(nome_arq, 'r'); # aponta para arquivo - o 'r' => "read",
ler

  try :
    linha_str = apont_arq.readline();
    m, n = map(int, linha_str.split()); # transforma "string" em 2 inteiros
    print("m=%d, n=%d" % (m,n);

    for linha_str in apont_arq :
      #D print(" %2d: (%3d) %s" % (conta,len(linha),linha), end=""); # a
linha finaliza com um \n => nao deixar quebra extra!
      conta += 1;
```

```
      linha = map(float, linha_str.split()); # transforma "string" em linha
de inteiros

      # (B) Python 3 precisa explicitar lista
      lista = list(linha); # no Python 2 pode deixar apenas "lista = linha" -
vide (B)

      mat.append(lista); # nova linha de dados
    print("Foram lidas %d linhas do arquivo %s" % (conta, nome_arq));
    apont_arq.close(); # fechar arquivo para evitar erros
    imprime_listas_vetores(mat,m,n);
  except ValueError:
    print("Erro ao tentar ler arquivo %s: tipo do erro %s" % (nome_arq,
ValueError));
    print("Era esperado um arquivo com apenas numeros!\nLinha: %d" % conta);
    print("%s" % linha);
  print();
  # Poderia devolver a matriz: return mat;
```

Cód. 2. Código ilustrando a leitura de um arquivo texto com uma matriz de "flutuantes" em Python.

Assim, o programa que precisar fazer a leitura da matriz em um arquivo, por exemplo, de nome "*dados1.txt*", bastaria invocar a função com:

```
matriz = []; resp = leitura_linha_for_numeros("dados1.txt", matriz);
```

ou seja, precisa dispor da uma matriz de "flutuantes" (como `matriz[][]`, inicialmente vazia). O número de linhas e colunas podem, respectivamente, ser conseguidos após "voltar" da função `leitura_linha_for_numeros(.)`, com os comandos:

```
m = len(matriz); n = len(matriz[0]);.
```

Note que, ao passar a matriz de nome `matriz`, por essa ser um *vetor/lista*, automaticamente será uma passagem por **referência**, assim, qualquer alteração no *parâmetro formal* `mat` dentro da função, implicará em alteração imediata dos correspondentes *parâmtros efetivos*, no caso `matriz`.

[Leônidas de Oliveira Brandão](#)
http://line.ime.usp.br

Alterações