

Projeto 1: Demanda de Mercadorias e Rentabilidade de Vendas

O problema

Uma distribuidora de equipamentos e peças industriais trabalha com um esquema de carteira de pedidos. Isso significa que seus clientes enviam uma programação de pedidos a serem entregues no futuro. O conjunto de itens que constituem esses pedidos compõe a chamada carteira de pedidos.

Com base na programação de entrega, a empresa precisa manter um controle de estoque que permita avaliar as necessidades de compras futuras, de modo a garantir que não falem mercadorias e os pedidos programados possam ser entregues. Assim, rotineiramente, o controlador desse estoque emite um relatório de controle que, com base no cadastro de produtos e nos pedidos programados, mostra dois conjuntos de informações:

1. Posição do estoque e demanda de mercadorias no período.
2. Totais da carteira, incluindo preço médio de venda e margem média de cada produto.

Este projeto consiste em elaborar um programa que gere tais informações a partir dos dados disponíveis na empresa, que são:

3. Cadastro com dados dos produtos que são necessários a esse programa (este será o arquivo PRODUTOS.TXT).

Registro da programação de entregas contendo as quantidades e preços unitários de venda de cada pedido programado (este será o arquivo VENDAS.TXT).

Uma questão adicional que se apresenta é o fato de que os dados reais da empresa não estão disponíveis para o programador. Como este necessita testar o programa que escreverá, será necessário gerar arquivos de testes.

O primeiro é algo simples, não é necessário ter muitos produtos cadastrados para realização dos testes, sendo que algo em torno de 10 a 20 produtos devem bastar. Um arquivo assim pode ser digitado em um editor de textos, como o programa Bloco de Notas.

Já o segundo arquivo não é tão simples. É necessário ter um volume maior de informações. Supondo que o período de avaliação seja de um mês (22 dias úteis) e que em cada



dia haja em torno de 25 entregas, serão 550 registros a serem digitados. Além do grande volume, é necessário que haja coerência e consistência nesses dados, de modo que sua produção manual é inviável.

Portanto, neste projeto serão desenvolvidos dois programas:

1. **gerador.py:** servirá para gerar o arquivo VENDAS.TXT.
2. **apurador.py:** gerará os resultados de estoque, demanda de compras e margem média dos produtos. Esses resultados serão gravados em um arquivo de saída chamado APURA.TXT.

Dados de entrada

Nesta etapa são descritos os dados de entrada que devem estar gravados nos dois arquivos texto: PRODUTOS.TXT e VENDAS.TXT.

PRODUTOS.TXT

Este arquivo está organizado de modo que cada linha contém os dados de um produto, delimitados pelo caractere ';', e tem suas linhas organizadas em ordem crescente de código do produto.

O primeiro campo é o código do produto, e não pode haver repetição deste. O segundo campo é a quantidade em estoque no início do período de apuração. O terceiro campo é a quantidade mínima do produto que deve haver em estoque. Essa quantidade mínima é mantida para que a empresa possa atender a demandas urgentes e não programadas que às vezes ocorrem (embora tais demandas não programadas não sejam consideradas neste projeto).

O quarto campo contém o preço unitário de compra do produto. O último campo é a margem mínima de venda, porcentagem que é aplicada ao preço de custo para se obter o preço de venda. A margem real aplicada a uma venda específica pode ser um pouco maior que essa margem mínima. Isso será explicado quando for definida a geração do arquivo de vendas.

Este arquivo tem os seguintes campos:



Posição	Informação	Formato	Observações
1	Código do produto	5 dígitos numéricos	
2	Quantidade em estoque	Nº inteiro	Quantidade de produtos em estoque no início do período.
3	Quantidade mínima	Nº inteiro	Quantidade mínima que a empresa deve ter em estoque
4	Custo unitário	Nº real	Preço unitário que a empresa paga quando adquire o produto.
5	Margem de venda	Nº real	Margem mínima que é aplicada ao preço de custo para gerar o preço de venda.

O Quadro 9.1 mostra um exemplo desse arquivo com quatro produtos.

```
12100;1417;500;2.30;38.80
12200;725;100;23.70;13.58
15100;618;500;5.60;34.90

15200;223;50;61.90;8.15
```

Quadro 9.1 Exemplo de arquivo PRODUTOS.TXT.

VENDAS.TXT

Este arquivo está organizado de modo que cada linha contém os dados de uma venda, delimitados pelo caractere ‘;’, sendo organizado por data (Ano/Mês/Dia), e as vendas da mesma data não seguem uma ordem específica.

Os três primeiros campos são Ano (com 4 dígitos), Mês e Dia (com DOIS dígitos cada). Em seguida, vem o Código do Produto. Todos esses dados são números inteiros.

O quinto campo é a quantidade vendida. O histórico de vendas da empresa mostra que em 60% das vendas essa quantidade fica entre 1 e 10 peças, em 25% fica entre 11 e 25 peças e em 15% fica entre 26 e 400 peças.

O preço unitário de venda é o último campo da linha.

Cada linha contém o registro de venda de um produto com o layout:



Posição	Informação	Formato	Observações
1	Ano	Nº inteiro	Os campos Ano, Mês e Dia em conjunto definem a data da venda.
2	Mês	Nº inteiro	
3	Dia	Nº inteiro	
4	Código do Produto	Nº inteiro	
5	Quantidade Vendida	Nº inteiro	
6	Preço unitário	Nº real	Preço unitário de venda.

2017;5;1;12100;475;3.42

2017;5;1;15200;87;68.18

2017;5;2;12100;254;3.19

2017;5;2;12200;37;27.39

2017;5;2;12100;251;3.26

2017;5;3;15200;47;71.90

2017;5;3;15200;59;68.18

2017;5;3;12100;364;3.19

2017;5;5;15200;48;69.42

Quadro 9.2 Exemplo de arquivo VENDAS.TXT.

Saída – primeira parte: demanda e compras

A saída a ser produzida neste projeto contém duas partes, que serão descritas nesta e na próxima etapa. Conforme mencionado anteriormente, essa saída será gravada em um arquivo texto chamado APURA.TXT.

A primeira parte da saída serão a demanda por produtos e a necessidade de compras para atender a essa demanda. O Quadro 9.3 exibe a saída que será gerada a partir dos dados contidos nos Quadros 9.1 e 9.2.



NECESSIDADE DE ESTOQUE NO PERÍODO

Prod.	Estoque Inicial	Demanda	Estoque Final	Estoque Mínimo	Neces. Compra
12100	1417	1344	73	500	427
12200	725	37	688	100	0
15100	618	0	618	500	0
15200	223	241	0	50	68

Quadro 9.3 Saída – primeira parte: demanda e compras.

O Estoque Inicial e o Estoque Mínimo vêm do cadastro de produtos (veja o Quadro 9.1 e compare com o Quadro 9.3). A Demanda é calculada a partir dos pedidos programados para o período. As colunas Estoque Final e Necessidade de Compras são calculadas como indicado a seguir:

$\text{Estoque Final} = \text{Estoque Inicial} - \text{Demanda}$

Caso resulte negativo, então $\text{Estoque Final} = 0$

$\text{Neces.Compras} = \text{Demanda} - \text{Estoque Inicial} + \text{Estoque Mínimo}$

O trabalho maior, neste caso, é calcular a Demanda. Para isso, as quantidades constantes nos pedidos devem ser agrupadas por produto e somadas. Desse agrupamento e totalização resulta a Demanda, que é a quantidade de produtos que deve ser entregue no período. Isto está demonstrado no Quadro 9.4, no qual os dados foram reordenados, alinhados, e os totais, calculados. Isso foi feito com o único propósito de demonstrar o cálculo necessário para gerar o resultado da saída. No programa a solução é um pouco diferente, pois não serão feitos esse agrupamento e essa ordenação.



```
2017 5 1 12100 475 3.42
2017 5 2 12100 254 3.19
2017 5 2 12100 251 3.26
2017 5 3 12100 364 3.19
Demanda 1344
2017 5 2 12200 37 27.39
Demanda 37
2017 5 1 15200 87 68.18
2017 5 3 15200 47 71.90
2017 5 3 15200 59 68.18
2017 5 5 15200 48 69.42
Demanda 241
```

Quadro 9.4 Exemplo de arquivo VENDAS.TXT.

Saída – segunda parte: totais e margem média

A segunda parte da saída contém, para cada produto, a demanda (cujo cálculo foi exemplificado no Quadro 9.4) e o valor total calculado multiplicando-se a quantidade da venda pelo preço unitário e totalizando esse valor de maneira semelhante ao que foi feito com a quantidade. Os valores totais dos produtos são totalizados para obtenção do Total Geral.

Para cada produto é possível calcular o preço médio praticado. Isso porque cada venda individual terá o próprio valor unitário. Esse preço médio, comparado ao preço de custo do produto, produz a margem média que a empresa ganha referente a cada produto. Essas duas colunas são calculadas como se segue. O Quadro 9.5 ilustra esses cálculos.

$$\text{Preço Médio} = \text{Valor Total} / \text{Qtde}$$
$$\text{Margem Média} = \text{Preço Médio} / \text{Preço Custo} * 100\%$$


TOTAIS DE PEDIDOS EM CARTEIRA					
Prod.	Valor Tot	Qtde	Pç Médio	Pç Custo	Margem Méd
12100	4414.18	1344	3.28	2.30	42.8%
12200	1013.43	37	27.39	23.70	15.6%
15100	0.00	0	0.00	5.60	0.0%
15200	16665.74	241	69.15	61.90	11.7%
Total	22093.35				

Quadro 9.5 Exemplo de arquivo VENDAS.TXT.

A solução – o programa apurador.py

Agora que o problema já está bem definido, é possível começar a tratar da solução. O primeiro passo é dispor dos arquivos de entrada PRODUTOS.TXT e VENDAS.TXT. Para isso, pode digitar os dados dos Quadros 9.1 e 9.2 e salvá-los com os respectivos nomes. Mais à frente será mostrado um programa capaz de gerar dados de vendas em grande volume.

Parte principal

Esse programa foi implementado utilizando-se diversas funções. A parte principal do programa, que é o ponto de início do programa, está mostrada no Exemplo 9.1.

Exemplo 9.1 Programa apurador.py – parte principal # Ponto de início da execução
ExibeApresentacao() # 1 exibe a tela inicial

Prods = LeArqProdutos() # 2 lê e retorna o arquivo de produtos Vendas = LeArqVendas()
3 lê e retorna os dados de vendas

abaixo abre o arquivo de saída

arqsai = open("APURA.TXT", "w", encoding="UTF-8") # 4 ApuraDemandaEstoque() # 5
apura a demanda e neces. compras ApuraTotaisPorProduto() # 6 apura os totais vendidos
arqsai.close() # 7 fecha o arquivo de saída

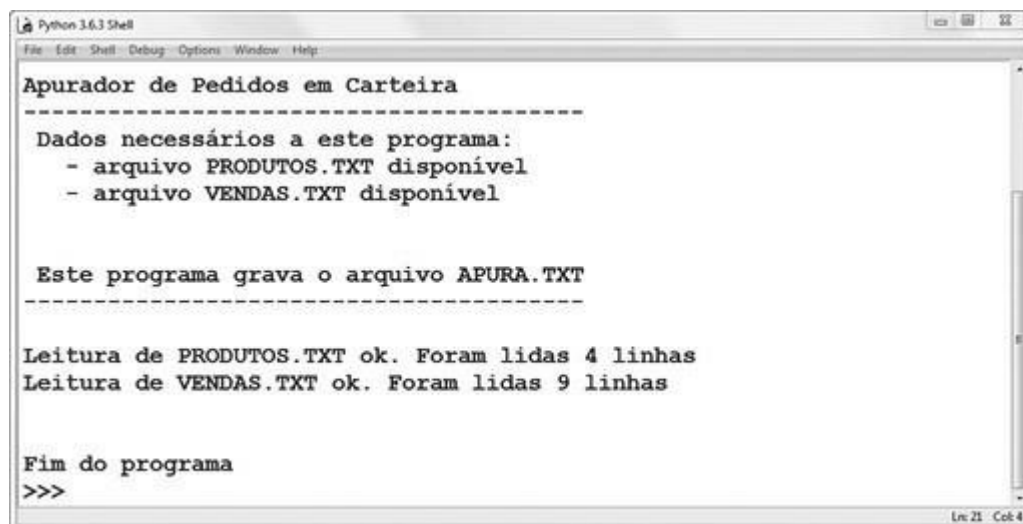
print("\n\nFim do programa")

Essa parte principal consiste em sete linhas de código nas quais são chamadas diversas funções, explicadas na sequência. Uma tela inicial é exibida na função ExibeApresentacao.



Em seguida, nas linhas 2 e 3, são feitas as leituras dos arquivos de entrada, cujos dados ficarão armazenados nos objetos globais Prods e Vendas. Essas variáveis, depois, serão utilizadas dentro de duas funções.

Na linha 4 o arquivo de saída é aberto, sendo fechado apenas na linha 7. Este arquivo será gravado dentro das funções ApuraDemandaEstoque e ApuraTotaisPorProduto.



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Apurador de Pedidos em Carteira
-----
Dados necessários a este programa:
- arquivo PRODUTOS.TXT disponível
- arquivo VENDAS.TXT disponível

Este programa grava o arquivo APURA.TXT
-----

Leitura de PRODUTOS.TXT ok. Foram lidas 4 linhas
Leitura de VENDAS.TXT ok. Foram lidas 9 linhas

Fim do programa
>>>
```

Execução do programa apurador.py.

A Figura 9.1 exibe o resultado da execução desse programa. Não há interação com o usuário. O programa simplesmente lê os arquivos de entrada, processa e encerra.

A função ExibeApresentacao é composta por diversos prints e mais nada. Ela monta um visual básico para informar ao usuário do programa que a execução ocorreu.

Exemplo 9.2 Programa apurador.py – função ExibeApresentacao def ExibeApresentacao():

```
print("\nApurador de Pedidos em Carteira")
print("-" * 40)
```

```
print(" Dados necessários a este programa:") print(" - arquivo PRODUTOS.TXT
disponível") print(" - arquivo VENDAS.TXT disponível") print("\n")
print(" Este programa grava o arquivo APURA.TXT") print("-" * 40, "\n")
```

Função LeArqProdutos

O objetivo desta função é ler o arquivo PRODUTOS.TXT e gerar o dicionário Prods contendo os dados dos produtos. Esse é um dicionário que contém dicionários aninhados, no



qual a chave é o código do produto e o valor associado é outro dicionário contendo os strings estq, qmin, pcunit e margem como chaves às quais estarão associados os valores lidos do arquivo.

```
Prods[12100]
Prods = {12100: {'estq': 1417, 'qmin': 500, 'pcunit': 2.3, 'margem': 38.8},
        12200: {'estq': 725, 'qmin': 100, 'pcunit': 23.7, 'margem': 13.58},
        15100: {'estq': 618, 'qmin': 500, 'pcunit': 5.6, 'margem': 34.9},
        15200: {'estq': 223, 'qmin': 50, 'pcunit': 61.9, 'margem': 8.15}
}
```

Estrutura do dicionário Prods.

A Figura 9.2 mostra como ficará o dicionário de produtos. Para se chegar a isso, a função do Exemplo 9.3 define o objeto dicProd como um dicionário vazio. Em seguida, abre o arquivo e usa um laço iterador que lê uma linha por vez, carregando no objeto S o string lido (linha 3).

Cada string é processado como descrito nos comentários do código. Em especial, tem-se que na linha 7 o dicionário aninhado é criado e nas linhas 8 a

11 ele é carregado. Feito isso, na linha 12 o dicionário de produtos é carregado com o item dicItem como valor vinculado ao código do produto.

Exemplo 9.3 Programa apurador.py – Função LeArqProdutos

```
def LeArqProdutos():
    dicProd = {} # cria o dicionário vazio
```

```
    arq = open("PRODUTOS.TXT") # abre o arquivo de produtos
    for S in arq.readlines(): # linha 3
```

```
        S = S.rstrip() # remove o '\n' do string lido
```

```
        L = S.split(";") # separa o string em uma lista
        codigo = int(L[0]) # obtém o cód. do produto
        dicItem = {} # linha 7
```

```
        dicItem['estq'] = int(L[1]) # linha 8
        dicItem['qmin'] = int(L[2]) # linha 9
        dicItem['pcunit'] = float(L[3]) # linha 10
        dicItem['margem'] = float(L[4]) # linha 11
        dicProd[codigo] = dicItem # linha 12
    arq.close() # fecha o arquivo
    print("Leitura de PRODUTOS.TXT ok. Foram lidas {} \n linhas".format(len(dicProd)))
```

```
    return dicProd # retorna o dicionário
```



Função LeArqVendas

As vendas já não podem ser armazenadas na forma dicionários aninhados, pois não há um campo que possa servir como chave do dicionário. Com isso, elas serão armazenadas como uma lista de tuplas. Dentro de cada tupla os dados de vendas estarão exatamente na mesma ordem em que aparecem no arquivo de entrada: ano, mês, dia, código produto, quantidade vendida e preço unitário de venda.

```
[ (2017, 5, 1, 12100, 475, 3.42),  
  (2017, 5, 1, 15200, 87, 68.18),  
  (2017, 5, 2, 12100, 254, 3.19),  
  (2017, 5, 2, 12200, 37, 27.39),  
  ...  
  (2017, 5, 5, 15200, 48, 69.42) ]
```

Figura 9.3 Estrutura da lista Vendas.

Essa função faz a leitura do arquivo de vendas, sendo que para cada linha elimina o '\n', faz um split no string, separando os dados na lista L, executa a conversão dos dados de string para inteiro ou real, no caso do preço, e, por fim, à lista V é adicionada a lista L convertida para tupla. Essa lista V é retornada pela função LeArqVendas.

Exemplo 9.4 Programa apurador.py – função LeArqVendas

```
V = [] # define a lista
```

```
arq = open("VENDAS.TXT") # abre o arquivo
```

```
for s in arq.readlines(): # para cada linha do arquivo s = s.rstrip() # elimina '\n' do final
```

```
L = s.split(';') # separa o string em uma lista for i in range(6): # converte os dados para int
```

```
if i < 5: # ou float no caso do último
```

```
    L[i] = int(L[i]) else:
```

```
    L[i] = float(L[i])
```

```
V.append(tuple(L)) # acrescenta a tupla à lista. arq.close()
```

```
print("Leitura de VENDAS.TXT ok. Foram lidas {} \nlinhas".format(len(V)))
```

```
return V # retorna V
```



Função ApuraDemandaEstoque

Esta é a função responsável pela apuração da demanda de quantidades e da eventual necessidade de compras. Na linha 2 são declaradas as três variáveis globais que serão utilizadas. O elemento-chave desta função é o dicionário aninhado dicEstq, construído nas linhas de 3 a 9 e que tem a seguinte estrutura:

```
dicEstq = {cód.produto:{‘estq’:0, ‘qmin’:0, ‘demanda’:0}, ...}
```

Nela, o código do produto é chave para o dicionário aninhado, que conta com as chaves estq, qmin e demanda. Os dois primeiros são obtidos diretamente do cadastro de produtos, e a demanda é iniciada com zero.

Nas linhas seguintes, de 11 a 13, é feita a interação com o dicionário Vendas e as quantidades são totalizadas na chave demanda (linha 13). Depois, a função implementa os cálculos já explicados no Item 9.1.2 (linhas 25 a 32) e grava cada linha no arquivo arqsai.

Exemplo 9.5 Programa apurador.py – função ApuraDemandaEstoque def ApuraDemandaEstoque():

```
global Prods, Vendas, arqsai # linha 2 dicEstq = {} # linha 3
for codprod in Prods.keys(): dicItem = {}
dicItem[‘estq’] = Prods[codprod][‘estq’] dicItem[‘qmin’] = Prods[codprod][‘qmin’]
dicItem[‘demanda’] = 0
dicEstq[codprod] = dicItem # linha 9 for v in Vendas: # linha 11
codprod = v[3]

dicEstq[codprod][‘demanda’] += v[4] # linha 13 arqsai.write("-"*52 + " Início de Bloco -" +
"\n")
arqsai.write("NECESSIDADE DE ESTOQUE NO PERÍODO\n")

arqsai.write("-"*70 + "\n")

arqsai.write(" "*9 + "Estoque " + " "*14 + "Estoque" + " "*4 + "Estoque" + " "*6 + "Neces.\n")
arqsai.write("Prod. Inicial Demanda" + " "*6 + "Final Mínimo Compra\n")
sgrava = "{:<5} {:>10d} {:>10d} {:>10d} {:>10d} {:>10d}\n"

for codprod, dados in dicEstq.items():
```



```
# bloco de cálculo desta função # linha 25 EstqFinal = dados['estq'] - dados['demanda'] if
EstqFinal < 0:
    EstqFinal = 0
```

```
NecCompra = dados['demanda'] - dados['estq'] + dados['qmin'] if NecCompra < 0:
NecCompra = 0
```

```
# fim do bloco de cálculo desta função # linha 32 arqsai.write(sgrava.format(
codprod, dados['estq'], dados['demanda'], EstqFinal, dados['qmin'], NecCompra))
arqsai.write("-"*55 + " Fim de Bloco -" + "\n\n\n")
```

Função ApuraTotaisPorProduto

Nesta função o objetivo é totalizar, para cada produto, as quantidades e os valores das vendas. A lógica é muito semelhante à da função anterior. Foi criado o dicionário dicTotais, que terá como chave os códigos de produtos e como valor um dicionário aninhado, o qual tem dois campos: totval, para o total do pedido, e totqtd, para o total da quantidade (linhas de 3 a 8).

```
dicTotais = {cód.produto:{'totval':0, 'totqtd':0}, ...}
```

Na sequência, as linhas 9 a 12 implementam o laço de iteração com a lista de vendas, realizando as totalizações necessárias nas linhas 11 e 12.

Exemplo 9.6 Programa apurador.py – função ApuraTotaisPorProduto def ApuraTotaisPorProduto():

```
global Prods, Vendas, arqsai # linha 2 dicTotais = {} # linha 3
for codprod in Prods.keys():

    dicItem = {} dicItem['totval'] = 0
    dicItem['totqtd'] = 0 dicTotais[codprod] = dicItem # linha 8 for v in Vendas: # linha 9
    codprod = v[3] dicTotais[codprod]['totval'] += v[4] * v[5]
    dicTotais[codprod]['totqtd'] += v[4] # linha 12 arqsai.write("-"*52 + " Início de Bloco -" + "\n")
arqsai.write("TOTAIS DE PEDIDOS EM CARTEIRA\n") arqsai.write("-"*70 + "\n") # linha 16
arqsai.write("Prod. Valor Tot Qtde " + "Pç Médio Pç Custo Margem Méd\n")
```



```
sgrava = "{:<5} {:>11.2f} {:>8d} {:>10.2f} {:>10.2f} \
```

```
{:>10.1f}%\n"
```

```
TotVendas = 0 # linha 21
```

```
for codprod, dados in dicTotais.items(): try:
```

```
TotVendas += dados['totval']
```

```
pcmedio = dados['totval'] / dados['totqtd'] lucrat = (pcmedio / dados['pcunit'] - 1) * 100
```

```
except:
```

```
pcmedio = lucrat = 0 arqsai.write(sgrava.format( codprod,  
dados['totval'],
```

```
dados['totqtd'], pcmedio,  
Prods[codprod]['pcunit'], lucrat))  
arqsai.write("-"*70 + "\n")
```

```
arqsai.write("Total {:>11.2f}\n".format(TotVendas)) arqsai.write("-"*55 + " Fim de Bloco -" +  
"\n\n\n")
```

Nota

Observe que, no código anterior, algumas linhas (17 e 19) ficaram muito extensas e foi necessário fazer uma quebra de linha. Para que o interpretador Python entenda essa quebra de linha, é necessário usar o caractere '\n' no final da linha quebrada.

Nas demais linhas dessa função é feita a gravação da saída no arquivo em disco, sendo que para isso são implementados os cálculos explicados no Item 9.1.3.

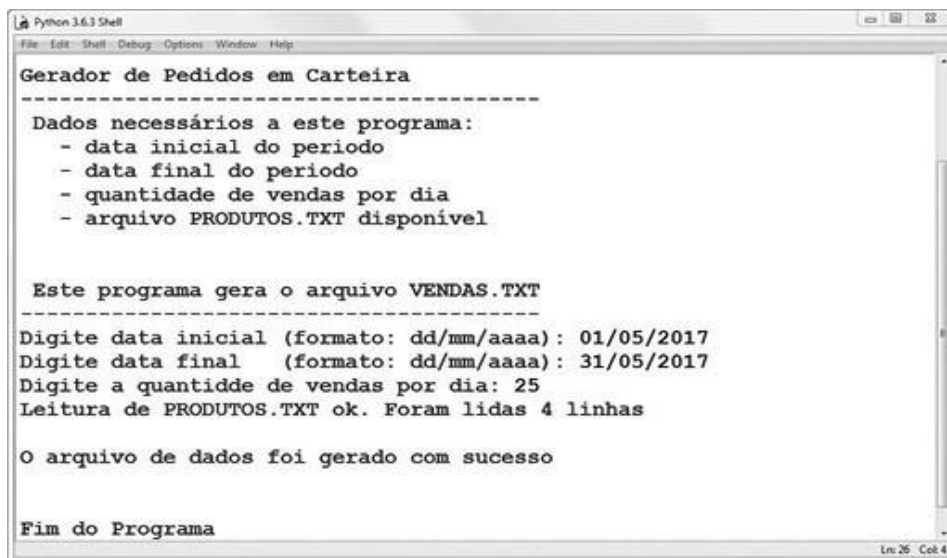
Para que consiga executar este programa, é necessário escrever todo o código, programa principal e todas as funções, em um único arquivo, sendo que as funções devem estar posicionadas antes do programa principal.

A solução – o programa gerador.py



Na descrição deste problema, Item 9.1, foi utilizado um arquivo VENDAS.TXT com apenas 9 linhas. Isso foi feito para ser possível explicar o problema e mostrar com um exemplo numérico o que deve ser calculado. Porém, convém testar o programa com diversos conjuntos de dados, que devem abranger períodos bem maiores, com muitas vendas diárias e, além disso, ser consistentes.

A melhor opção para dispor de arquivos assim é escrever um programa capaz de criá-los. Esta é a finalidade do programa gerador.py. Ao executá-lo, o usuário precisa fornecer três informações: as datas inicial e final do período e a quantidade de vendas por dia (essa quantidade varia em uma empresa real, mas aqui é razoável trabalhar com uma quantidade fixa).



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help

Gerador de Pedidos em Carteira
-----
Dados necessários a este programa:
- data inicial do periodo
- data final do periodo
- quantidade de vendas por dia
- arquivo PRODUTOS.TXT disponível

Este programa gera o arquivo VENDAS.TXT
-----
Digite data inicial (formato: dd/mm/aaaa): 01/05/2017
Digite data final (formato: dd/mm/aaaa): 31/05/2017
Digite a quantidde de vendas por dia: 25
Leitura de PRODUTOS.TXT ok. Foram lidas 4 linhas

O arquivo de dados foi gerado com sucesso

Fim do Programa
Ln: 26 Col: 4
```

Figura 9.4 Resultado da execução do programa gerador.py.

A Figura 9.4 exibe a tela de execução desse programa e o Exemplo 9.7 é o programa completo. Esse programa também foi escrito com o uso de funções, e a seguir é feito o detalhamento dele.

Descrição do programa – parte principal

O programa principal inicia exibindo uma tela de apresentação por meio da função `ExibeApresentacao` e é igual ao que foi feito no primeiro programa.

A função `ObtemEntradas` efetua a leitura dos dados a serem digitados pelo usuário e os retorna em uma tupla com a data inicial, a data final e a quantidade de vendas por dia, que são associadas aos objetos `DtIni`, `DtFim` e `Qtde`.



O próximo passo é efetuar a leitura do arquivo de produtos e a consequente carga do dicionário Prods. É exatamente igual ao que foi feito no primeiro programa e está descrito no Item 9.2.2.

Agora começa a parte essencial do programa. Trata-se de um laço que percorrerá todos os dias, desde a data inicial até a data final. Isso é feito com o auxílio dos objetos e funções da biblioteca datetime importada no início do código. Observe a linha. Para somar um dia em uma data e, com isso, obter a data do dia seguinte, é preciso usar o método datetime.timedelta(days = 1). Para simplificar o código, foi criado o objeto UmDia já carregado com o timedelta de um dia. Dentro do laço controlado pelas datas, é feita a verificação se é dia útil ou não. Para isso, utiliza-se o método datetime.weekday, que retorna 0 para segunda-feira, 1 para terça, e assim por diante. Portanto, quando esse retorno for 5 (sábado) ou 6 (domingo), o programa não deve gerar vendas.

E assim chega-se à função GeraDadosDia, que é a responsável pela efetiva geração dos dados. Ela utiliza, basicamente, a geração de números aleatórios para:

1. Sortear um produto gerando um número inteiro aleatório é utilizado como índice da lista de códigos de produtos. Essa lista de códigos de produto foi gerada a partir das chaves do dicionário Prods.
2. Gerar a quantidade vendida, segundo as proporções descritas no Item 9.1.1.
3. Gerar o valor unitário de venda a partir do preço unitário de compra e da margem mínima que constam do arquivo de produtos. Além disso, há uma variação entre 0% e 10% adicionais que são somados à margem, aumentando-a. Esse dado também é gerado aleatoriamente.

Após tudo ter sido gerado, a função efetua a gravação no disco.

Esse programa pode ser usado para gerar dados referentes a quaisquer períodos e para qualquer quantidade de vendas por dia. E esses dados podem ser usados para testar o programa **apurador.py**.

Exemplo 9.7 Programa gerador.py – programa completo

```
import datetime
from random import randint
def ExibeApresentacao():
    print("\nGerador de Pedidos em Carteira")
    print("-" * 40)
    print("Dados necessários a este programa:")
    print("- data inicial do período")
    print("- data final do período")
```




```
print(" - quantidade de vendas por dia")
```

```
print(" - arquivo PRODUTOS.TXT disponível") print("\n")
```

```
print(" Este programa gera o arquivo VENDAS.TXT") print("-" * 40)
```

```
def ConverteData(d): d = d.split("/")
```

```
data = datetime.date(int(d[2]), int(d[1]), int(d[0])) return data
```

```
def ObtemEntradas():
```

```
s = input("Digite data inicial (formato: dd/mm/aaaa): ") ini = ConverteData(s)
```

```
s = input("Digite data final (formato: dd/mm/aaaa): ") fim = ConverteData(s)
```

```
q = int(input("Digite a quantidade de vendas por dia: ")) return ini, fim, q
```

```
def LeArqProdutos(): dicProd = {}
```

```
arq = open("PRODUTOS.TXT")
```

```
for S in arq.readlines(): S = S.rstrip()
```

```
L = S.split(";") codigo = int(L[0]) dicltem = {}
```

```
dicltem['estq'] = int(L[1])
```

```
dicltem['qmin'] = int(L[2]) dicltem['pcunit'] = float(L[3]) dicltem['margem'] = float(L[4])
```

```
dicProd[codigo] = dicltem arq.close()
```

```
print("Leitura de PRODUTOS.TXT ok. Foram lidas {} linhas".format(len(dicProd)))
```

```
return dicProd
```

```
def GeraQtdeVenda(codprod): global Prods
```

```
sorteio = randint(1, 100) if sorteio <= 60:
```

```
q = randint(1, 10) elif sorteio <= 85: q = randint(11, 25) else:
```

```
q = randint(26, 400) return q
```

```
def GeraPcUnitVenda(codprod):
```

```
global Prods
```

```
pccompra = Prods[codprod]['pcunit'] margem = Prods[codprod]['margem'] / 100 variacao =
```



```
randint(0, 10) / 100
```

```
pcvenda = pccompra * (1 + margem + variacao) return pcvenda
```

```
def GeraDadosDia(dia, qtvendas):
```

```
    global Prods, arq
```

```
    L = list(Prods.keys()) # carrega a lista L com os cód. prod. for x in range(qtvendas):
```

```
    # sorteia produto, gera um índice e o utiliza na lista iprod = randint(0, len(Prods)-1)
```

```
    codprod = L[iprod]
```

```
    # randomiza a quantidade vendida
```

```
    qtitem = GeraQtdeVenda(codprod) # randomiza o preco de venda
```

```
    pcunit = GeraPcUnitVenda(codprod)
```

```
    a = str(dia.year)+';' +str(dia.month)+';' +str(dia.day) a = a + ';' + str(codprod)
```

```
    a = a + ';' + "{:d}".format(qtitem)
```

```
    a = a + ';' + "{:.2f}".format(pcunit) a = a + "\n"
```

```
    arq.write(a)
```

```
    # Ponto de início da execução ExibeApresentacao()
```

```
    DtIni, DtFim, Qtde = ObtemEntradas() Prods = LeArqProdutos()
```

```
    arq = open("VENDAS.TXT", 'w', encoding="UTF-8") UmDia = datetime.timedelta(days=1)
```

```
    Cont = DtIni
```

```
    while Cont <= DtFim:
```



```
if Cont.weekday() < 5: # só permite dia útil GeraDadosDia(Cont, Qtde)
Cont = Cont + UmDia arq.close()
print("\nO arquivo de dados foi gerado com sucesso")

print("\n\nFim do Programa")
```

Projeto 2: Controle de Torneios Esportivos

Problema

Motivação

Pouco tempo depois de ter concluído o curso de Processamento de Dados da Fatec São Paulo, um amigo perguntou ao autor deste capítulo se seria fácil desenvolver um programa de computador para controlar a classificação dos times participantes de um campeonato de futebol de salão, que utilizava o sistema de pontos corridos. O ponto de partida para a obtenção da classificação seriam os resultados dos jogos já realizados. À época estudante de Educação Física, ele gerenciava dois torneios de futsal cuja regra de disputa era essa de pontos corridos, semelhante ao que atualmente se pratica no campeonato brasileiro. Naquela época ele fazia todo o trabalho à mão e sabia bem do que precisava. O autor topou, o amigo explicou as regras e o resultado foi um programa escrito em linguagem Pascal que fazia tudo de que ele precisava.

A proposta deste capítulo é implementar um programa semelhante, porém, escrito em Python.

Descrição do torneio de pontos corridos

Em um torneio de pontos corridos, todos os times jogam entre si em rodadas sucessivas, podendo existir um ou dois turnos. No caso de apenas um turno, há apenas uma partida entre dois times. Por exemplo, isso ocorre na fase de grupos da Copa do Mundo, em que, dentro de cada grupo, quatro seleções jogam entre si. Quando há dois turnos, como no caso do Brasileirão, dois times se enfrentam duas vezes, sendo que em uma o time é o mandante, dono do campo e conta com maior torcida a favor, e na outra é visitante. Como o amigo do autor deste capítulo gerencia torneios dos dois tipos, isso precisa estar previsto no programa.

Em torneios assim há alguns números que são relevantes à estrutura do programa. Todos esses números dependem da quantidade de times participantes do torneio, que será



denominada N. O Quadro 10.1 exemplifica esses números.

Sendo N o número de times	N é par	N é impar	ex. N = 8	ex. N = 9
Quantidades de jogos por rodada	$\frac{N}{2}$	$\frac{(N-1)}{2}$	4	4 (um time folga)
Quantidade de rodadas por turno (no caso de 2 turnos, dobrar esse nº)	N - 1	N	7	9
Quantidade de jogos por turno (no caso de 2 turnos, dobrar esse nº)	$\frac{(N^2 - N)}{2}$	$\frac{(N^2 - N)}{2}$	28	36

Quadro 10.1 Números principais de um torneio por pontos corridos.

Em campeonatos de grande visibilidade, nos quais muitos times querem participar, é praxe que o número de times participantes seja par, pois equilibra o emparelhamento em cada rodada. Nos torneios gerenciados pelo amigo do autor, nem sempre estão inscritos times em número par, mas os torneios serão disputados de qualquer maneira. Isso não chega a ser um problema, não havendo nada que impeça um torneio com um número ímpar de times. O único detalhe a ser considerado em casos assim é que, a cada rodada, um dos times não joga, fica de folga.

Neste tipo de campeonato, à medida que os jogos transcorrem, os times são classificados de acordo com a quantidade de pontos conquistados. A vitória confere 3 pontos ao vencedor e o derrotado não pontua. Em caso de empate, cada time recebe 1 ponto. Quanto mais pontos um time tiver, mais bem colocado ele estará, e em caso de dois ou mais times com a mesma quantidade de pontos são adotados outros critérios de classificação, na seguinte ordem: maior número de vitórias, maior saldo de gols, maior número de gols marcados e confronto direto. O campeonato brasileiro ainda conta com critérios adicionais, como menor número de cartões vermelhos e amarelos, mas esse tipo de parâmetro não será considerado aqui.

Requisitos do programa

Essa é uma visão geral do tipo de torneio ao qual está dirigido este projeto. Para atendê-lo, o programa a ser desenvolvido deve contar com as funcionalidades descritas no Quadro 10.2.



Funcionalidade	Detalhes
Criação de torneio	<ol style="list-style-type: none">1. Obtém o nome do torneio e a quantidade de turnos.2. Os nomes dos times devem ser lidos do teclado.3. Todos os jogos devem ser gerados e organizados em rodadas.4. Cada jogo deve ser identificado por um número único.5. Pode haver qualquer número de times maior que 2.
Exclusão de torneio	<ol style="list-style-type: none">1. Torneios passados, sobre os quais já não há mais interesse, podem ser excluídos.
Gestão de torneio	<ol style="list-style-type: none">1. Para cada time o programa deve computar os pontos, o número de vitórias, empates e derrotas, gols pró e contra e o saldo de gols.2. O programa deve mostrar a classificação geral, que deve levar em conta o conjunto de critérios a seguir. Fica na frente quem tiver:<ul style="list-style-type: none">• maior número de pontos;• maior número de vitórias;3. Deve ser possível informar o resultado de cada jogo, via teclado.4. Deve ser possível eliminar um resultado de jogo, para ser utilizado em caso de erro de digitação.5. O programa deve gerar um arquivo HTML com a classificação para que seja publicado em um website.

Quadro 10.2 Funcionalidades que devem estar disponíveis no programa deste projeto.

A solução

A solução desse projeto está implementada em um programa chamado “torneio.py” e será descrita em três partes:



1. Modo de armazenamento dos dados do programa.
2. As telas e suas funcionalidades.
3. A implementação do programa, contendo a descrição do código desenvolvido.

Apresentação das telas do programa

Para iniciar, a apresentação do programa, será feita a partir da descrição das telas que este contém, dando uma visão geral de seu uso. São telas simples, construídas para serem visualizadas em modo console, ou seja, nada mais são do que telas em modo texto, formatadas exclusivamente com o uso do comando print.

A Figura 10.1 exibe a tela inicial, na qual é possível criar um novo torneio, escolher um torneio para gerenciar e sair do sistema. A escolha da opção é feita digitando-se a letra ou número que aparece entre parênteses à esquerda de cada opção. No caso de letras, pode-se digitar maiúsculas ou minúsculas que o resultado é o mesmo. E, se for digitado algo inválido, nada acontece.

Observe na figura que há seis torneios cadastrados. Nestes exemplos os torneios 1, 2, 3 e 6 estão em andamento e os torneios 4 e 5 estão encerrados, com todos os jogos já realizados.



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
-----
Programa Torneio
Menu Principal
-----
Opções:
(N) Criar Novo Torneio
(.) Gerenciar Torneio Existente
    (1) Colégio Luchinni Handebol
    (2) Colégio Luchinni Infantil
    (3) Colégio Luchinni Juvenil
    (4) SP Oeste 2015
    (5) SP Oeste 2016
    (6) SP Oeste 2017
(S) Sair do programa

para escolher digite o que está entre parênteses
sua opção? >>>
```

Tela inicial do programa torneio.py – menu principal do programa.

A Figura 10.2 mostra a tela de gerenciamento de torneio. Escolheu-se, aqui, exibir o torneio denominado “SP Oeste 2016”. Nessa tela podem ser vistos os nomes dos times participantes, as quantidades de rodadas e de jogos e a classificação das equipes.

A tabela de classificação mostra a posição e os nomes dos times, os pontos ganhos (PG),



o número de jogos (J) já realizados, as vitórias (V), empates (E) e derrotas (D), os números de gols pró (GP), contra (GC) e o saldo de gols (SG).

```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
-----
Programa Torneio
Gerenciamento de Torneio
-----
-- Times deste Torneio SP Oeste 2016 --
-- Americano Backyard Castelão Caxingui FC --
-- Covil FC Portuguesa S.E. Bonfa Tormenta FC --
-- N° de Rodadas: 14 - N° de Jogos: 56 --
-----
Pos Time PG J V E D GP GC SG
-----
1 Backyard 26 14 8 2 4 29 24 5
2 S.E. Bonfa 25 14 8 1 5 27 23 4
3 Tormenta FC 24 14 7 3 4 23 14 9
4 Covil FC 19 14 6 1 7 22 24 -2
5 Caxingui FC 18 14 5 3 6 24 25 -1
6 Portuguesa 18 14 5 3 6 24 28 -4
7 Americano 17 14 5 2 7 22 25 -3
8 Castelão 13 14 4 1 9 20 28 -8
-----
Opções:
(.) Para ver uma rodada digite seu número.
Rodadas Válidas de 1 a 14
(G) Grava o Torneio em HTML
(E) Exclui o Torneio
(S) Voltar ao Menu Principal
sua opção? >>>
```

Tela de gerenciamento de torneio.

Essa tela também contém um menu com novas opções. Pode-se escolher uma rodada para visualizar, pode-se gravar um arquivo HTML com a tabela de classificação e pode-se excluir um torneio que já não é mais necessário.

Para ver uma rodada deve-se escolher seu número. A Figura 10.3 mostra a tela da rodada, na qual foi escolhida a última rodada, de número 14. Os resultados de cada partida já estão lançados. No caso de uma rodada ainda não jogada, nenhum número seria exibido.

Nessa tela é possível lançar o resultado de um jogo. Para isso, deve-se digitar o número do jogo e as quantidades de gols de cada time, separados por vírgulas. Por exemplo: ao digitar 53,2,1 significa que o jogo 53 teve resultado 2 a 1, ou seja, dois gols para o time à esquerda e um gol para o time à direita na tabela. Também é possível digitar: 53,limpa, o que significa que o resultado cadastrado para o jogo 53 deve ser apagado. Isso serve para corrigir eventuais erros




```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
-----
Programa Torneio
Gerenciamento de Torneio
-----
Times deste Torneio SP Oeste 2016
Backyard S.E. Bonfa Tormenta FC Covil FC
Caxingui FC Portuguesa Americano Castelh o
-----
N  de Rodadas: 14 - N  de Jogos: 56
-----
*** Rodada 14 ***
53 Castelh o 2 x 1 Americano
54 Caxingui FC 1 x 1 Covil FC
55 Portuguesa 2 x 4 Backyard
56 S.E. Bonfa 0 x 3 Tormenta FC
-----
Op  es:
(.) Para atualizar o placar de um jogo digite:
   N Jogo,GolsA,GolsB exemplo: 12,2,1
(.) Para limpar o placar de um jogo digite:
   N Jogo,limpa exemplo: 12,limpa
(S) Voltar ao Menu do Torneio
sua op   ? >>>
```

Tela de visualiza  o da rodada.

Voltando   Figura 10.2, ao usar a funcionalidade de excluir o torneio (op   o “E”), o programa exibe um pedido de confirma  o dessa exclus  o e, uma vez confirmada, todos os seus dados s o eliminados e o torneio desaparece da tela principal.

```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
-----
Confirma Exclus  o do Torneio SP Oeste 2016 -----
Op  es:
(C) para Confirmar
qualquer outra tecla para retornar
sua op   ? >>>
```

Confirma  o da exclus  o de torneio.

Por fim, nessa tela de gerenciamento, est  dispon  vel a op   o de grava  o da classifica  o do torneio em um arquivo HTML. Na vers o original desse programa, citada no in cio do cap  tulo, era feita a impress o da tabela e esta era enviada via fax para os l deres dos times. Com a internet, tudo ficou mais f cil e o amigo do autor deste cap  tulo sabe fazer o upload de arquivos HTML no provedor de internet que utiliza. Assim, no programa basta gravar o arquivo no disco do computador local. O upload para o local de hospedagem da p gina   feito manualmente. O HTML gravado tem a apar ncia mostrada na Figura 10.5.



Gerenciador de Torneio: x

Arquivo: C:\Users\Python\Documents\Exemplos_Codigo\Capitulo10\SP Oeste 2016.html

Torneio: SP Oeste 2016

Acompanhamento de Torneio atualizado em: 6/4/2018

Classificação

Pos	Time	PG	J	V	E	D	GP	GC	SG
1	Backyard	26	14	8	2	4	29	24	5
2	S.E. Bonfa	25	14	8	1	5	27	23	4
3	Tormenta FC	24	14	7	3	4	23	14	9
4	Covil FC	19	14	6	1	7	22	24	-2
5	Caxingui FC	18	14	5	3	6	24	25	-1
6	Portuguesa	18	14	5	3	6	24	28	-4
7	Americano	17	14	5	2	7	22	25	-3
8	Castelão	13	14	4	1	9	20	28	-8

Arquivo HTML com a classificação do torneio gerado pelo programa.

Voltando às opções do menu principal, a criação de um novo torneio é feita digitando-se “N”, e a Figura 10.6 mostra um exemplo no qual foi criado um torneio para a chave E da Copa do Mundo de 2018.

```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Nome do Novo Torneio: COPA 2018 - Chave E
Digite a quantidade de turnos (1 ou 2):
Digite 0 para cancelar.
quantos turnos? >>> 1

-----
Programa Torneio
Criação de Novo Torneio
-----

--
Novo Torneio: COPA 2018 - Chave E
(a qualquer momento digite 'sair' para desistir)
--

Digite os nomes dos times participantes
Digite 'fim' para concluir e salvar os nomes dos times
Time 1: Brasil
Time 2: Suíça
Time 3: Costa Rica
Time 4: Servia
Time 5: fim
```

Criação de um novo torneio.

A criação do torneio começa pela digitação de um nome para ele, seguido do número de turnos. Em seguida, digitam-se os nomes das equipes participantes, um a um. A qualquer momento, se for digitada a palavra “sair”, o processo é cancelado. Ao digitar “fim”, o programa entende que todos os times já foram inseridos, gera as rodadas de jogos e as salva no banco de dados. Depois disso, o novo torneio passa a estar presente no menu da tela inicial.

