

Disciplina	Turma	Descrição deste documento	Prazo de Entrega
IAL002 Algoritmos e Lógica de Programação	Noite	Arquivos em Python	Não há entrega

Gravação e Leitura de Arquivos texto em Python

De que se trata?

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 forma permanente e para isso devem ser gravados em algum equipamento de hardware que não necessite estar ligado permanentemente em uma fonte de energia.

Motivação

Toda linguagem de programação possui 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 de forma indireta acessando serviços do Sistema Operacional. 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 S.O. utiliza e, de outro, os comandos da linguagem que o programador deve usar para executar as operações.

Arquivos podem ser classificados em dois tipos, quanto à natureza de seu conteúdo

Arquivos texto

Neste tipo de arquivo os bytes são interpretados segundo uma tabela de caracteres e o conjunto resultante será uma cadeia de caracteres, ou string. 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

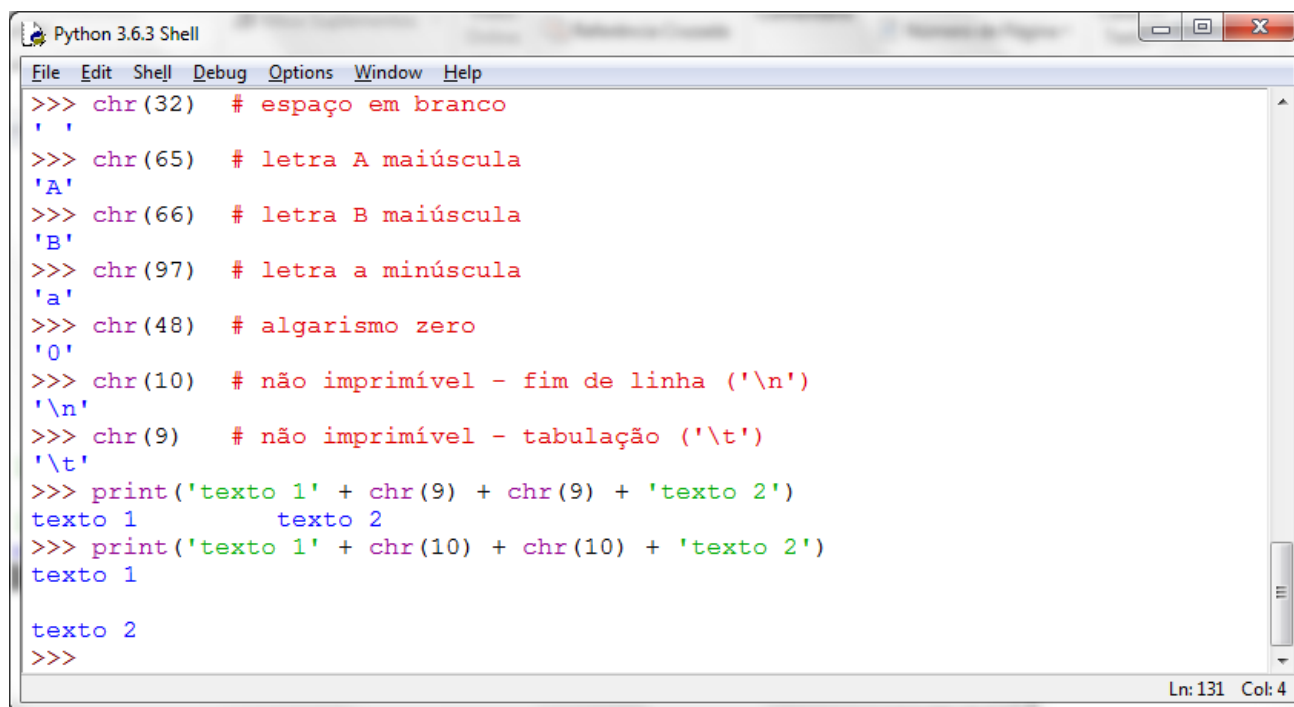
Neste caso os bytes são lidos e trazidos para a memória sem qualquer interpretação ou decodificação. São bytes em estado bruto e caberá ao programador escrever o programa de maneira apropriada a fim de interpretar corretamente tais bytes. Como exemplo de arquivos binários comuns tem-se: arquivos de imagens, áudio e vídeo; arquivos de bancos de dados; arquivo executável de um programa compilado; arquivos compactados através de um algoritmo de compressão, entre outros.

Neste documento vamos trabalhar apenas com arquivos texto no padrão ASCII.

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 usado na codificação dos caracteres. Com estes 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 possuem 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 correspondente 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.

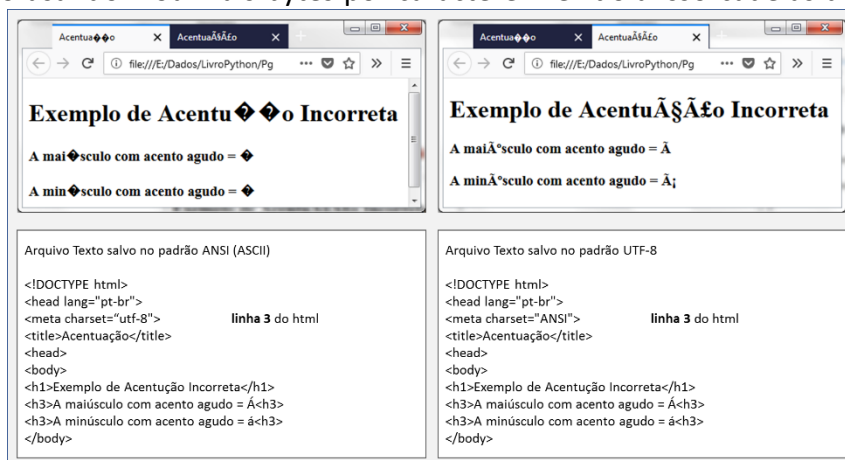


```
>>> chr(32) # espaço em branco
' '
>>> chr(65) # letra A maiúscula
'A'
>>> chr(66) # letra B maiúscula
'B'
>>> chr(97) # letra a minúscula
'a'
>>> chr(48) # algarismo zero
'0'
>>> chr(10) # não imprimível - fim de linha ('\n')
'\n'
>>> chr(9) # não imprimível - tabulação ('\t')
'\t'
>>> print('texto 1' + chr(9) + chr(9) + 'texto 2')
texto 1      texto 2
>>> print('texto 1' + chr(10) + chr(10) + 'texto 2')
texto 1

texto 2
>>>
```

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.

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). Este sistema permite a representação e manipulação de texto de forma 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 Unicode trabalha com a opção de codificação usando 1 ou mais bytes por caractere. Devido a isso cadeias de texto construídas usando-se a codificação Unicode são conhecidos no mundo da computação como "*wide-character strings*" ou "*wide-strings*". Conhecer estes conceitos é importante para o programador, uma vez que com frequência precisar deste conhecimento para não incorrer em erros comuns. A título de exemplo considere-se a figura ao lado. 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



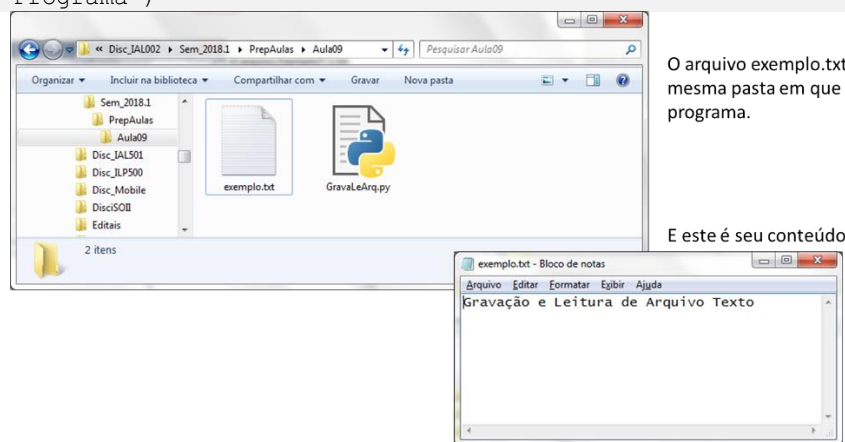
arquivo texto foi salvo com codificação ANSI, que usa a tabela ASCII. Do lado direito foi criada a situação inversa: ou seja, o html especifica que a codificação usada é "ANSI", mas o arquivo foi com codificação UTF-8.

Arquivos em Python

Considere-se o exemplo a seguir. Este programa tem duas partes. Na primeira grava-se o arquivo "exemplo.txt" e na segunda parte o mesmo arquivo é lido.

```
print("Início do Programa")
# Parte 1 - Gravação do arquivo
G = "Gravação e Leitura de Arquivo Texto" # carrega o string G
arq = open("exemplo.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("exemplo.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")
```



Copie este programa e execute-o para constatar a criação do arquivo.

Exemplo 1

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.

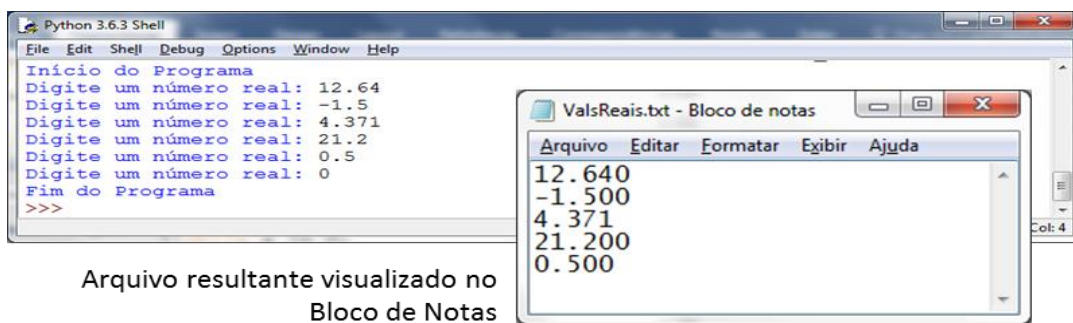
Serão criadas duas soluções para este exercício. Na primeira será usado o método `write()` e na segunda será usado o método `writelines()`.

Na linha 2 o arquivo é aberto. Na linha 5 é usado o método `write()` para efetuar a gravação. Como este 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.

Solução do Exemplo 1 – Versão A – Sequência de Números Reais gravada em arquivo com `write()`

```
print("Início do Programa")
arcreais = open('ValsReais.txt', 'w') # linha 2
x = float(input("Digite um número real: "))
while x != 0:
    arcreais.write("{0:.3f}\n".format(x)) # linha 5
    x = float(input("Digite um número real: "))
arcreais.close() # linha 7
print("Fim do Programa")
```

O resultado é mostrado na próxima figura.



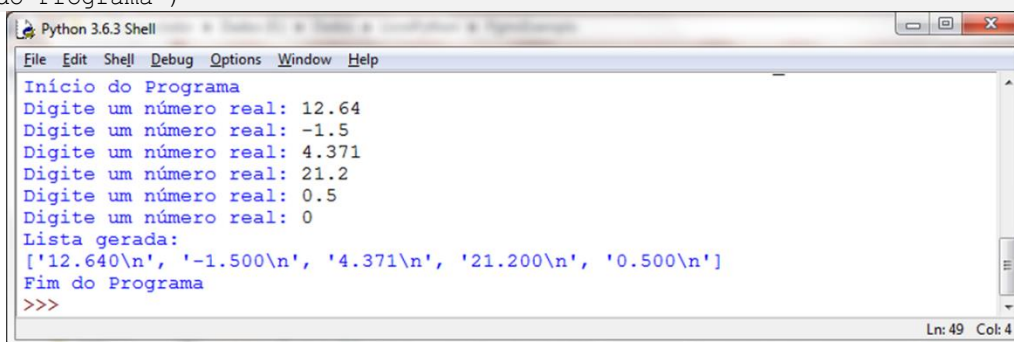
Arquivo resultante visualizado no
Bloco de Notas

A segunda solução para este enunciado está abaixo. Como o objetivo é usar o método `writelines()` é necessário produzir uma lista contendo os strings que serão gravados no disco. Isto está feito abaixo, onde o objeto `L` é criado como uma lista vazia e dentro do laço é carregada com os strings formatados do mesmo modo como o feito na solução versão A. Ao término do laço, o arquivo é aberto, gravado e fechado. O resultado produzido é o mesmo do exercício anterior.

Solução do Exemplo 1 – Versão B – Sequência de Números Reais gravada em arquivo com `writelines()`

```
print("Início do Programa")
L = []                                # inicia a lista vazia
x = float(input("Digite um número real: "))
while x != 0:
    L.append("{:.3f}\n".format(x))    # inclui o string em L
    x = float(input("Digite um número real: "))

arqreais = open('ValsReais2.txt', 'w') # abre o arquivo
arqreais.writelines(L)                # grava a lista toda
arqreais.close()                     # fecha o arquivo
print("Fim do Programa")
```



Exemplo 2

Escreva um programa que leia um arquivo texto contendo um número inteiro em cada linha. Exiba na tela e faça a totalização dos valores lidos.

Este exercício será resolvido de duas formas diferentes. Na primeira, Versão A, será utilizado um laço `while`. A leitura da primeira linha do arquivo é feita fora do laço. Caso seu retorno seja diferente de string vazio, o laço é iniciado e prosseguirá até que o final do arquivo seja atingido. O método `readline()` retorna um string vazio quando a leitura do arquivo chega ao final. A cada repetição do laço o string lido é convertido para inteiro e totalizado no objeto `Soma`.

Na conversão de `S` para número inteiro pode ocorrer um erro se `S` for um string vazio. Por isso foi usada a lógica de se fazer a primeira leitura fora do laço e as demais leituras como último comando dentro laço, evitando um `if` adicional.

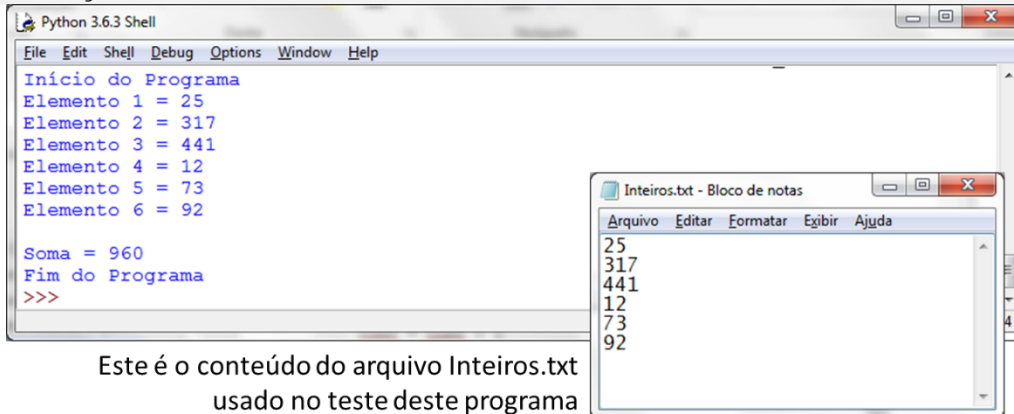
Solução do Exemplo 2 – Versão A – Leitura de arquivo e totalização de valores – solução com `while`

```
print("Início do Programa")
Soma = 0
```

```

Cont = 0
arq = open('Inteiros.txt', 'r')
S = arq.readline()          # garante que o laço seja iniciado
while S != '':
    N = int(S)               # converte o texto S para N
    Soma = Soma + N          # totaliza o valor N em Soma
    Cont = Cont + 1          # conta mais um elemento
    print("Elemento {0} = {1}".format(Cont, N))
    S = arq.readline()      # lê a próxima linha
arq.close()
print("\nSoma = {0}".format(Soma))
print("Fim do Programa")

```



A solução acima funcionará perfeitamente, porém existe outra solução bem melhor no que diz respeito ao uso dos recursos da linguagem Python. A Versão B abaixo mostra essa outra solução, na qual foi usado o conceito de **Iterador de Arquivo** (*file iterator*).

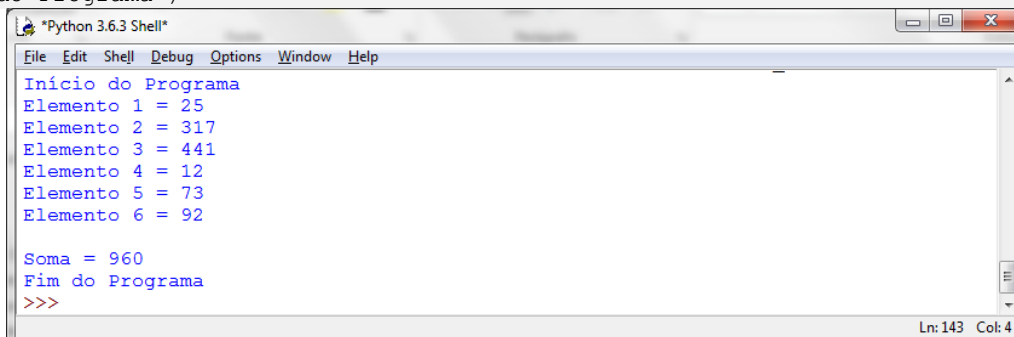
Quando o objetivo é percorrer todo o arquivo linha a linha, recuperando e processando uma linha por vez, então usar um iterador de arquivo é opção mais interessante pois o código fica mais enxuto, o interpretador gerencia melhor a memória e otimiza a execução, não é necessário fechar o arquivo pois isso será automático ao fim do processo. Também pode ser observado que não é necessário atribuir o retorno do comando open a um identificador. O iterador gerenciará um objeto temporário em memória dispensando um identificador.

Solução do Exemplo 2 – Versão B – Leitura de arquivo e totalização de valores – solução com iterador de arquivo

```

print("Início do Programa")
Soma = 0
Cont = 0
for S in open('Inteiros.txt'):      # note que 'r' foi omitido pois
    N = int(S)                      # é default
    Soma = Soma + N
    Cont = Cont + 1
    print("Elemento {0} = {1}".format(Cont, N))
print("\nSoma = {0}".format(Soma))
print("Fim do Programa")

```



As duas soluções, produzem exatamente o mesmo resultado. E em linhas gerais pode-se dizer que, enquanto a primeira talvez seja mais alinhada com a forma de pensar de um programador experiente em outras linguagens, a segunda reflete o jeito Python de ser. No jargão da comunidade Python, a segunda solução é *Pythonic*.

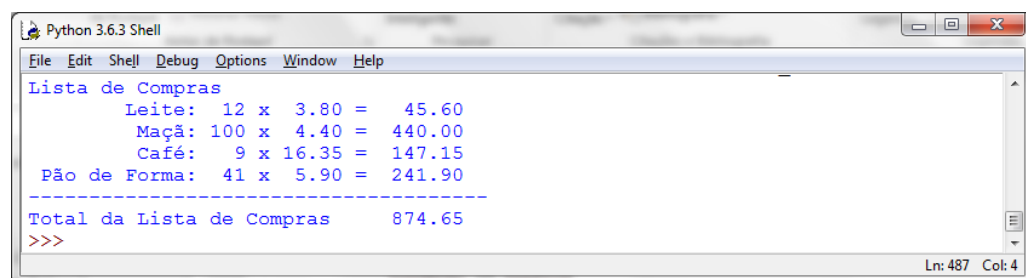
Exercícios Propostos

1. Escreva um programa que grave o arquivo NUMEROS.TXT com 2.000 números, um em cada linha, gerados com a função *randint* no intervalo fechado [1, 100.000].
2. Escreva um programa que leia um número inteiro N e grave em um arquivo em disco todos os números primos existentes no intervalo fechado [2, N], um número em cada linha. Sugestão: para verificar se um número é primo utilize as soluções desenvolvidas para o problema no. 8 da lista de exercícios da Aula 06 de 02/04/2018 (estão disponíveis na área de downloads do site do professor).
3. 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 abaixo.

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

Arquivos assim também são conhecidos pelo termo "comma separated values" (CSV) e muito usados para troca de dados entre diferentes sistemas.

O que se deseja é efetuar a leitura de um arquivo de entrada como esse e exibir as informações na tela como mostrado na figura abaixo. Ou seja, além de ler cada linha, será necessário separar as informações nela contidas e usar tais informações para calcular os totais.



```
Python 3.6.3 Shell  
File Edit Shell Debug Options Window Help  
Lista de Compras  
Leite: 12 x 3.80 = 45.60  
Maçã: 100 x 4.40 = 440.00  
Café: 9 x 16.35 = 147.15  
Pão de Forma: 41 x 5.90 = 241.90  
-----  
Total da Lista de Compras 874.65  
>>>
```

Para ler e separar corretamente os dados de um arquivo assim, será necessário usar o método `split()` pertencente aos objetos da classe `string`. Pesquise sobre isso e faça uns testes usando o IDLE.

4. Escreva um programa que leia um número inteiro N ($10 < N < 10.000$) e grave um arquivo com N linhas com os dados listados no quadro abaixo. 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.

Quadro - Formato para o arquivo do Exercício 4

Campo	Descrição
Código do produto	Número inteiro entre 10000 e 50000. Não pode haver repetição deste código e pede-se que não sejam sequenciais (devem ser 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)