



Trabalhando com Arquivos



Dados de um Programa

Durante sua execução, um programa típico processa um ou mais conjuntos de **dados**.

- ✓ dados de entrada (usados pelo programa) e dados de saída (gerados pelo programa)

Até agora, nossos programas usam dados diretamente inseridos no código ou digitados pelo usuário.

- ✓ o que não é conveniente se temos muitos dados de entrada, e/ou se precisamos **reusar** esses dados!



Arquivos

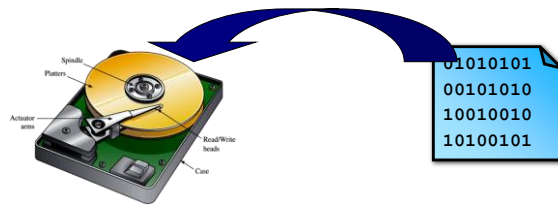
Guardam dados de diversos tipos de forma permanente.

- ✓ documentos, imagens, músicas, planilhas, programas fonte,

Podem ser armazenados em diferentes tipos de mídia:

- ✓ disco rígido, pendrive, CD-ROM, ...

Em sua forma mais básica, um arquivo é uma **sequência de bytes**, como a memória do computador.

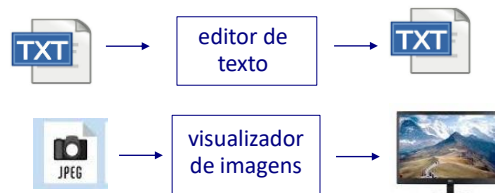


Formatos de Arquivos

De acordo com seu tipo, os dados podem ser guardados em diversos **formatos**.

- ✓ geralmente, a **extensão** no nome do arquivo indica o seu tipo e formato (imagem1.jpg, lista_alunos.txt, tart.py)

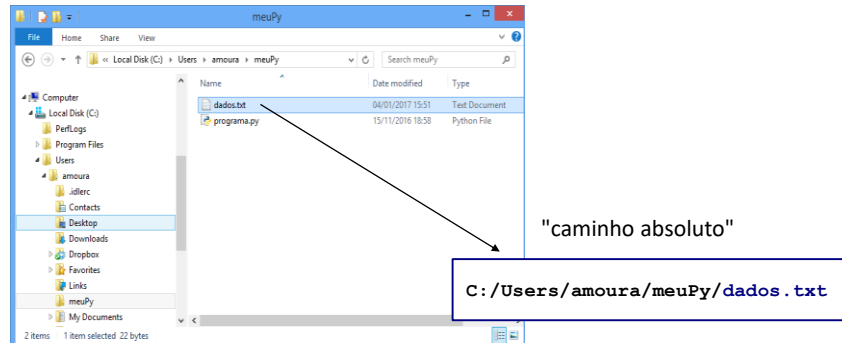
Para ler/escrever os dados de um arquivo, um programa precisa conhecer e saber trabalhar com seu formato!



Arquivos e Caminhos

Arquivos são organizados em **pastas** (diretórios).

Um arquivo é localizado pelo seu **caminho** (*path*).



Manipulando Arquivos

O uso de um arquivo envolve as seguintes etapas:

- ① **abrir** o arquivo: associar um arquivo físico a um **objeto arquivo**
- ② **manipular** o arquivo: ler ou gravar dados usando os métodos do objeto arquivo
- ③ **fechar** o arquivo

① → `arq = open("dados.txt", "r")`

② → `texto = arq.read()`
`print(texto)`

③ → `arq.close()`



Abrindo Arquivos

Para abrir o arquivo, o programa deve informar sua localização, especificando qual o seu **caminho** (*path*)

- ✓ se o arquivo e o programa estão em pastas (diretórios) diferentes, usamos o “caminho absoluto”

```
C:/Users/amoura/meuPy/dados.txt
```

- ✓ se o arquivo e o programa estão no mesmo diretório, podemos usar apenas o nome do arquivo (“caminho relativo”)



Abrir Arquivos: leitura ou escrita

Para abrir o arquivo, o programa deve informar sua localização, especificando qual o seu **caminho** (*path*)

- ✓ se o arquivo e o programa estão em pastas (diretórios) diferentes, usamos o “caminho absoluto”

```
C:/Users/amoura/meuPy/dados.txt
```

- ✓ se o arquivo e o programa estão no mesmo diretório, podemos usar apenas o nome do arquivo (“caminho relativo”)

Ao abrir o arquivo, informamos também se o usaremos para **leitura** ou para **escrita**



Variável Arquivo

Name	Date modified	Type
dados.txt	04/01/2017 15:51	Text Document
programa.py	15/11/2016 18:58	Python File

C:/Users/amoura/meuPy/dados.txt

```
arq_ler = open("C:/Users/amoura/meuPy/dados.txt", "r")
```

"r" indica que o arquivo será lido (read)

a variável **arq_ler** guarda a referência para o "objeto" arquivo



Leitura de Arquivo: "r"

Name	Date modified	Type
dados.txt	04/01/2017 15:51	Text Document
programa.py	15/11/2016 18:58	Python File

C:/Users/amoura/meuPy/dados.txt

```
arq_ler = open("C:/Users/amoura/meuPy/dados.txt", "r")
```

```
arq_ler = open("dados.txt", "r")
```

"r" indica que o arquivo será lido (read)

a variável **arq_ler** guarda a referência para o "objeto" arquivo



Escrita em Arquivo: "w"

Name	Date modified	Type
dados.txt	04/01/2017 15:51	Text Document
programa.py	15/11/2016 18:58	Python File

C:/Users/amoura/meuPy/dados.txt

```
arq_ler = open("C:/Users/amoura/meuPy/dados.txt", "r")
```

```
arq_ler = open("dados.txt", "r")
```

"r" indica que o arquivo será lido (read)

a variável **arq_ler** guarda a referência para o "objeto" arquivo

```
arq_esc = open("C:/Users/amoura/meuPy/dados.txt", "w")
```

```
arq_esc = open("dados.txt", "w")
```

"w" indica que o arquivo será escrito (write)

(se o arquivo existe, será sobrescrito; devemos usar "a" para escrever a partir do final de um arquivo já existente)



Fechando um Arquivo

Quando terminamos de usar um arquivo devemos fechá-lo:

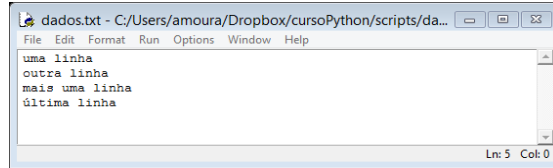
```
arq_ler = open("C:/Users/amoura/meuPy/dados.txt", "r")  
  
< usando o arquivo >  
  
arq_ler.close()
```

um objeto do tipo arquivo tem um método **close** que fecha o arquivo

Arquivos Texto

São arquivos que armazenam texto (caracteres).

- ✓ um arquivo texto é composto por uma sequência de linhas
- ✓ cada linha é uma sequência de caracteres que termina com um caractere especial chamado "nova linha" (*newline*)

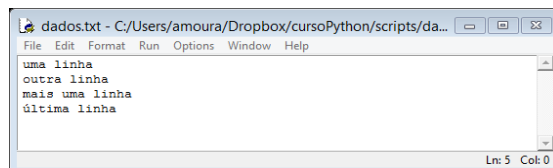


```
dados.txt - C:/Users/amoura/Dropbox/cursoPython/scripts/da...
File Edit Format Run Options Window Help
uma linha
outra linha
mais uma linha
última linha
Ln: 5 Col: 0
```

Arquivos Texto e Strings

São arquivos que armazenam texto (caracteres).

- ✓ um arquivo texto é composto por uma sequência de linhas
- ✓ cada linha é uma sequência de caracteres que termina com um caractere especial chamado "nova linha" (*newline*)



```
dados.txt - C:/Users/amoura/Dropbox/cursoPython/scripts/da...
File Edit Format Run Options Window Help
uma linha
outra linha
mais uma linha
última linha
Ln: 5 Col: 0
```

- ✓ Como uma linha é uma sequência de caracteres, podemos ler seu conteúdo para uma string.



Lendo um Arquivo Texto

Podemos ler todo o conteúdo do arquivo, de uma vez, para uma string:

```
arq = open("dados.txt", "r")
texto = arq.read()
print(texto)
arq.close()
```

o método `read` retorna todo o conteúdo do arquivo
como uma única string



Lendo Linha a Linha: repetição

Podemos ler uma cada linha de cada vez, usando um laço `while`:

```
arq = open("dados.txt", "r")
linha = arq.readline()
while linha:
    print(linha)
    linha = arq.readline()
arq.close()
```

o método `readline` retorna uma string com
o conteúdo da próxima linha do arquivo

Linha a Linha: controlar final

Podemos ler uma cada linha de cada vez, usando um laço while:

```
arq = open("dados.txt", "r")
linha = arq.readline()
while linha:
    print(linha)
    linha = arq.readline()
arq.close()
```

no final do arquivo,
`readline` retorna a
string vazia ("")
e o laço `while` termina

o método `readline` retorna uma string com
o conteúdo da próxima linha do arquivo

Iterando sobre um Arquivo

Podemos usar um laço `for` para iterar sobre as linhas do arquivo:

```
arq = open("dados.txt", "r")
for linha in arq:
    print(linha)
arq.close()
```

em cada passo da iteração, a variável `linha` recebe uma
string com o conteúdo da próxima linha do arquivo



Eliminando “nova linha”

As linhas terminam com o caractere “nova linha”.

```
arq = open("dados.txt", "r")
for linha in arq:
    print(linha)
arq.close()
```

```
>>> uma linha
      outra linha
      mais uma linha
      última linha
>>>
```



Eliminando “nova linha” com strip

As linhas terminam com o caractere “nova linha”.

```
arq = open("dados.txt", "r")
for linha in arq:
    print(linha)
arq.close()
```

```
>>> uma linha
      outra linha
      mais uma linha
      última linha
>>>
```

Podemos eliminar esse caractere com o método **strip**

```
arq = open("dados.txt", "r")
for linha in arq:
    linha = linha.strip()
    print(linha)
arq.close()
```

```
>>> uma linha
      outra linha
      mais uma linha
      última linha
>>>
```



Escrevendo em um Arquivo Texto

Para escrever em um arquivo texto, usamos o método **write**:

```
arq = open("dados.txt", "w")
arq.write("uma string")
arq.write("outra string")
arq.write("\n")
...
arq.close()
```

"w" indica que o arquivo será escrito (write)

escrevemos uma string com o caractere "nova linha" quando queremos terminar uma linha do arquivo



Escrevendo Texto

O argumento de método **write** deve ser uma string!

```
arq = open("dados.txt", "w")
x = 42
arq.write(x)
```

TypeError: write() argument must be str, not int

Outros valores devem ser convertidos para strings

```
arq.write(str(x))
```

Uma alternativa é usar o operador de formato **%**

```
arq.write("%d" % x)
```

Arquivos: mãos na massa!

Experimente criar um arquivo “dados.txt” com o editor do IDLE e execute o script abaixo:

```
arq = open("dados.txt", "r")
texto = arq.read()
print(texto)
arq.close()
```

Trabalhando linha a linha

Mude agora o seu script para que ele leia o arquivo linha a linha, exibindo cada linha lida.

- ✓ você pode usar um laço while ou um for para isso

Você reparou que há duas “quebras de linha” para cada linha lida? Por que? (uma delas é exibida pelo print...)

- ✓ mude o seu script para que haja apenas uma quebra de linha ao exibir as linhas do arquivo.

Exercício: alunos e notas

Com o editor do IDLE, crie um arquivo onde cada linha tem o nome do aluno e suas duas notas, como no exemplo a seguir: (**mas crie mais alunos!**)

```
Ana Carolina Medeiros,10.0,8.0  
Manoel de Souza Filho,5.0,4.0  
Zuenir Ventura,2.5,9.0
```

Escreva uma função **lista_aprovados** que receba o nome do arquivo que você criou e imprima o nome dos alunos aprovados.

- ✓ um aluno está aprovado se a media das suas duas notas é maior ou igual a 5.0

Dicas para “alunos e notas”

- ✓ Você pode procurar as posições das vírgulas e usar “fatiamento” para obter as substrings com o nome e as notas.
- ✓ Para poder calcular a média das notas, você precisa converter as substrings com as notas para valores do tipo float:

```
nota = "8.5"
```

```
type(nota)
```

```
type(float(nota))
```



```
<type 'str'>
```

```
<type 'float'>
```

Uma solução para alunos e notas

```
def lista_aprovados(nome):  
    arq = open(nome, "r")  
    for linha in arq:  
        p1 = linha.find(",") # virgula depois do nome  
        nome = linha[:p1]  
  
        p2 = linha.find(",", p1+1) # virgula entre as notas  
        n1 = float(linha[p1+1:p2])  
        n2 = float(linha[p2+1:])  
  
        if (n1+n2)/2 >= 5.0:  
            print(nome)  
  
    arq.close()  
    return
```

Exercício: arquivo HTML

Veja abaixo o conteúdo de um arquivo html **muito** simples que exibe uma tabela de nomes.

- ✓ um arquivo html é um arquivo texto!

```
<html><head><title>Alunos</title></head>  
<body><table border="1">  
  <tr><th>Nome</th></tr>  
  <tr><td>José da Silva</td></tr>  
  <tr><td>Zuenir Ventura</td></tr>  
</table></body></html>
```

Escreva uma função **cria_tabela** que receba o nome do arquivo que você usou no exercício anterior e crie um novo arquivo, chamado "tabela.html", que exiba uma tabela com os nomes dos alunos.

Dicas para o exercício HTML

Sua função deve abrir dois arquivos (e fechá-los no final):

- ✓ o arquivo de entrada, para leitura
- ✓ o arquivo de saída ("tabela.html"), para escrita

O conteúdo inicial deste novo arquivo é fixo, portanto basta escrever as linhas com esse conteúdo.

Repare que para cada nome no arquivo de entrada, você deverá escrever uma linha no arquivo de saída como

```
<tr><td>NOME</td></tr>
```

onde **NOME**, a cada linha, tem o nome de um aluno. Você pode, então, usar uma string com esse "modelo" para servir de base para a linha a ser escrita.

Uma solução para HTML

```
def cria_tabela(nome):
    arq_ent = open(nome, "r")
    arq_nomes = open("tabela.html", "w")

    arq_nomes.write('<html><head><title>Alunos</title></head>' + '\n')
    arq_nomes.write('<body><table border="1">' + '\n')
    arq_nomes.write('<tr><th>Nome</th></tr>' + '\n')

    modelo = '<tr><td>NOME</td></tr>' + '\n'

    for linha in arq_ent:
        nome = linha[:linha.find(",")]
        arq_nomes.write(modelo.replace("NOME", nome))

    arq_nomes.write('</table></body></html>')

    arq_ent.close()
    arq_nomes.close()
    return
```

Exercício: desenho misterioso

Copie para seu diretório de trabalho o arquivo chamado “misterio.txt” que está em <link>

As linhas desse arquivo contém ou a palavra CIMA, ou a palavra BAIXO, ou um par de números, como no exemplo:

```
CIMA
-218 185
BAIXO
-240 189
-246 188
CIMA
-218 185
```

Desenho misterioso: turtle

```
CIMA
-218 185
BAIXO
-240 189
-246 188
CIMA
-218 185
```

CIMA e **BAIXO** são instruções para a tartaruga colocar sua cauda, respectivamente, para **cima** (up) ou para **baixo** (down).

O par de números são coordenadas x e y, para onde a tartaruga deve caminhar (usando “goto”).

Escreva um programa que leia o arquivo dado e use a tartaruga para desenhar a figura descrita por suas instruções e seu conjunto de pontos.



Solução para desenho misterioso

```
import turtle
pat = turtle.Turtle()
arq = open("misterio.dat", "r")
for linha in arq:
    linha = linha.strip() # para tirar o newline
    if linha == "CIMA":
        pat.up()
    elif linha == "BAIXO":
        pat.down()
    else:
        posicao = linha.find(" ")
        x = int(linha[:posicao])
        y = int(linha[posicao+1:])
        pat.goto(x,y)
pat.hideturtle()
```



Voilà: Desenho Misterioso!

