

FUNÇÕES DE MANIPULAÇÃO DE ARQUIVOS

Link para download de projeto básico de manipulação de arquivos [link](#).

Operações básicas

Conceitos

Abrindo um arquivo

Veja as operações básicas de manipulação de arquivos:

Abrir

Fechar

Ler

Escrever

A primeira operação que precisamos realizar, independentemente se vamos ler o conteúdo de um arquivo ou adicionar um conteúdo, é **abrir o arquivo**.

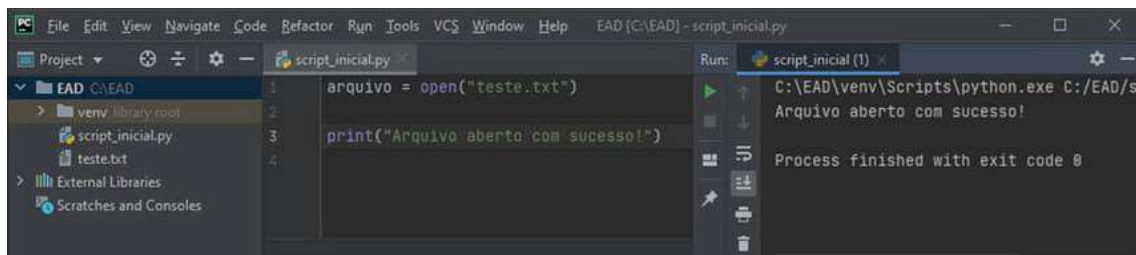
Para abrir um arquivo, o Python disponibiliza a função interna chamada `open`. Essa função está disponível globalmente, ou seja, não é preciso importá-la.

A função `open` retorna um objeto do tipo arquivo. Sua forma mais simples de utilização tem a seguinte sintaxe:

```
arquivo = open (caminho)
```

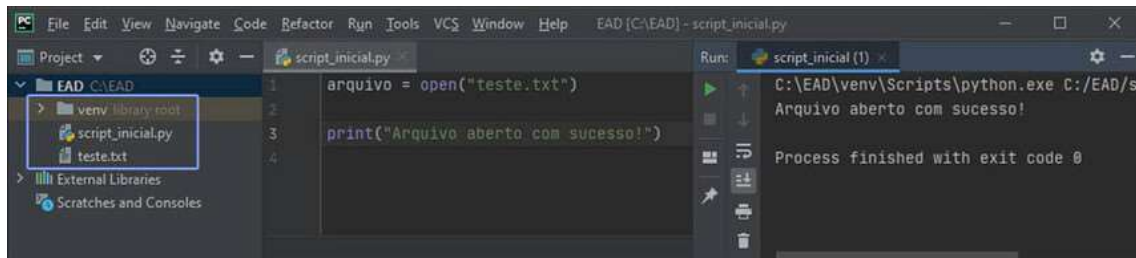
Utilizamos a função `open` com o parâmetro `caminho`. Esse parâmetro é uma string que representa a localização do arquivo no sistema de arquivos.

Veja como é fácil abrir um arquivo:



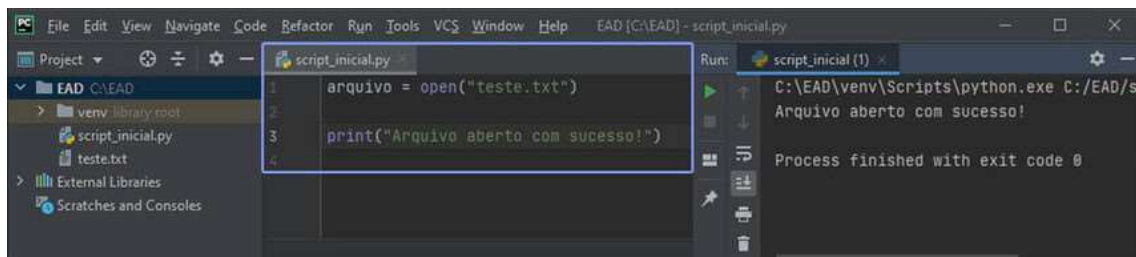
Abertura do arquivo.

Temos, inicialmente, o script inicial e sua saída.



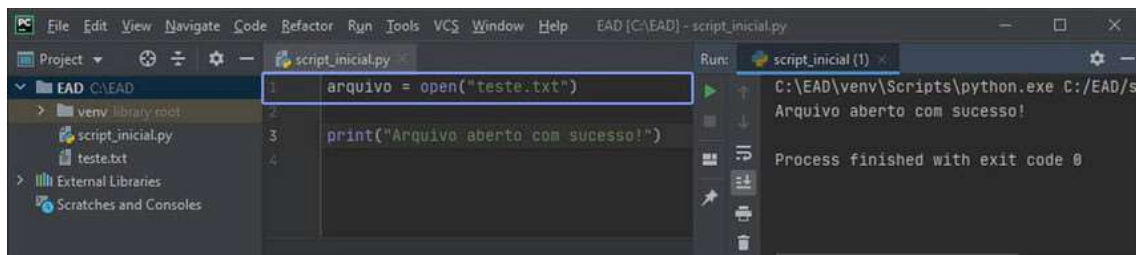
Arquivos `script_inicial.py` e `teste.txt`.

Temos, à esquerda da imagem, a árvore de diretório, onde verificamos a existência dos arquivos `script_inicial.py` e `teste.txt`, ambos no mesmo diretório EAD.



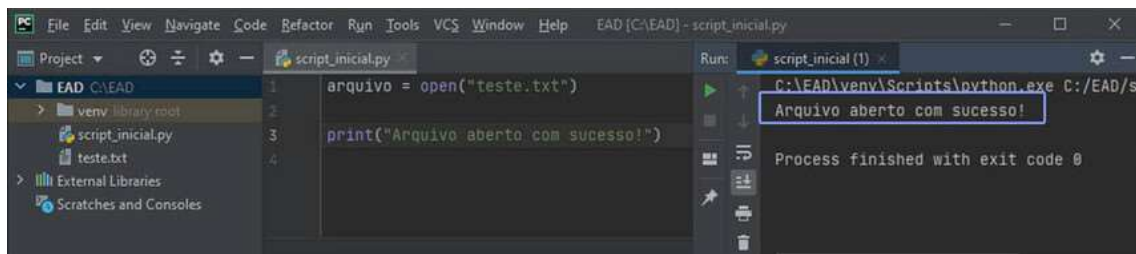
Código do script e a saída de controle.

Temos, ao centro, o código do nosso script e, à direita, a saída do console.



A função `open` para a abertura do arquivo `teste.txt`.

Utilizamos, na linha 1 do script, a função `open` para abrir o arquivo `teste.txt`. Isso já é suficiente para termos um objeto do tipo arquivo e começar a manipulá-lo.



Impressão da frase "Arquivo aberto com sucesso!".

Imprimimos, na linha 3, a frase "Arquivo aberto com sucesso!" apenas para verificar se o programa foi executado sem problemas.

Nesse exemplo, o caminho utilizado para abrir o arquivo foi “teste.txt”, pois o script e o arquivo que abrimos estão no mesmo diretório. Porém, não precisamos nos limitar a manter os arquivos e scripts no mesmo diretório.

Veja como o Python trata o acesso aos arquivos a seguir. O caminho de um arquivo pode ser classificado em dois tipos:

Absoluto

É a referência completa para se encontrar um arquivo ou diretório. Ele deve começar com uma barra (/) ou o rótulo do drive (C:, D: ...).

Exemplo:

`open("C:\Downloads\arquivo.txt")` – utilizado em ambientes MS Windows.
`open("/home/usuario/arquivo.txt")` – utilizado em ambientes Linux.

Relativo

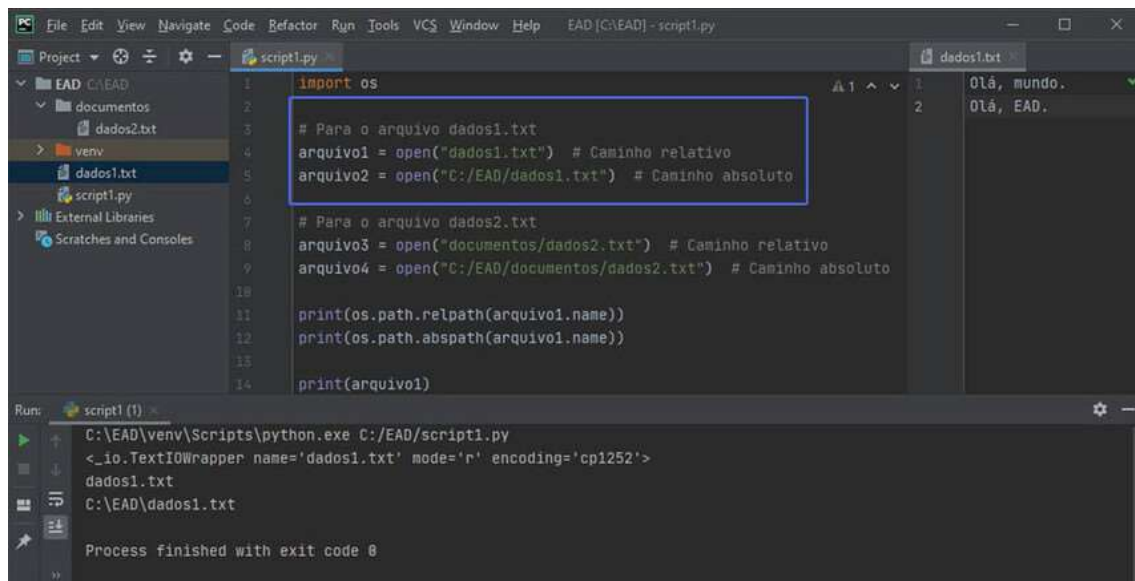
É a referência para se encontrar um arquivo ou diretório a partir de outro diretório. Normalmente, a partir do diretório de onde o script está.

Exemplo:

`open("arquivo.txt")`, para os casos em que o arquivo está no mesmo diretório do **script**.
`open("../arquivo.txt")`, para os casos em que o arquivo está no diretório acima do **script**.

Vamos criar um script que ilustra as diferentes formas de referenciar um arquivo com caminhos absolutos e relativos.

No exemplo, a seguir, alteramos um pouco a forma de exibir o conteúdo:



The screenshot shows an IDE window titled "EAD [C:\EAD] - script1.py". The left sidebar displays a project tree with the following structure:

- EAD C:\EAD
 - documentos
 - dados2.txt
 - venv
 - dados1.txt
 - script1.py
 - External Libraries
 - Scratches and Consoles

The main editor displays the following Python code in `script1.py`:

```
1 import os
2
3 # Para o arquivo dados1.txt
4 arquivo1 = open("dados1.txt") # Caminho relativo
5 arquivo2 = open("C:/EAD/dados1.txt") # Caminho absoluto
6
7 # Para o arquivo dados2.txt
8 arquivo3 = open("documentos/dados2.txt") # Caminho relativo
9 arquivo4 = open("C:/EAD/documentos/dados2.txt") # Caminho absoluto
10
11 print(os.path.relpath(arquivo1.name))
12 print(os.path.abspath(arquivo1.name))
13
14 print(arquivo1)
```

The right sidebar shows the content of `dados1.txt`:

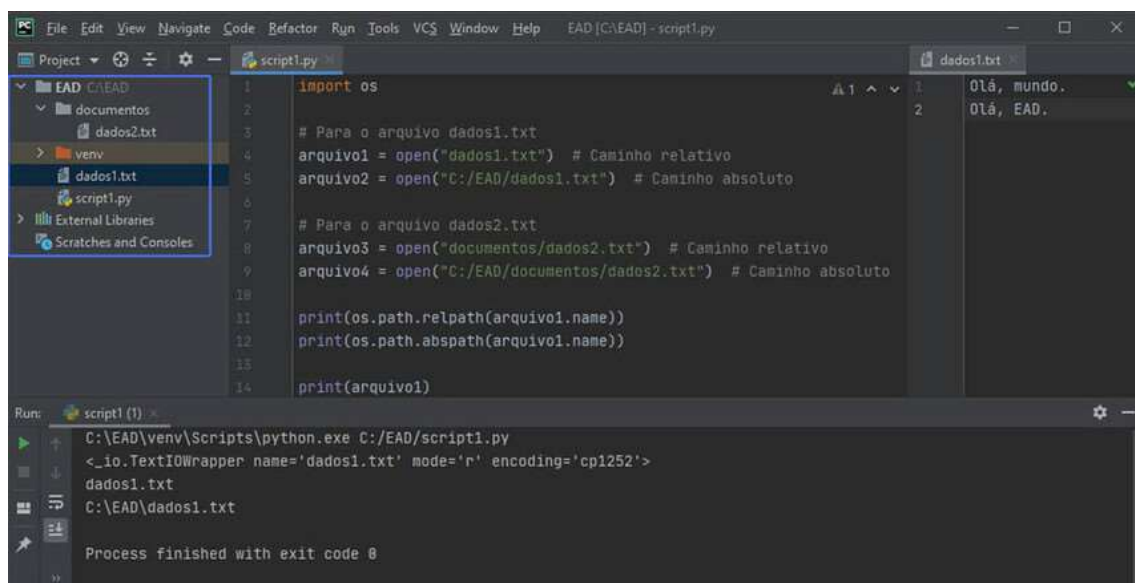
```
1 Olá, mundo.
2 Olá, EAD.
```

The bottom "Runs" panel shows the execution output for `script1 (1)`:

```
C:\EAD\venv\Scripts\python.exe C:/EAD/script1.py
<_io.TextIOWrapper name='dados1.txt' mode='r' encoding='cp1252'>
dados1.txt
C:\EAD\dados1.txt
Process finished with exit code 0
```

Script 1, sua saída e o arquivo dados1.txt.

Temos, inicialmente, o **script1**, sua saída e arquivo dados1.txt.



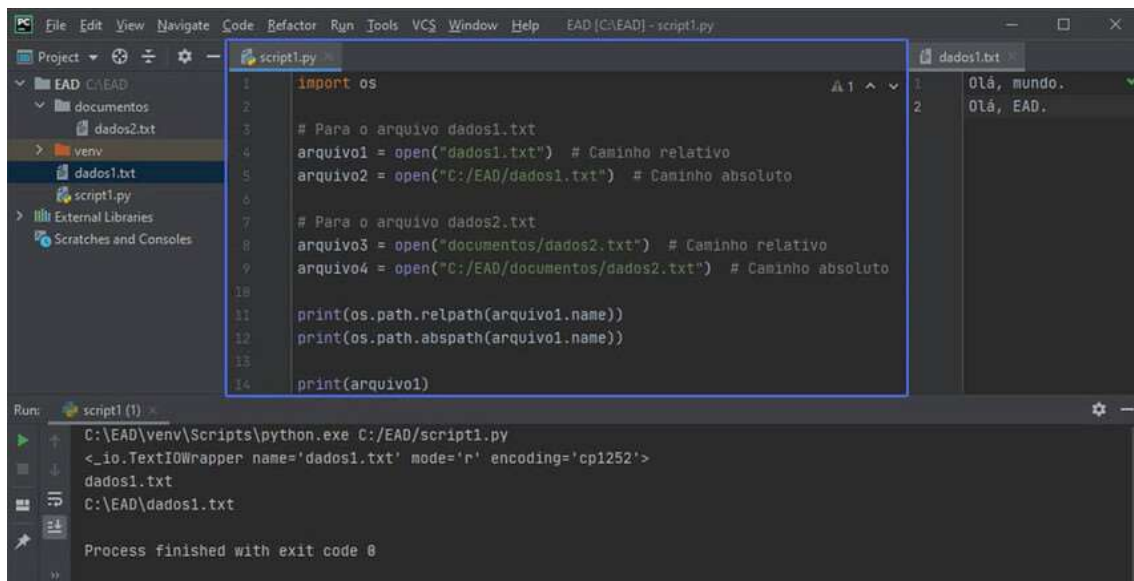
This screenshot is similar to the first one, but with a blue box highlighting the project tree in the left sidebar:

- EAD C:\EAD
 - documentos
 - dados2.txt
 - venv
 - dados1.txt
 - script1.py
 - External Libraries
 - Scratches and Consoles

The main editor and the "Runs" panel show the same content as the first screenshot.

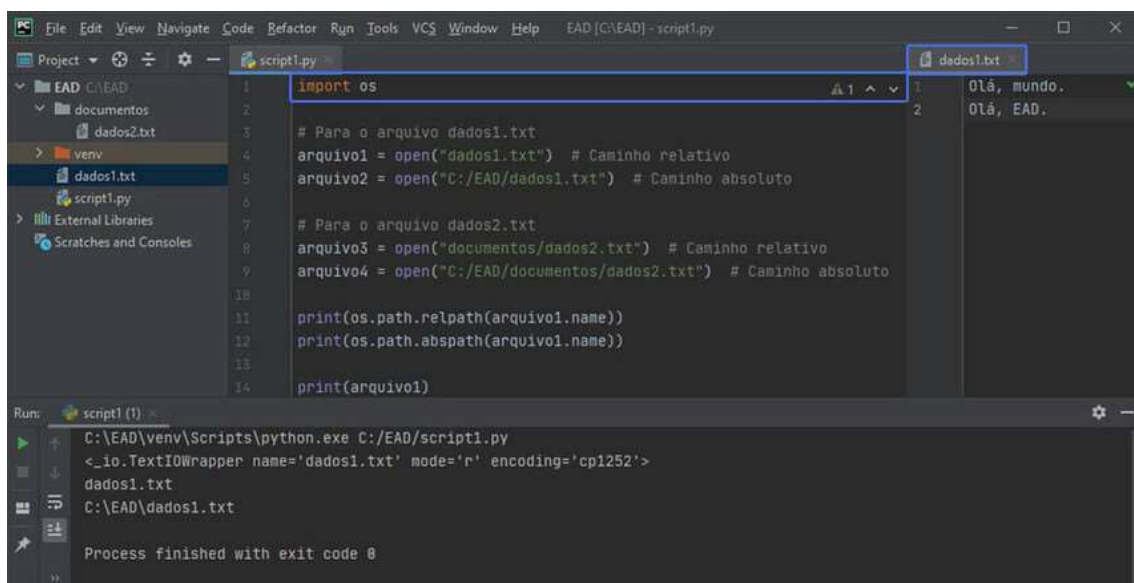
Árvore de diretórios.

Temos, à **esquerda**, nossa árvore de diretórios.



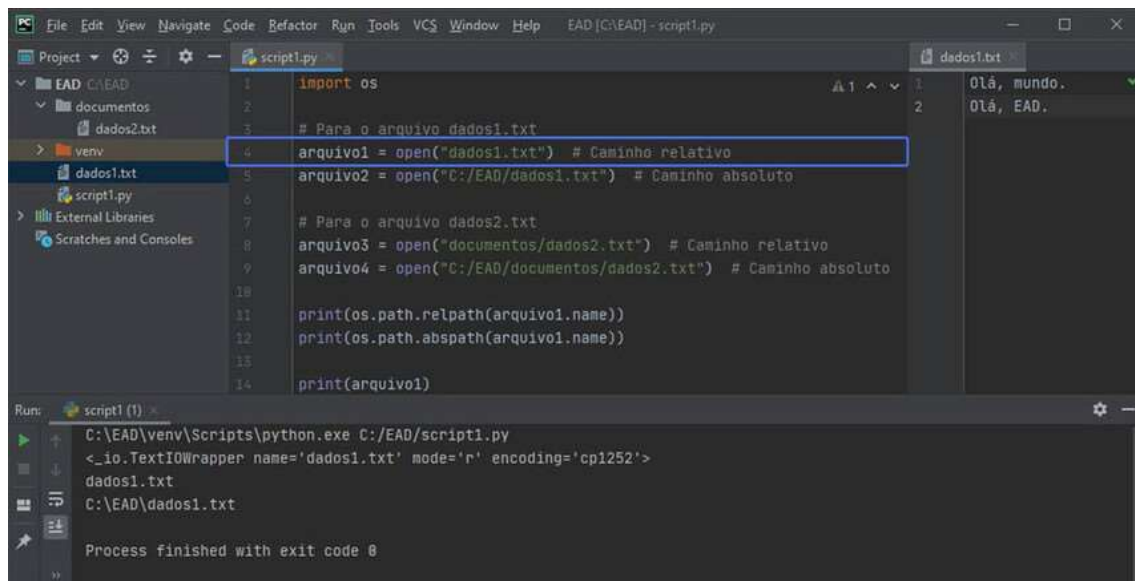
O script1.py.

Temos, **ao centro**, o script1.py.



O arquivo dados.txt e saída do console do Python.

Temos, **à direita**, o arquivo dados.txt, e, abaixo, a saída do console do Python. Na linha 1 do script1.py, importamos o módulo os do Python.



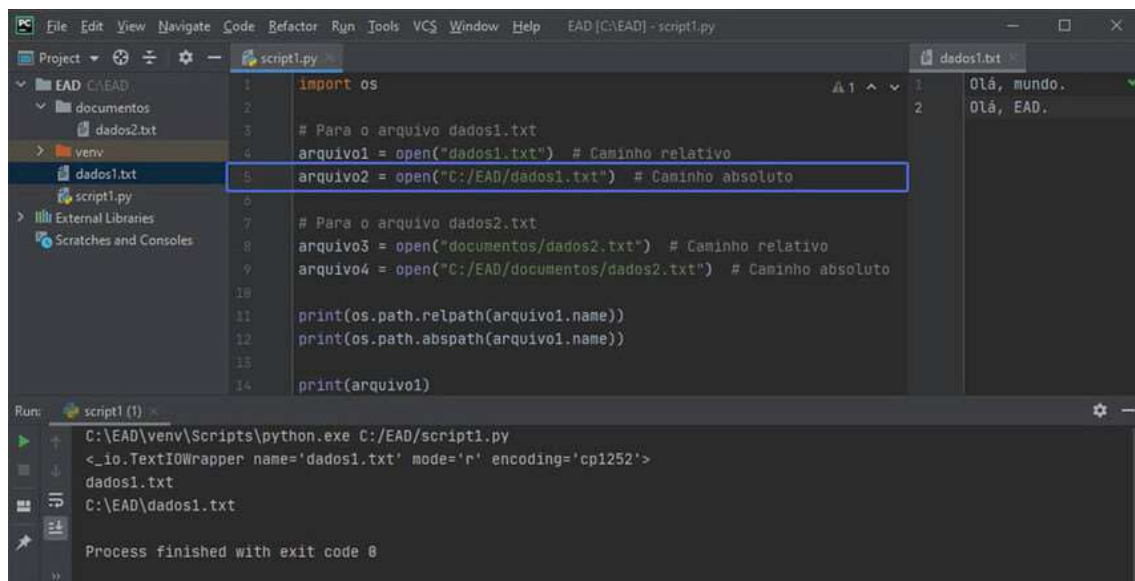
```
1 import os
2
3 # Para o arquivo dados1.txt
4 arquivo1 = open("dados1.txt") # Caminho relativo
5 arquivo2 = open("C:/EAD/dados1.txt") # Caminho absoluto
6
7 # Para o arquivo dados2.txt
8 arquivo3 = open("documentos/dados2.txt") # Caminho relativo
9 arquivo4 = open("C:/EAD/documentos/dados2.txt") # Caminho absoluto
10
11 print(os.path.relpath(arquivo1.name))
12 print(os.path.abspath(arquivo1.name))
13
14 print(arquivo1)
```

Runs: script1 (1) x

```
C:\EAD\venv\Scripts\python.exe C:/EAD/script1.py
<_io.TextIOWrapper name='dados1.txt' mode='r' encoding='cp1252'>
dados1.txt
C:\EAD\dados1.txt
Process finished with exit code 0
```

Abertura do arquivo “dados1.txt” pelo caminho relativo.

Utilizamos, na **linha 4**, a função `open` para abrir o arquivo “**dados1.txt**”, que se encontra no mesmo diretório do nosso script. Nessa linha, utilizamos o caminho relativo. Observe que, como o arquivo `dados1.txt` está na mesma pasta `EAD` que o `script1.py`, basta escrever o nome do arquivo como argumento.



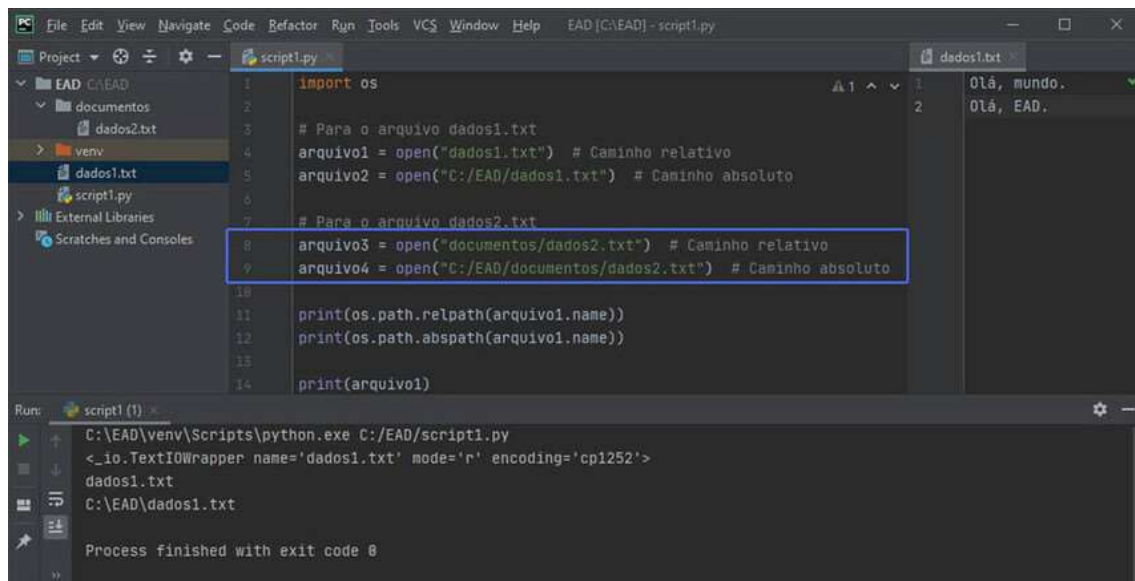
```
1 import os
2
3 # Para o arquivo dados1.txt
4 arquivo1 = open("dados1.txt") # Caminho relativo
5 arquivo2 = open("C:/EAD/dados1.txt") # Caminho absoluto
6
7 # Para o arquivo dados2.txt
8 arquivo3 = open("documentos/dados2.txt") # Caminho relativo
9 arquivo4 = open("C:/EAD/documentos/dados2.txt") # Caminho absoluto
10
11 print(os.path.relpath(arquivo1.name))
12 print(os.path.abspath(arquivo1.name))
13
14 print(arquivo1)
```

Runs: script1 (1) x

```
C:\EAD\venv\Scripts\python.exe C:/EAD/script1.py
<_io.TextIOWrapper name='dados1.txt' mode='r' encoding='cp1252'>
dados1.txt
C:\EAD\dados1.txt
Process finished with exit code 0
```

Abertura do arquivo “dados1.txt” pelo caminho absoluto.

Abrimos, na **linha 5**, o mesmo arquivo `dados1.txt`, utilizando o caminho absoluto (completo), que, no nosso exemplo, é: `"C:/EAD/dados1.txt"`.



Abertura do arquivo dados2.txt usando os dois caminhos.

Abrimos, **nas linhas 8 e 9**, o arquivo dados2.txt, que se encontra na pasta documentos. Na linha 8, utilizamos o caminho relativo desse arquivo para abri-lo: "documentos/dados2.txt", enquanto, na linha 9, utilizamos o caminho absoluto: "C:/EAD/documentos/dados2.txt".

O Python também disponibiliza algumas funções para exibir os caminhos absolutos e relativos de um arquivo ou diretório, que são:

- Na linha 11, utilizamos a função **path.relpath** para imprimir o caminho relativo do arquivo1, a partir do nome do arquivo passado como parâmetro.
- Na linha 12, utilizamos a função **path.abspath** para exibir o caminho absoluto do mesmo arquivo. Observe que, mesmo utilizando o caminho relativo para abrir o arquivo (linha 4), é possível obter o caminho absoluto utilizando essa função. Isso pode ser verificado na saída do console.
- Na linha 14, utilizamos a função interna **print** para imprimir a variável arquivo1.

Verifique, na saída do console, onde foi impressa a representação do objeto arquivo1:

```
<_io.TextIOWrapper name='dados1.txt' mode='r' encoding='cp1252'>
```

Desmembrando essa saída, temos:

- O tipo do objeto, TextIOWrapper, que trata de arquivos de texto.
- O nome do arquivo, name='dados.txt'.
- O modo de acesso ao arquivo, mode='r'.
- A codificação do arquivo, encoding='cp1252'.

Neste módulo, vamos tratar apenas de arquivos do tipo texto, ou seja, objetos `TextIOWrapper`. A seguir, vamos apresentar os diferentes modos de acesso aos arquivos.

Modos de acesso a um arquivo

Quando abrimos um arquivo, precisamos informar ao Python o que desejamos fazer, ou seja, qual será o modo (*mode*) de acesso ao arquivo. O modo é um dos parâmetros da função *open*, e cada modo é representado por uma string.

Os principais modos são:

r: leitura (read)

w: escrita (write)

a: acrescentar (append)

O modo padrão da função *open* é o modo leitura (“r”).

Esses modos podem ser combinados e para informar que desejamos ler e escrever em um arquivo, utilizamos a string “r+”, por exemplo.

O Python também nos permite diferenciar arquivos texto de arquivos binários, como uma imagem, por exemplo. Para informar que desejamos abrir um arquivo binário, adicionamos a string “b” ao modo, ficando “rb”, “wb” e “ab”.

A tabela abaixo resume os modos de acesso a arquivos:

Caractere	Significado
'r'	Abre o arquivo para leitura (default).
'w'	Abre o arquivo para escrita, truncando o arquivo primeiro.
'x'	Cria um arquivo para escrita e falha, caso ele exista.
'a'	Abre o arquivo para escrita, acrescentando conteúdo ao final do arquivo, caso ele exista.
'b'	Modo binário.
't'	Modo texto (default).

Caractere	Significado
'+'	Abre o arquivo para atualização (leitura ou escrita).

Tabela: Modos de abertura de arquivos em Python.
Adaptado de Python (2020).

Atributos do objeto tipo arquivo

Atributos de um arquivo

O objeto do tipo arquivo contém alguns atributos importantes, como name, mode e closed, veja:

```

1 arquivo = open("dados.txt")
2
3 print("Nome do arquivo:", arquivo.name)
4 print("Modo do arquivo:", arquivo.mode)
5 print("Arquivo fechado?", arquivo.closed)
6
7 arquivo.close()
8
9 print("Arquivo fechado?", arquivo.closed)
10

```

```

Run: script3
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe C:/Users/ftoli/PycharmProjects/EAD/script3.py
Nome do arquivo: dados.txt
Modo do arquivo: r
Arquivo fechado? False
Arquivo fechado? True

Process finished with exit code 0

```

Script2, sua saída e arquivo dados.txt.

Temos o script2, sua saída e arquivo dados.txt.

The screenshot shows the PyCharm IDE with a project named 'EAD'. The file explorer on the left shows a directory structure with 'dados.txt', 'script1.py', and 'script2.py'. The main editor window displays 'script2.py' with the following code:

```
1 arquivo = open("dados.txt")
2
3 print("Nome do arquivo:", arquivo.name)
4 print("Modo do arquivo:", arquivo.mode)
5 print("Arquivo fechado?", arquivo.closed)
6
```

The Run console at the bottom shows the output of the script:

```
Run: script2
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe C:/Users/ftoli/PycharmProjects/EAD/script2.py
Nome do arquivo: dados.txt
Modo do arquivo: r
Arquivo fechado? False
Process finished with exit code 0
```

Abertura do arquivo pela função *open*

Abrimos, **na linha 1**, o arquivo utilizando a função *open*. Como não explicitamos o parâmetro *mode*, o arquivo será aberto no modo leitura ("r").

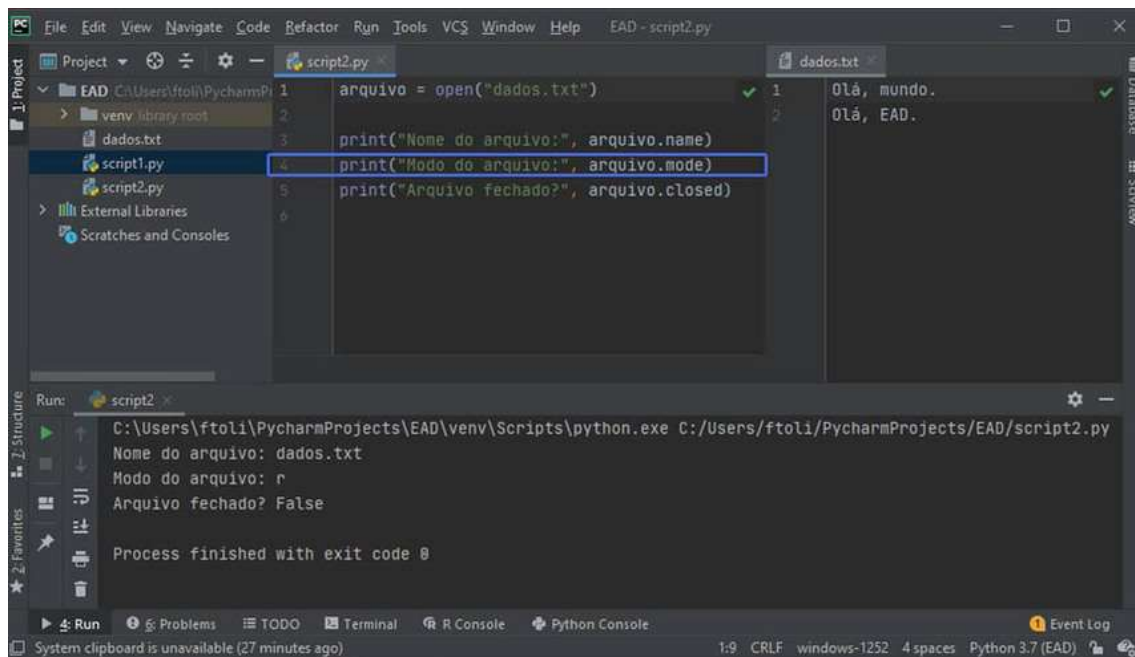
This screenshot is similar to the first one, but with a blue box highlighting line 3 of the code in 'script2.py':

```
1 arquivo = open("dados.txt")
2
3 print("Nome do arquivo:", arquivo.name)
4 print("Modo do arquivo:", arquivo.mode)
5 print("Arquivo fechado?", arquivo.closed)
6
```

The Run console output is identical to the first screenshot, showing the file name 'dados.txt' printed.

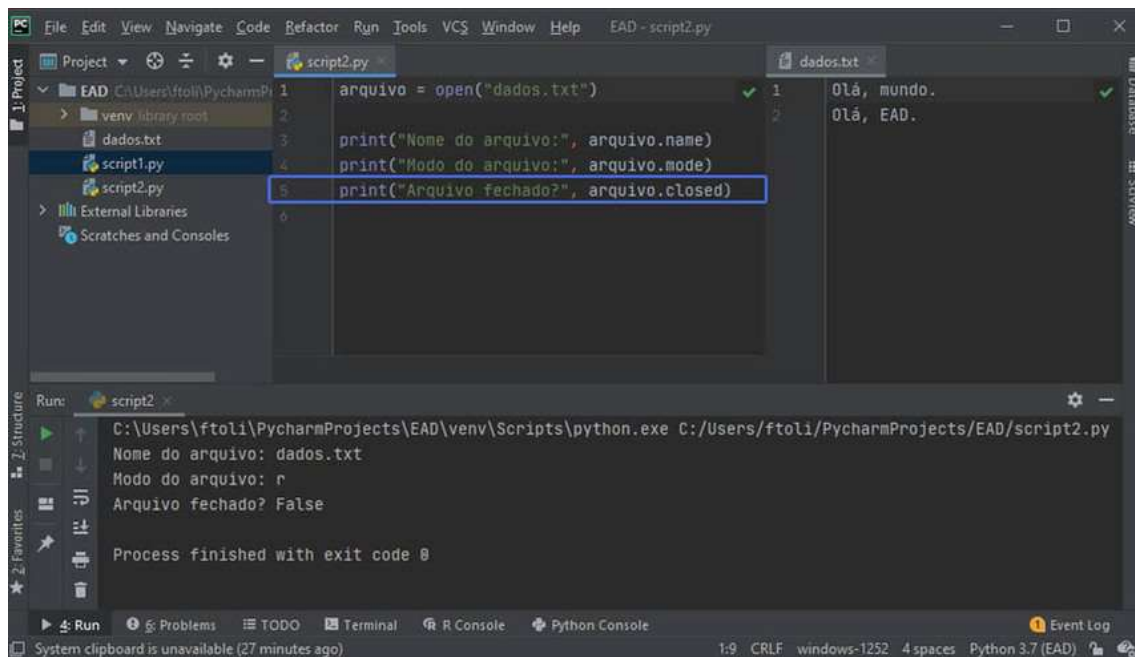
Impressão do atributo *name*.

Imprimimos, **na linha 3**, o atributo *name* do objeto arquivo. Esse atributo contém o nome do arquivo.



Impressão do atributo *name*.

Imprimimos, **na linha 4**, o atributo *mode* do objeto arquivo. Esse atributo contém o modo de acesso do arquivo (r, w, a, rb ...).



Impressão do atributo *closed*.

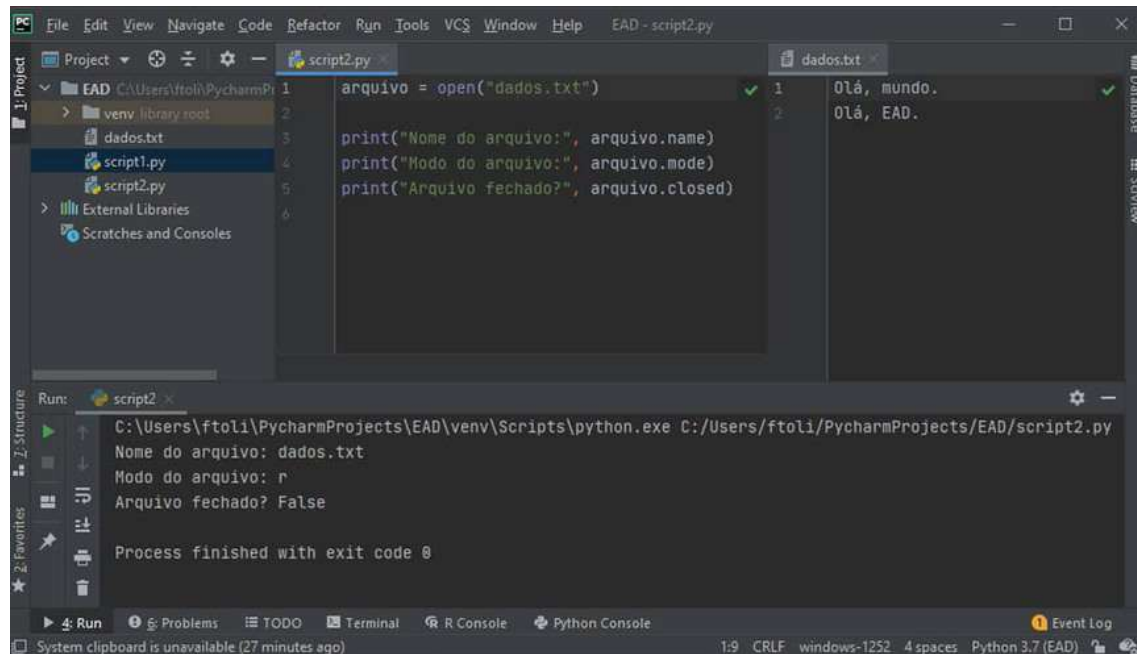
Imprimimos, **na linha 5**, o atributo *closed* do objeto arquivo. Essa atributo serve para verificar se um arquivo está ou não fechado.

Os valores de cada atributo podem ser verificados no console abaixo da imagem.

Fechando um arquivo

Após realizar a operação desejada no arquivo, precisamos liberá-lo. Para isso, utilizamos o método **close()**, que libera a memória alocada pelo interpretador e o uso do arquivo por outros programas, por exemplo.

Agora, vamos utilizar o script do exemplo anterior como base e adicionar uma chamada ao método **close()** e verificar o atributo `closed` novamente:



Script3, sua saída e seu arquivo dados.txt.

Em relação ao script do exemplo anterior, adicionamos, na linha 7, uma chamada ao método `close()` do objeto `arquivo`.

Na linha 9, imprimimos novamente a propriedade `closed`, onde podemos observar que seu valor agora é `True`.

Lendo o conteúdo de um arquivo

Agora que já sabemos abrir e fechar um arquivo, vamos ver as formas de ler seu conteúdo.

O Python disponibiliza os seguintes métodos para leitura do conteúdo de um arquivo-texto:

Read()

Retorna **todo** o conteúdo de um arquivo como uma única string.

Readline()

Retorna uma **linha** de arquivo, incluindo caracteres de final (\n ou \r\n), e avança o cursor para a próxima.

Readlines()

Retorna uma **lista** em que cada item da lista é uma linha do arquivo.

Abaixo, temos três scripts, em que cada um utiliza um dos métodos descritos anteriormente para leitura do arquivo. Observe que explicitamos o modo de operação como leitura ("r"):

```
script4.py
1 arquivo = open("dados.txt", "r")
2 conteudo = arquivo.read()
3 print("Tipo do conteúdo:", type(conteudo))
4 print("Conteúdo retornado pelo read:")
5 print(repr(conteudo))
6
7 arquivo.close()
8
9
10
11

script5.py
1 arquivo = open("dados.txt", "r")
2 conteudo = arquivo.readline()
3 print("Tipo do conteúdo:", type(conteudo))
4 print("Conteúdo retornado pelo readline:")
5 print(repr(conteudo))
6
7 proximo_conteudo = arquivo.readline()
8 print("Próximo Conteúdo retornado:")
9 print(repr(proximo_conteudo))
10
11 arquivo.close()
12
13
14
15
16

script6.py
1 arquivo = open("dados.txt", "r")
2 conteudo = arquivo.readlines()
3 print("Tipo do conteúdo:", type(conteudo))
4 print("Conteúdo retornado pelo readlines:")
5 print(repr(conteudo))
6
7 arquivo.close()
8
9
10
11
```

Run script4: C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts: Tipo do conteúdo: <class 'str'> Conteúdo retornado pelo read: 'Olá, mundo.\nOlá, EAD.'

Run script5: C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts: Tipo do conteúdo: <class 'str'> Conteúdo retornado pelo readline: 'Olá, mundo.\n' Próximo Conteúdo retornado: 'Olá, EAD.'

Run script6: C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts: Tipo do conteúdo: <class 'list'> Conteúdo retornado pelo readlines: ['Olá, mundo.\n', 'Olá, EAD.']

Scripts 4, 5 e 6 e cada uma de suas saídas.

Temos os **scripts 4, 5 e 6** e suas respectivas saídas.

```
script4.py
1 arquivo = open("dados.txt", "r")
2 conteudo = arquivo.read()
3 print("Tipo do conteúdo:", type(conteudo))
4 print("Conteúdo retornado pelo read:")
5 print(repr(conteudo))
6
7 arquivo.close()
8
9
10
11

script5.py
1 arquivo = open("dados.txt", "r")
2 conteudo = arquivo.readline()
3 print("Tipo do conteúdo:", type(conteudo))
4 print("Conteúdo retornado pelo readline:")
5 print(repr(conteudo))
6
7 proximo_conteudo = arquivo.readline()
8 print("Próximo Conteúdo retornado:")
9 print(repr(proximo_conteudo))
10
11 arquivo.close()
12
13
14
15
16

script6.py
1 arquivo = open("dados.txt", "r")
2 conteudo = arquivo.readlines()
3 print("Tipo do conteúdo:", type(conteudo))
4 print("Conteúdo retornado pelo readlines:")
5 print(repr(conteudo))
6
7 arquivo.close()
8
9
10
11
```

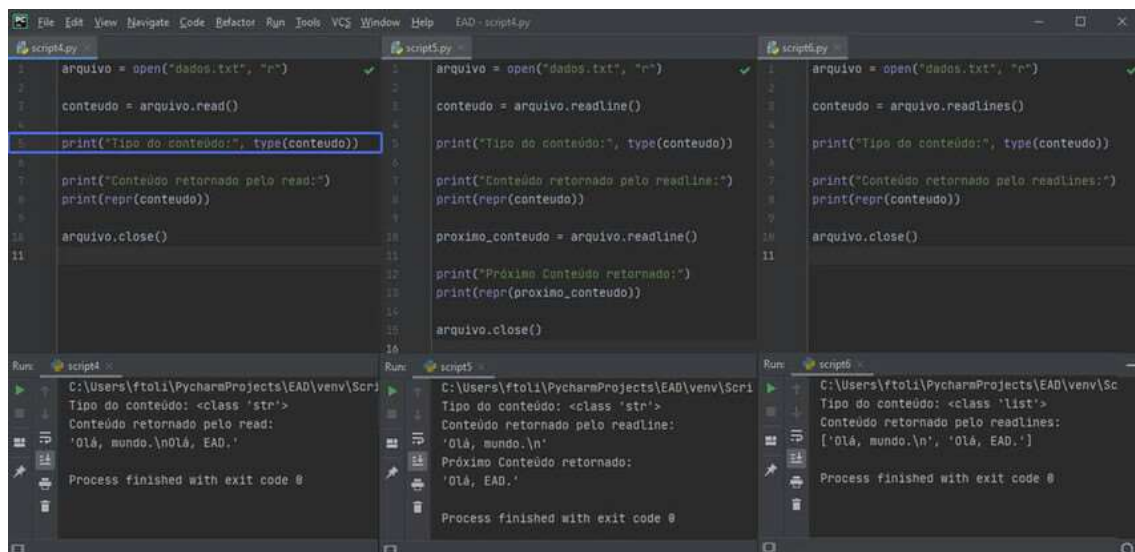
Run script4: C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts: Tipo do conteúdo: <class 'str'> Conteúdo retornado pelo read: 'Olá, mundo.\nOlá, EAD.'

Run script5: C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts: Tipo do conteúdo: <class 'str'> Conteúdo retornado pelo readline: 'Olá, mundo.\n' Próximo Conteúdo retornado: 'Olá, EAD.'

Run script6: C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts: Tipo do conteúdo: <class 'list'> Conteúdo retornado pelo readlines: ['Olá, mundo.\n', 'Olá, EAD.']

Abertura do arquivo pelo método read().

Abrimos, **no script4.py**, mais à esquerda, abrimos o arquivo na linha 1 e, na linha 3, utilizamos o método **read()** do objeto **arquivo** para ler o conteúdo de dados.txt e armazená-lo na variável **conteudo**.



```
script4.py
1 arquivo = open("dados.txt", "r")
2
3 conteudo = arquivo.read()
4
5 print("Tipo do conteúdo:", type(conteudo))
6
7 print("Conteúdo retornado pelo read:")
8 print(repr(conteudo))
9
10 arquivo.close()
11

Run: script4
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts
Tipo do conteúdo: <class 'str'>
Conteúdo retornado pelo read:
'Olá, mundo.\nOlá, EAD.'
Process finished with exit code 0

script5.py
1 arquivo = open("dados.txt", "r")
2
3 conteudo = arquivo.readline()
4
5 print("Tipo do conteúdo:", type(conteudo))
6
7 print("Conteúdo retornado pelo readline:")
8 print(repr(conteudo))
9
10 proximo_conteudo = arquivo.readline()
11
12 print("Próximo Conteúdo retornado:")
13 print(repr(proximo_conteudo))
14
15 arquivo.close()
16

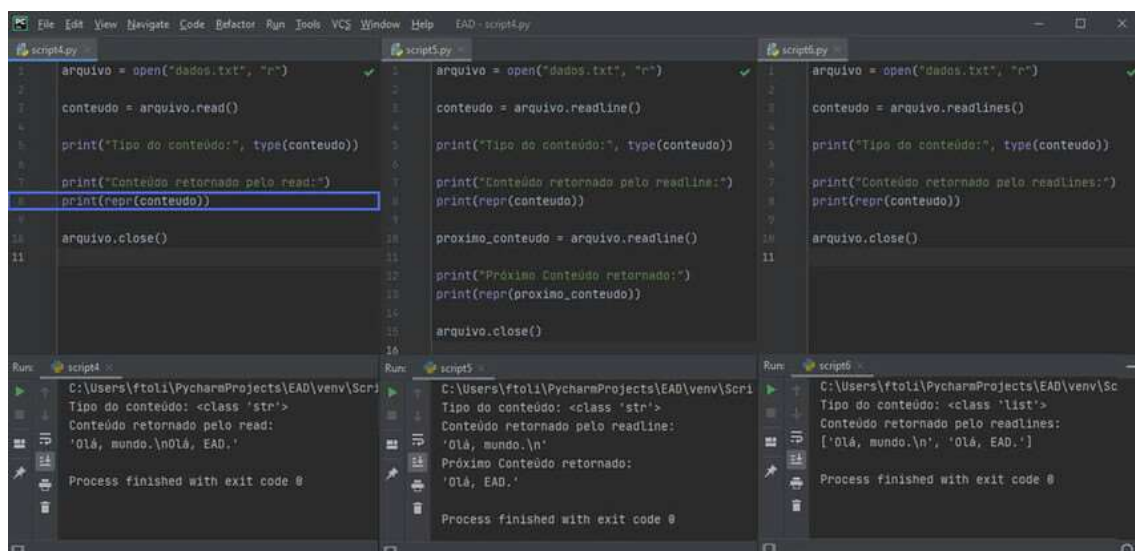
Run: script5
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts
Tipo do conteúdo: <class 'str'>
Conteúdo retornado pelo readline:
'Olá, mundo.\n'
Próximo Conteúdo retornado:
'Olá, EAD.'
Process finished with exit code 0

script6.py
1 arquivo = open("dados.txt", "r")
2
3 conteudo = arquivo.readlines()
4
5 print("Tipo do conteúdo:", type(conteudo))
6
7 print("Conteúdo retornado pelo readlines:")
8 print(repr(conteudo))
9
10 arquivo.close()
11

Run: script6
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts
Tipo do conteúdo: <class 'list'>
Conteúdo retornado pelo readlines:
['Olá, mundo.\n', 'Olá, EAD.']
Process finished with exit code 0
```

Impressão do atributo *name*.

Verificamos, na linha 5, o tipo do conteúdo retornado pelo método **read()**, utilizando a função interna **type**. Conforme exibido no console, a variável **conteudo** é um objeto do tipo **str** (string).



```
script4.py
1 arquivo = open("dados.txt", "r")
2
3 conteudo = arquivo.read()
4
5 print("Tipo do conteúdo:", type(conteudo))
6
7 print("Conteúdo retornado pelo read:")
8 print(repr(conteudo))
9
10 arquivo.close()
11

Run: script4
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts
Tipo do conteúdo: <class 'str'>
Conteúdo retornado pelo read:
'Olá, mundo.\nOlá, EAD.'
Process finished with exit code 0

script5.py
1 arquivo = open("dados.txt", "r")
2
3 conteudo = arquivo.readline()
4
5 print("Tipo do conteúdo:", type(conteudo))
6
7 print("Conteúdo retornado pelo readline:")
8 print(repr(conteudo))
9
10 proximo_conteudo = arquivo.readline()
11
12 print("Próximo Conteúdo retornado:")
13 print(repr(proximo_conteudo))
14
15 arquivo.close()
16

Run: script5
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts
Tipo do conteúdo: <class 'str'>
Conteúdo retornado pelo readline:
'Olá, mundo.\n'
Próximo Conteúdo retornado:
'Olá, EAD.'
Process finished with exit code 0

script6.py
1 arquivo = open("dados.txt", "r")
2
3 conteudo = arquivo.readlines()
4
5 print("Tipo do conteúdo:", type(conteudo))
6
7 print("Conteúdo retornado pelo readlines:")
8 print(repr(conteudo))
9
10 arquivo.close()
11

Run: script6
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts
Tipo do conteúdo: <class 'list'>
Conteúdo retornado pelo readlines:
['Olá, mundo.\n', 'Olá, EAD.']
Process finished with exit code 0
```

Impressão do atributo *mode*.

Imprimimos, **na linha 8**, o conteúdo em si, porém utilizamos a função interna **repr** para mostrar o conteúdo real contido da variável **conteudo**. Observe que foi retornado todo o texto existente no arquivo dados.txt, que também pode ser verificado no console.

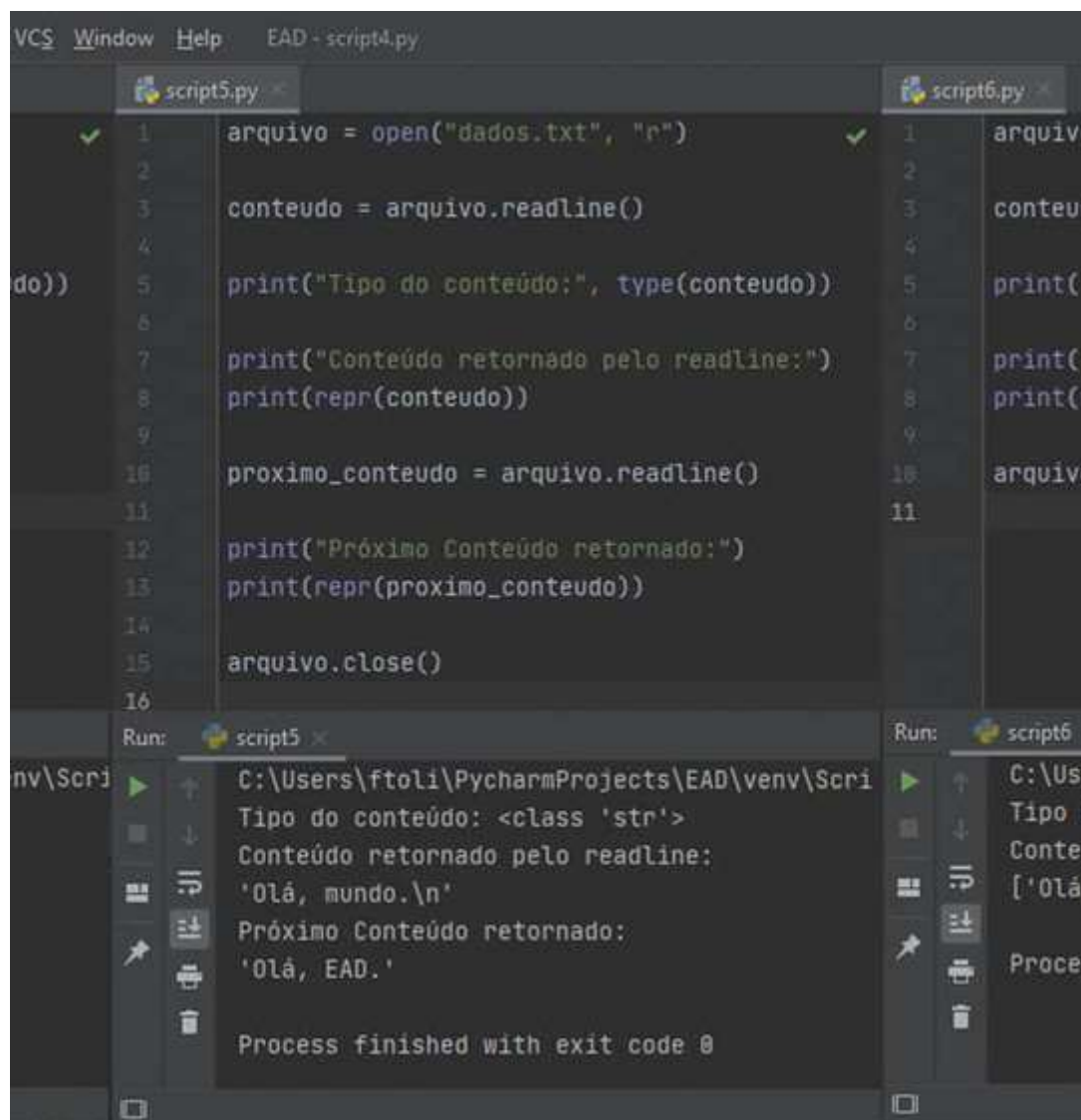
No **script5.py**, seguimos os mesmos passos do script anterior, porém, na linha 3, utilizamos o método **readline()**.

Na linha 5, verificamos que o tipo do conteúdo retornado pelo método `readline()` também é um objeto do tipo `str` (string).

Na linha 8, imprimimos a representação do conteúdo que contém apenas a **primeira linha** do arquivo `dados.txt` (incluindo o caractere de final de linha `\n`). Isso aconteceu porque, quando abrimos o arquivo utilizando o modo leitura (`'r'`), o cursor interno de leitura fica posicionado no início do arquivo.

Se chamarmos novamente o método `readline()`, linha 10, será retornado à próxima linha do arquivo, que foi impressa na linha 13. Confira a saída do script 5 no console abaixo dele; seguimos os mesmos passos do script anterior, porém, na linha 3, utilizamos o método `readline()`.

Veja tudo isso na imagem a seguir:



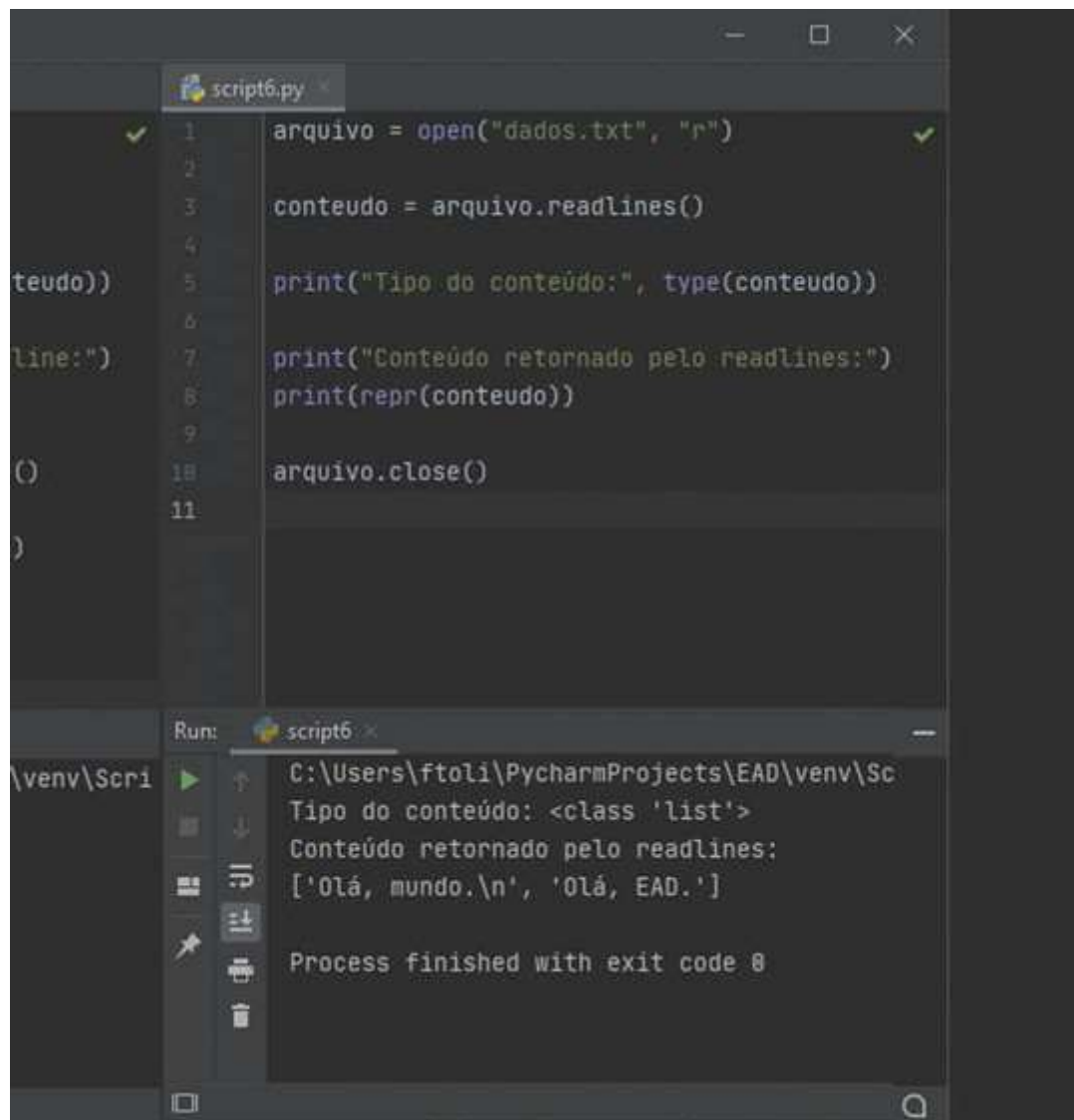
Script5 e sua saída.

No **script6.py**, seguimos novamente os mesmos passos, mas, desta vez, utilizamos o método `readlines()`.

Na linha 5, verificamos que o tipo do conteúdo retornado pelo método `readlines()` é um objeto do tipo *list* (lista).

Na linha 8, imprimimos o conteúdo retornado, que é uma lista na qual **cada item é uma linha do arquivo**. Veja a saída desse script no console abaixo dele.

Além dos três métodos já apresentados, os objetos do tipo arquivo são iteráveis. Com isso, podemos utilizar o laço **for** diretamente sobre os objetos desse tipo. Veja tudo isso na imagem a seguir:



The image shows a screenshot of a code editor with a file named `script6.py`. The code in the editor is as follows:

```
1 arquivo = open("dados.txt", "r")
2
3 conteudo = arquivo.readlines()
4
5 print("Tipo do conteúdo:", type(conteudo))
6
7 print("Conteúdo retornado pelo readlines:")
8 print(repr(conteudo))
9
10 arquivo.close()
11
```

Below the code editor is a terminal window titled `Run: script6`. It shows the output of the script:

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Sc
Tipo do conteúdo: <class 'list'>
Conteúdo retornado pelo readlines:
['Olá, mundo.\n', 'Olá, EAD.']

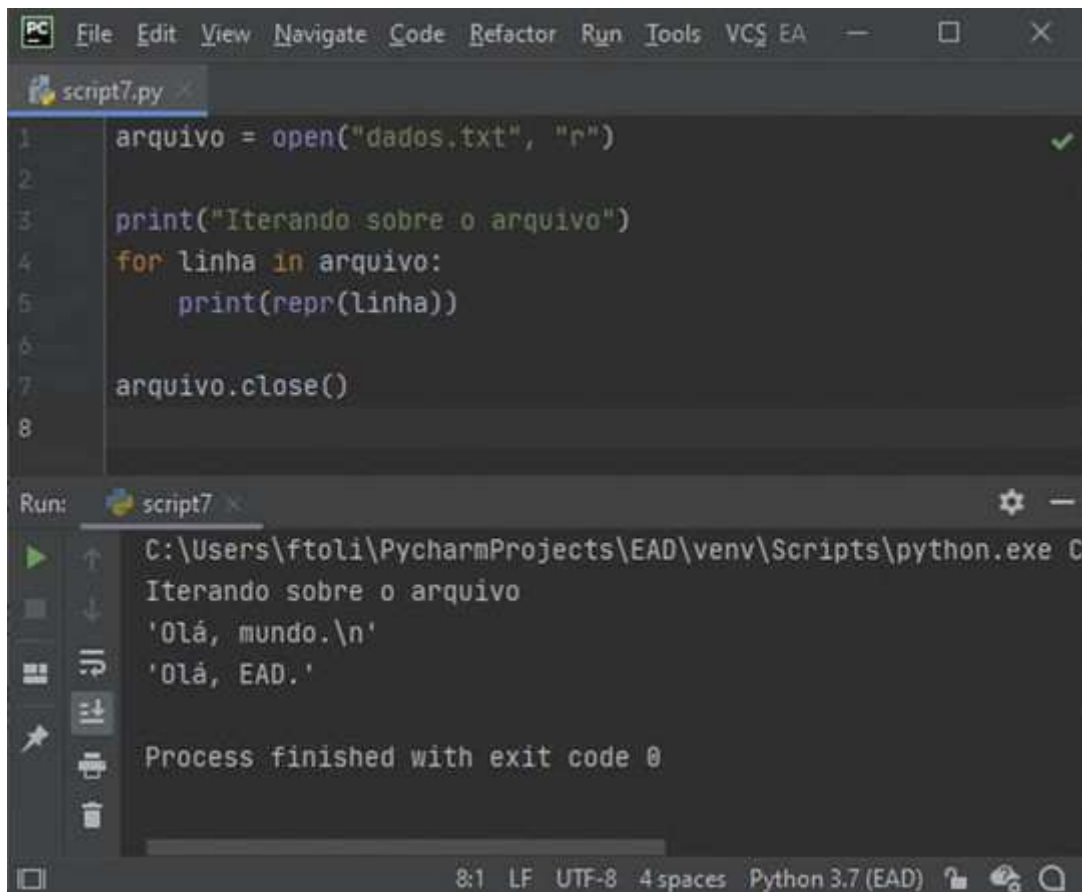
Process finished with exit code 0
```

Script6 e sua saída.

Veja agora como iterar diretamente sobre um arquivo:

Na linha 1, abrimos o arquivo da mesma maneira que fizemos nos exemplos anteriores.

Na linha 4, utilizamos o laço `for` para iterar diretamente sobre a variável `arquivo`. Para cada iteração, recebemos uma nova linha do arquivo, disponibilizada na variável `linha`, impressa na linha 5. Observe a saída do console abaixo da imagem:



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, and EA. A tab labeled 'script7.py' is open. The editor contains the following Python code:

```
1 arquivo = open("dados.txt", "r")
2
3 print("Iterando sobre o arquivo")
4 for linha in arquivo:
5     print(repr(linha))
6
7 arquivo.close()
8
```

Below the editor is the 'Run' window, which shows the execution of the script. The command used is 'C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe C'. The output is:

```
Iterando sobre o arquivo
'Olá, mundo.\n'
'Olá, EAD.'
```

At the bottom of the Run window, it states 'Process finished with exit code 0'. The status bar at the bottom indicates '8:1 LF UTF-8 4 spaces Python 3.7 (EAD)'.

Script7 e sua saída.

Quando precisamos abrir um arquivo muito grande, é inviável utilizar os métodos *read* e *readlines*, pois eles retornam todo o conteúdo do arquivo de uma só vez, seja na forma de string, seja na forma de lista. Isso pode consumir todos os recursos do computador, travando seu programa.

Nesses casos, precisamos chamar o método *readline* inúmeras vezes até o final do arquivo ou iterar diretamente sobre o objeto do tipo arquivo.

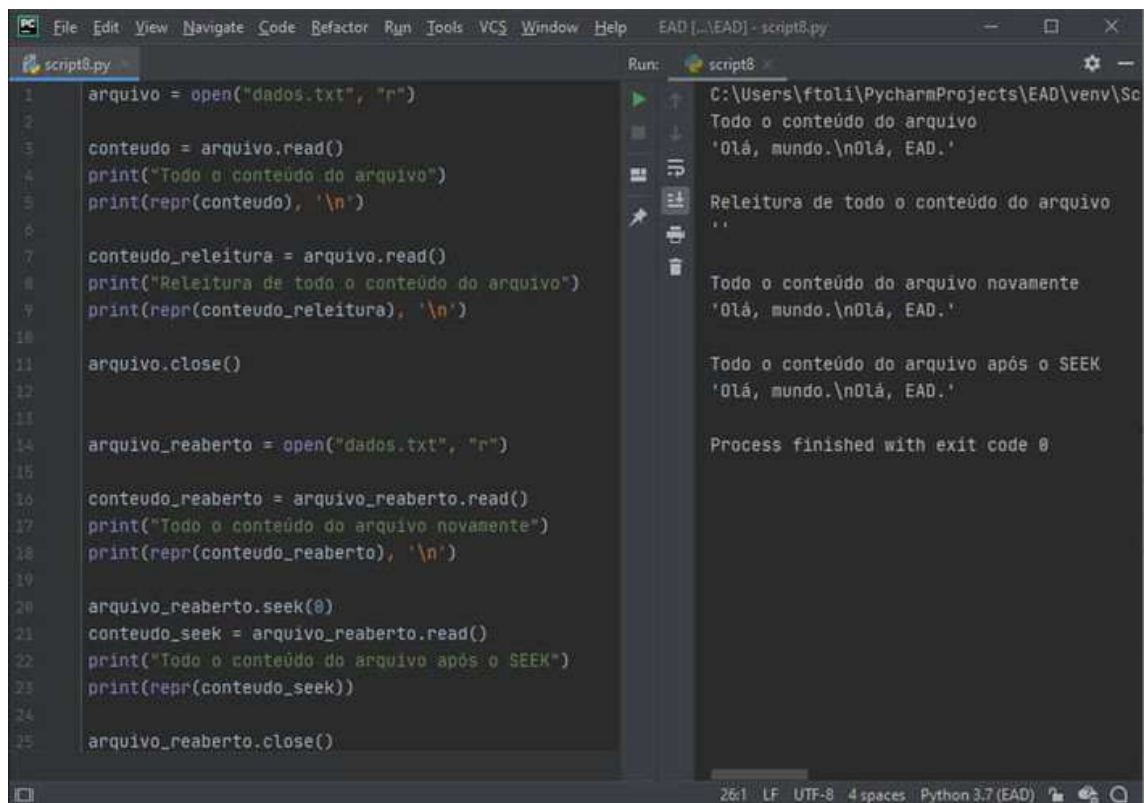
Após utilizar qualquer um dos métodos para leitura do arquivo apresentado, não podemos utilizá-los novamente. Isso acontece porque o cursor estará posicionado ao final do arquivo, e as chamadas aos métodos *read*, *readline* ou *readlines* retornarão vazias.

Para situações em que precisamos ler o conteúdo de um arquivo mais de uma vez, temos duas opções. Na sua opinião, quais são elas?

Fechar e abrir novamente o arquivo.

Utilizar o método *seek(n)*, passando como argumento o número da linha onde desejamos posicionar o cursor. A chamada *seek(0)* retorna o cursor para o início do arquivo.

Depois de conhecer a resposta correta, confira o exemplo do script8, onde exploramos na prática essa situação:



```
1 arquivo = open("dados.txt", "r")
2
3 conteudo = arquivo.read()
4 print("Todo o conteúdo do arquivo")
5 print(repr(conteudo), '\n')
6
7 conteudo_releitura = arquivo.read()
8 print("Releitura de todo o conteúdo do arquivo")
9 print(repr(conteudo_releitura), '\n')
10
11 arquivo.close()
12
13
14 arquivo_reaberto = open("dados.txt", "r")
15
16 conteudo_reaberto = arquivo_reaberto.read()
17 print("Todo o conteúdo do arquivo novamente")
18 print(repr(conteudo_reaberto), '\n')
19
20 arquivo_reaberto.seek(0)
21 conteudo_seek = arquivo_reaberto.read()
22 print("Todo o conteúdo do arquivo após o SEEK")
23 print(repr(conteudo_seek))
24
25 arquivo_reaberto.close()
```

Run: script8

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Sc
Todo o conteúdo do arquivo
'Olá, mundo.\nOlá, EAD.'
Releitura de todo o conteúdo do arquivo
''
Todo o conteúdo do arquivo novamente
'Olá, mundo.\nOlá, EAD.'
Todo o conteúdo do arquivo após o SEEK
'Olá, mundo.\nOlá, EAD.'
Process finished with exit code 0
```

Script8 e sua saída.

Vejamos agora o que se sucedeu em cada linha a seguir:

Na linha 1

Abrimos o arquivo em modo leitura.

Na linha 3

Lemos todo seu conteúdo utilizando o método *read*, que é impresso na linha 5. Veja no console à direita da imagem.

Na linha 7

Utilizamos o método *read* para ler novamente o conteúdo do arquivo e atribuímos o valor retornado à variável *conteudo_releitura*. Na linha 9, imprimimos o conteúdo dessa variável, que, conforme exibido no console, é uma string vazia (").

Para contornar esse problema, fechamos e abrimos o arquivo novamente, linhas 11 e 14, respectivamente. Na linha 16, utilizamos novamente o método *read* e imprimimos o

conteúdo retornado na linha 18. Observe que, mais uma vez, conseguimos acessar todo o conteúdo do arquivo.

Para demonstrar a utilização do método `seek`, no mesmo arquivo que já estava aberto, arquivo_reaberto, utilizamos o método `seek(0)`, linha 20. Imprimimos mais uma vez o conteúdo correto do arquivo na linha 23.

Toda a sequência pode ser acompanhada pelo console.

Atenção!

Todos os três métodos apresentados aceitam como parâmetro a quantidade de *bytes* que desejamos ler.

O valor padrão para esse parâmetro é -1, o que corresponde a todo o arquivo.

Escrevendo conteúdo em um arquivo

Confira agora como escrever conteúdo em um arquivo a partir da função `open`. Vamos lá!

A primeira modificação é alterar o modo de acesso ao arquivo. Para escrita de texto, podemos utilizar o modo `w` (*write*) ou o modo `a` (*append*), a seguir:

- O modo **w** abre o arquivo para escrita, truncando o arquivo em primeiro lugar. Caso ele não exista, será criado um.
- O modo **a** abre o arquivo para escrita, acrescentando conteúdo ao final dele, caso ele exista; do contrário, será criado um arquivo.

O Python disponibiliza dois métodos para escrita de conteúdo em um arquivo texto, para o modo **w** e para o modo **a**. Os métodos `write` e `writelines` são descritos abaixo:

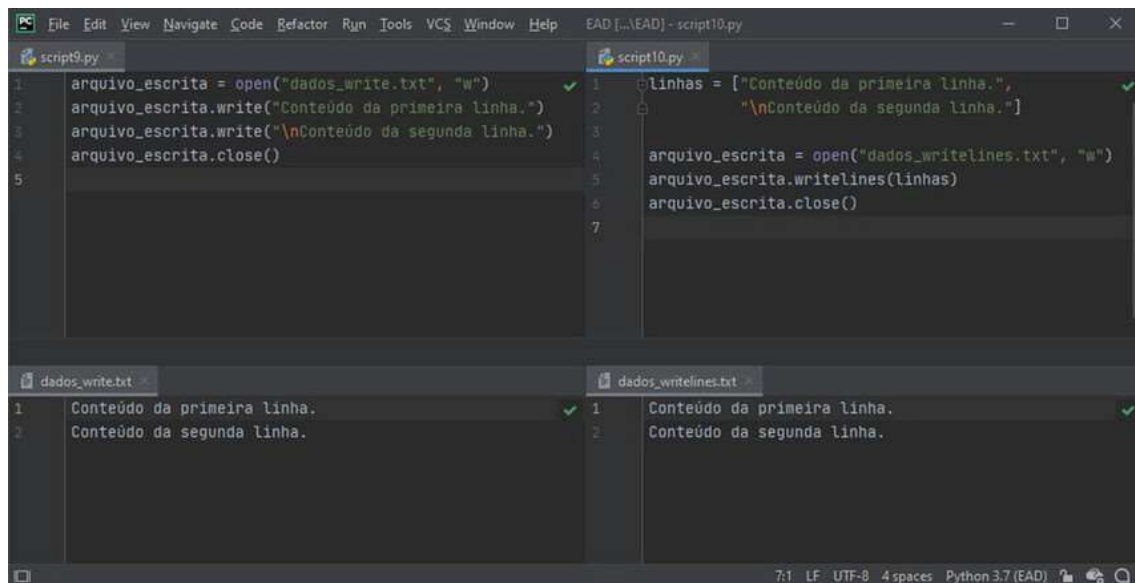
Write (texto)

Escreve todo o conteúdo passado como parâmetro no arquivo.

Writelines (iterável)

Escreve cada item do iterável (exemplo: lista) no arquivo.

No exemplo a seguir, vamos criar dois scripts para mostrar o uso do modo `w`. No primeiro, script9, vamos utilizar o método `write`. No segundo, script10, vamos utilizar o método `writelines`.



```
script9.py
1 arquivo_escrita = open("dados_write.txt", "w")
2 arquivo_escrita.write("Conteúdo da primeira linha.")
3 arquivo_escrita.write("\nConteúdo da segunda linha.")
4 arquivo_escrita.close()
5

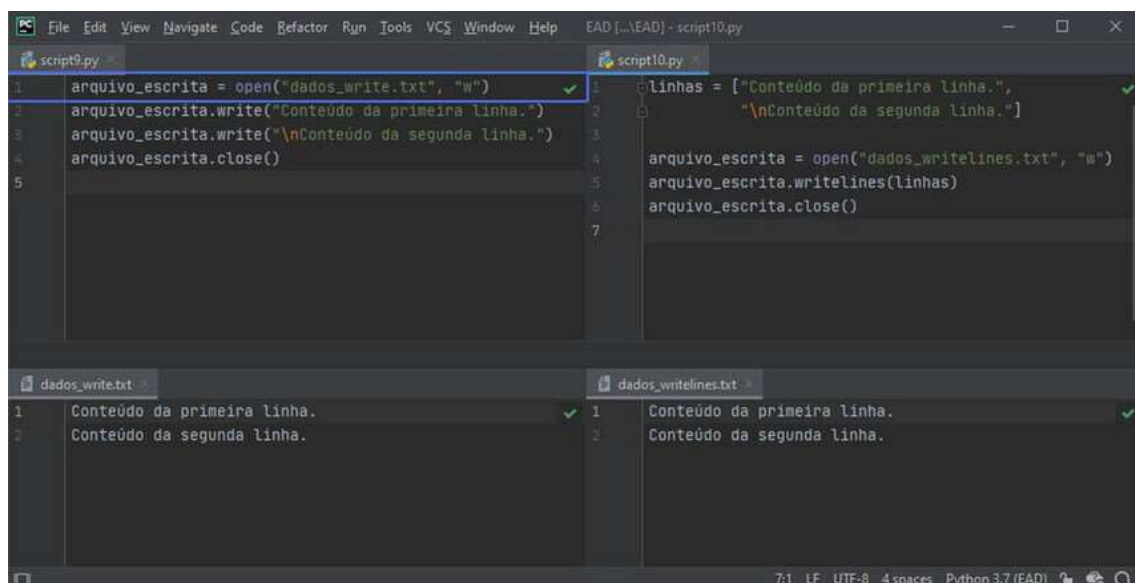
dados_write.txt
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.

script10.py
1 linhas = ["Conteúdo da primeira linha.",
2          "\nConteúdo da segunda linha."]
3
4 arquivo_escrita = open("dados_writelines.txt", "w")
5 arquivo_escrita.writelines(linhas)
6 arquivo_escrita.close()
7

dados_writelines.txt
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
```

Scripts 9 e 10 e suas saídas.

Temos os scripts 9 e 10 e suas respectivas saídas.



```
script9.py
1 arquivo_escrita = open("dados_write.txt", "w")
2 arquivo_escrita.write("Conteúdo da primeira linha.")
3 arquivo_escrita.write("\nConteúdo da segunda linha.")
4 arquivo_escrita.close()
5

dados_write.txt
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.

script10.py
1 linhas = ["Conteúdo da primeira linha.",
2          "\nConteúdo da segunda linha."]
3
4 arquivo_escrita = open("dados_writelines.txt", "w")
5 arquivo_escrita.writelines(linhas)
6 arquivo_escrita.close()
7

dados_writelines.txt
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
```

Abertura do arquivo dados_write.txt.

Abrimos, **no script9**, o arquivo dados_write.txt para escrita utilizando o modo w na linha 1.

The screenshot shows an IDE with four open files. The top-left file, `script9.py`, contains the following code:

```
1 arquivo_escrita = open("dados_write.txt", "w")
2 arquivo_escrita.write("Conteúdo da primeira linha.")
3 arquivo_escrita.write("\nConteúdo da segunda linha.")
4 arquivo_escrita.close()
5
```

The top-right file, `script10.py`, contains the following code:

```
1 linhas = ["Conteúdo da primeira linha.",
2           "\nConteúdo da segunda linha."]
3
4 arquivo_escrita = open("dados_writelines.txt", "w")
5 arquivo_escrita.writelines(linhas)
6 arquivo_escrita.close()
7
```

The bottom-left file, `dados_write.txt`, contains the output of `script9.py`:

```
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
```

The bottom-right file, `dados_writelines.txt`, contains the output of `script10.py`:

```
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
```

Escrita dos conteúdos pelo método `write`.

Escrevemos os conteúdos utilizando o método `write`, conforme linhas 2 e 3, e fechamos o arquivo, como exposto na linha 4.

This screenshot is identical to the one above, showing the same IDE with the same four files and their contents. The only difference is that the `dados_write.txt` file is highlighted with a blue border, indicating it is the focus of the current observation.

Resultado do arquivo `dados_write.txt`.

Observe como ficou o arquivo `dados_write.txt`, abaixo do `script9`, após a execução desse script.

The screenshot shows an IDE with two Python scripts and their output files. The top-left pane shows `script9.py` with the following code:

```
1 arquivo_escrita = open("dados_write.txt", "w")
2 arquivo_escrita.write("Conteúdo da primeira linha.")
3 arquivo_escrita.write("\nConteúdo da segunda linha.")
4 arquivo_escrita.close()
```

The top-right pane shows `script10.py` with the following code:

```
1 linhas = ["Conteúdo da primeira linha.",
2           "\nConteúdo da segunda linha."]
3
4 arquivo_escrita = open("dados_writelines.txt", "w")
5 arquivo_escrita.writelines(linhas)
6 arquivo_escrita.close()
7
```

The bottom-left pane shows the output of `dados_write.txt`:

```
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
```

The bottom-right pane shows the output of `dados_writelines.txt`:

```
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
```

Resultado do arquivo `dados_write.txt`.

Criamos, no **script10**, uma lista chamada **linhas** na linha 1. Abrimos o arquivo `dados_write.txt` para escrita na linha 4, utilizando o mesmo modo de acesso ao arquivo, modo `w`. Para escrever o conteúdo da lista **linhas** no arquivo, utilizamos o método `writelines`, linha 5.

The screenshot shows an IDE with two Python scripts and their output files. The top-left pane shows `script9.py` with the following code:

```
1 arquivo_escrita = open("dados_write.txt", "w")
2 arquivo_escrita.write("Conteúdo da primeira linha.")
3 arquivo_escrita.write("\nConteúdo da segunda linha.")
4 arquivo_escrita.close()
```

The top-right pane shows `script10.py` with the following code:

```
1 linhas = ["Conteúdo da primeira linha.",
2           "\nConteúdo da segunda linha."]
3
4 arquivo_escrita = open("dados_writelines.txt", "w")
5 arquivo_escrita.writelines(linhas)
6 arquivo_escrita.close()
7
```

The bottom-left pane shows the output of `dados_write.txt`:

```
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
```

The bottom-right pane shows the output of `dados_writelines.txt`:

```
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
```

Resultado do conteúdo do arquivo `dados_writelines.txt`.

Verifique também o conteúdo do arquivo `dados_writelines.txt` após a execução do script, abaixo do `script10` na imagem.

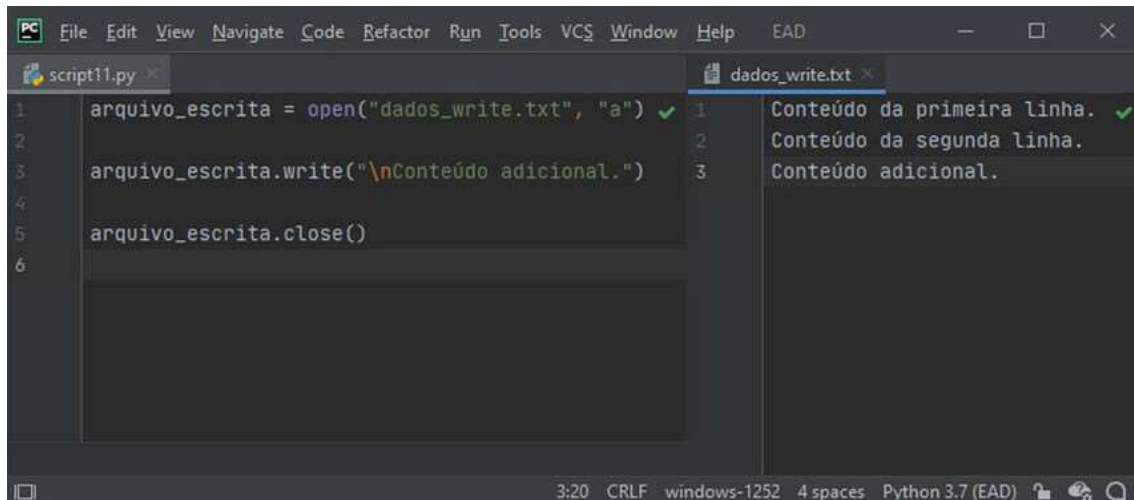
Fique atento às seguintes orientações:

- O Python não insere quebra de linha (`'\n'`) entre os elementos da lista. Precisamos fazer isso **manualmente**!

- Como o modo w trunca o arquivo, ou seja, remove todo o conteúdo do arquivo, caso ele exista, podemos executar esses scripts repetidas vezes e, ainda assim, o resultado será sempre o mesmo.

No próximo exemplo, vamos mostrar como utilizar o modo append (a) para adicionar conteúdo a um arquivo já existente.

Para isso, vamos abrir o arquivo dados_write.txt, criado pelo script9, utilizando o modo a. Utilizamos esse modo para acrescentar conteúdo a um arquivo. Confira o próximo script:

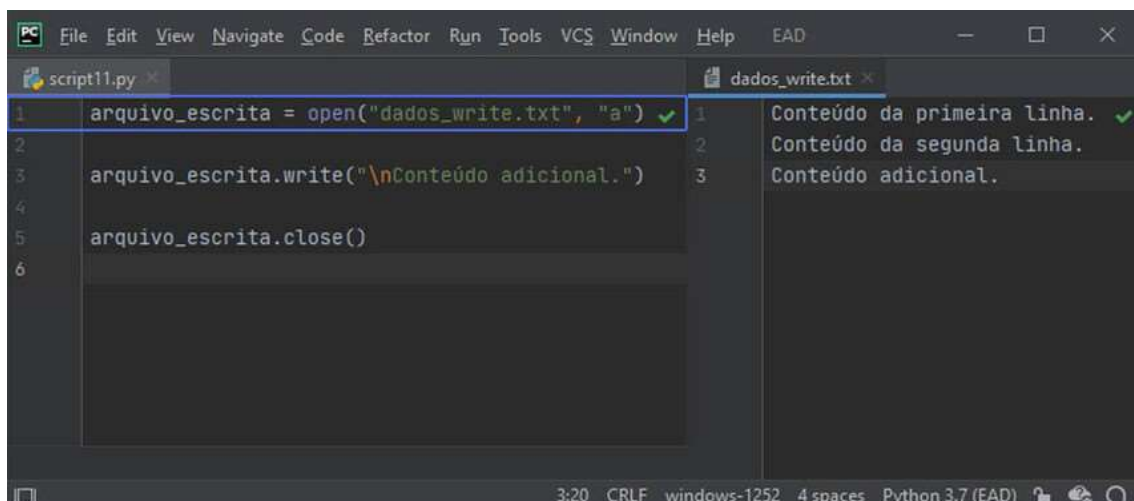


```
script11.py
1 arquivo_escrita = open("dados_write.txt", "a")
2
3 arquivo_escrita.write("\nConteúdo adicional.")
4
5 arquivo_escrita.close()
6

dados_write.txt
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
3 Conteúdo adicional.
```

Script 11 e saída.

Temos o script11 e sua saída.



```
script11.py
1 arquivo_escrita = open("dados_write.txt", "a")
2
3 arquivo_escrita.write("\nConteúdo adicional.")
4
5 arquivo_escrita.close()
6

dados_write.txt
1 Conteúdo da primeira linha.
2 Conteúdo da segunda linha.
3 Conteúdo adicional.
```

Abertura do arquivo dados_write.txt pelo append (a).

Abrimos, em primeiro lugar, na linha 1, o arquivo dados_write.txt utilizando o modo escrita a (append).

```
1 arquivo_escrita = open("dados_write.txt", "a") ✓
2
3 arquivo_escrita.write("\nConteúdo adicional.")
4
5 arquivo_escrita.close()
6
```

```
1 Conteúdo da primeira linha. ✓
2 Conteúdo da segunda linha.
3 Conteúdo adicional.
```

Utilização do método *write*.

Utilizamos, **na linha 3**, o método *write* para acrescentar o texto `"\nConteúdo adicional."` ao final do arquivo `dados_write.txt`.

```
1 arquivo_escrita = open("dados_write.txt", "a") ✓
2
3 arquivo_escrita.write("\nConteúdo adicional.")
4
5 arquivo_escrita.close()
6
```

```
1 Conteúdo da primeira linha. ✓
2 Conteúdo da segunda linha.
3 Conteúdo adicional.
```

Fechamento do arquivo.

Fechamos, **na linha 5**, o arquivo.

```
1 arquivo_escrita = open("dados_write.txt", "a") ✓
2
3 arquivo_escrita.write("\nConteúdo adicional.")
4
5 arquivo_escrita.close()
6
```

```
1 Conteúdo da primeira linha. ✓
2 Conteúdo da segunda linha.
3 Conteúdo adicional.
```

Resultado do conteúdo final do arquivo.

Observe como ficou o conteúdo final do arquivo à direita da imagem, onde a nova frase foi posicionada corretamente ao final do arquivo.

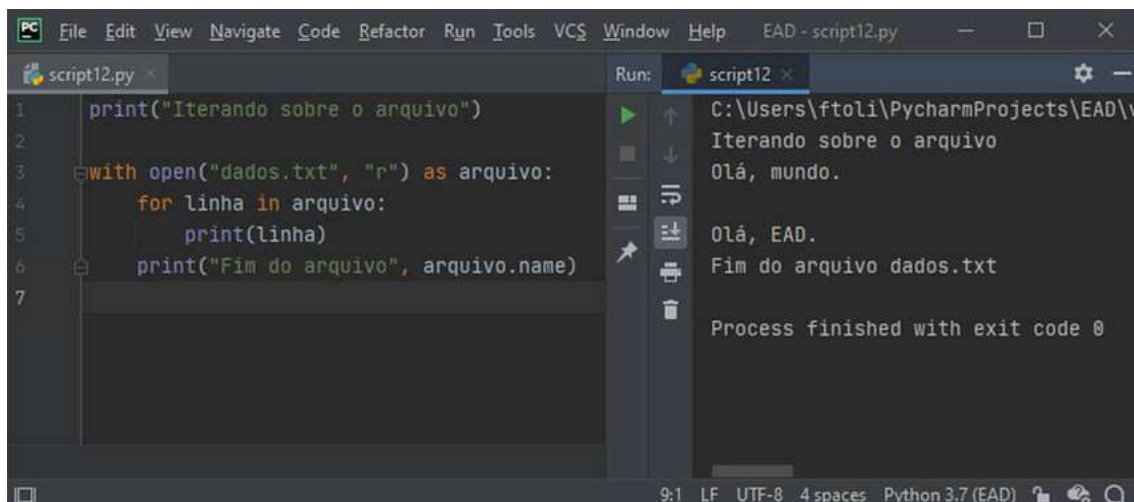
Boas práticas

Ao lidar com arquivos, devemos utilizar a palavra reservada *with*, disponibilizada pelo Python. Ela garante que o arquivo será fechado adequadamente após utilizarmos o arquivo, não sendo necessário chamar o método `close` explicitamente. A sintaxe de utilização do *with* é:

with open(caminho, modo) as nome: (seu código indentado)

Iniciamos com a palavra reservada *with*, seguida da função *open*, a palavra reservada *as*, um nome de variável que receberá o objeto do tipo arquivo e dois pontos. Todo o código indentado posteriormente está dentro do contexto do *with*, no qual o arquivo referenciado pela variável *nome* estará disponível.

Veja como utilizar o *with* no exemplo a seguir. Clique nas setas e acompanhe:



The screenshot shows the PyCharm IDE interface. On the left, the editor displays a Python script named `script12.py` with the following code:

```
1 print("Iterando sobre o arquivo")
2
3 with open("dados.txt", "r") as arquivo:
4     for linha in arquivo:
5         print(linha)
6 print("Fim do arquivo", arquivo.name)
7
```

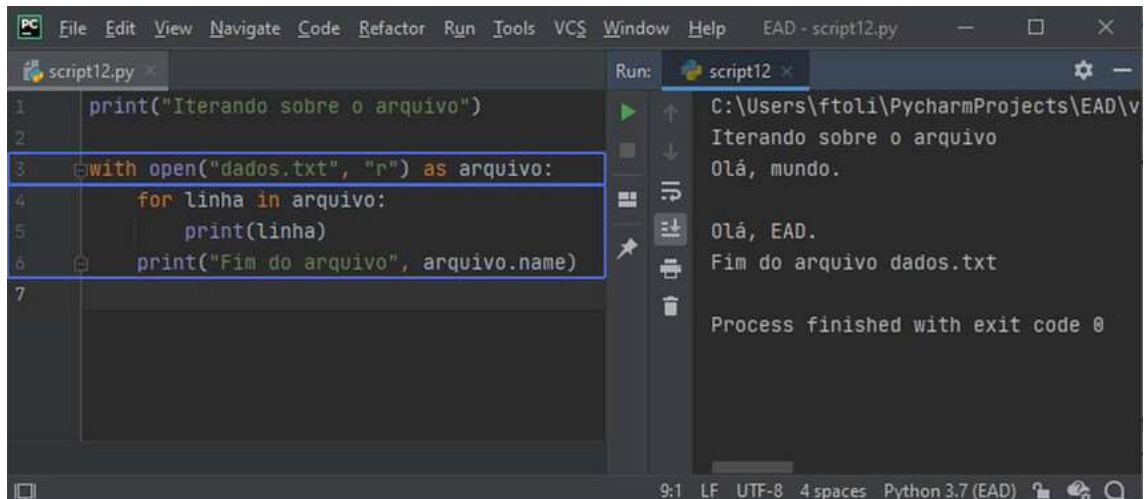
On the right, the 'Run' console shows the output of the script:

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe C:\Users\ftoli\PycharmProjects\EAD\script12.py
Iterando sobre o arquivo
Olá, mundo.
Olá, EAD.
Fim do arquivo dados.txt
Process finished with exit code 0
```

The status bar at the bottom indicates the file is at line 9, column 1, using LF line endings, UTF-8 encoding, 4 spaces for indentation, and Python 3.7 (EAD) interpreter.

Script 12 e saída.

Temos o `script12` e sua saída.



The screenshot shows the PyCharm IDE with a Python script named `script12.py` open. The script contains the following code:

```
1 print("Iterando sobre o arquivo")
2
3 with open("dados.txt", "r") as arquivo:
4     for linha in arquivo:
5         print(linha)
6     print("Fim do arquivo", arquivo.name)
7
```

The code is running, and the output is displayed in the Run console on the right:

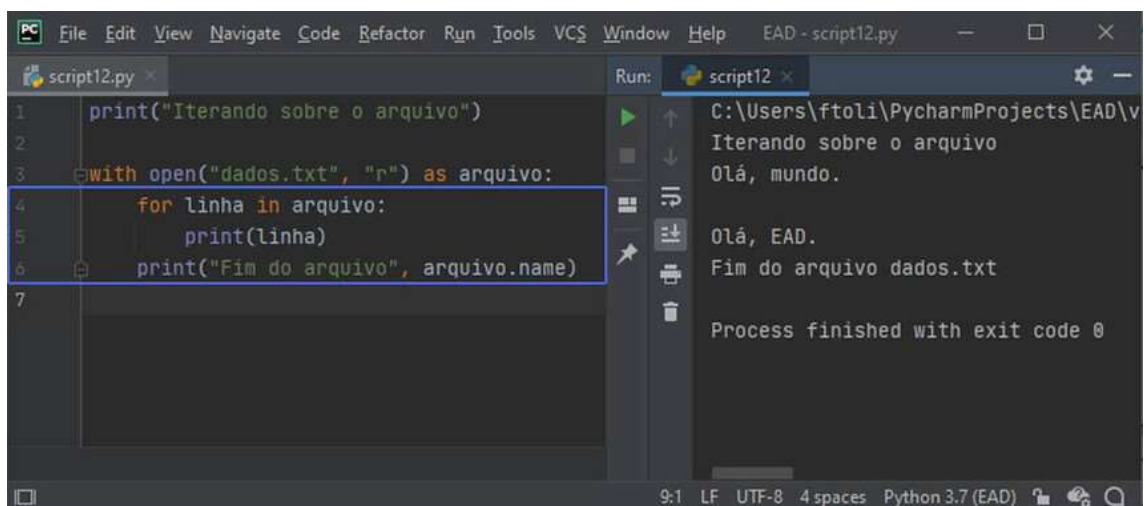
```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe C:\Users\ftoli\PycharmProjects\EAD\script12.py
Iterando sobre o arquivo
Olá, mundo.

Olá, EAD.
Fim do arquivo dados.txt

Process finished with exit code 0
```

Abertura do arquivo `dados.txt` no modo leitura.

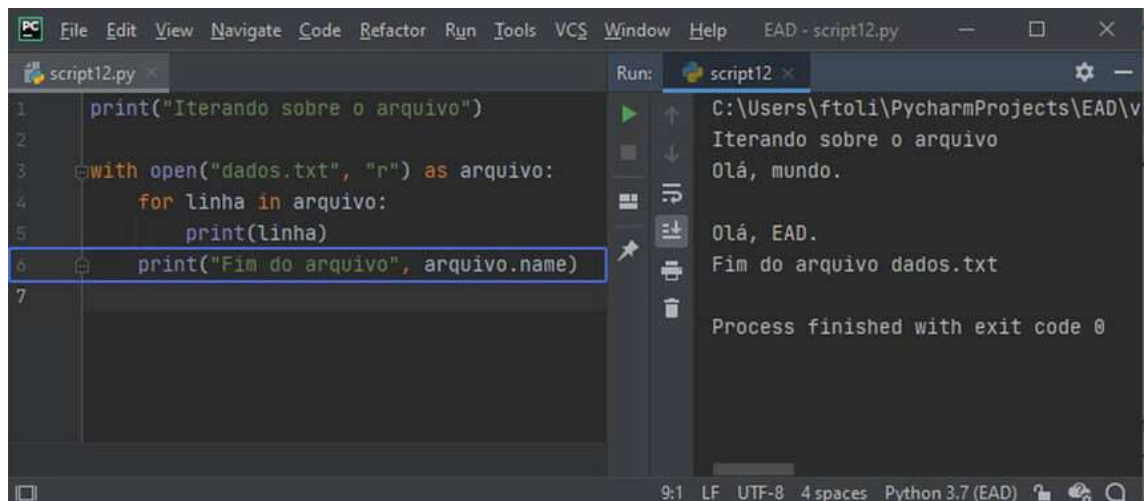
Utilizamos a sintaxe do `with`, na linha 3, e o arquivo `dados.txt` é aberto no modo leitura, atribuído à variável `arquivo`. Esta variável está disponível em todo o escopo do `with`, linhas 4, 5 e 6.



This screenshot is identical to the one above, showing the same Python script and its execution output in the PyCharm IDE.

Iteração sobre o conteúdo do arquivo.

Iteramos, **nas linhas 4 e 5**, sobre o conteúdo do arquivo e imprimimos linha por linha.



The screenshot shows the PyCharm IDE with a Python script named `script12.py` open. The script contains the following code:

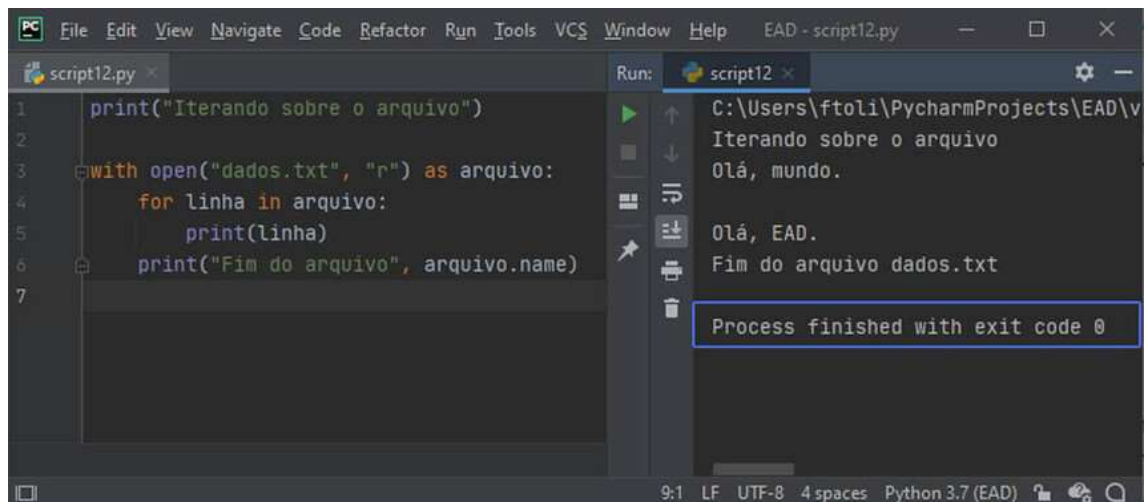
```
1 print("Iterando sobre o arquivo")
2
3 with open("dados.txt", "r") as arquivo:
4     for linha in arquivo:
5         print(linha)
6     print("Fim do arquivo", arquivo.name)
7
```

The line `print("Fim do arquivo", arquivo.name)` on line 6 is highlighted with a blue box. To the right, the Run console shows the output of the script:

```
C:\Users\ftoli\PycharmProjects\EAD\script12.py
Iterando sobre o arquivo
Olá, mundo.
Olá, EAD.
Fim do arquivo dados.txt
Process finished with exit code 0
```

Impressão do nome do arquivo.

Imprimimos, **na linha 6**, o nome do arquivo.



This screenshot is identical to the previous one, showing the same Python script and Run console output. However, the text `Process finished with exit code 0` in the Run console is now highlighted with a blue box.

Saídas no console à direita.

Verifique as saídas no console à direita.