

FUNÇÕES DE MANIPULAÇÃO DE STRINGS

Link de download dos arquivos do projeto [link](#).

Manipulação de strings

Métodos de manipulação de strings

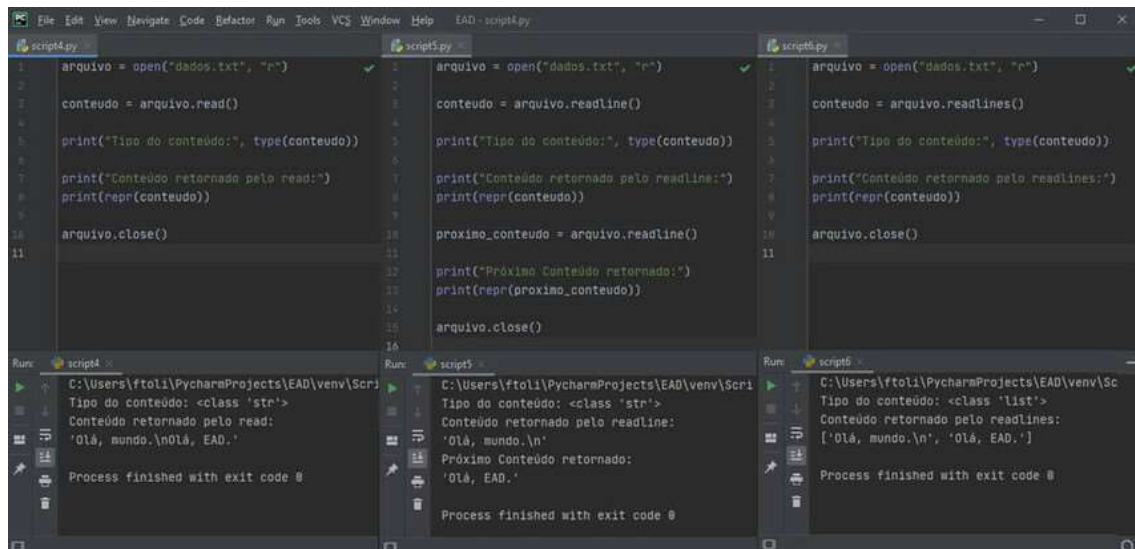


Durante a vida de programador, é muito comum nos depararmos com situações em que precisamos realizar alguns ajustes e operações sobre os textos lidos de arquivos, como remover espaço em branco, colocar todas as letras maiúsculas, substituir e contar palavras.

Neste módulo, veremos alguns métodos presentes nos objetos do tipo `str` (string), que são muito utilizados em conjunto com a manipulação de arquivos.

Método *strip*

Como mostrado nos scripts da imagem abaxio (script5), ao ler o conteúdo do arquivo, o Python retorna os caracteres de final de linha (`\r` e `\n`). Muitas vezes, essa informação não é necessária, principalmente se estivermos tratando uma linha de cada vez. Veja:



```
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()
```

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

```
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()
```

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

```
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()
```

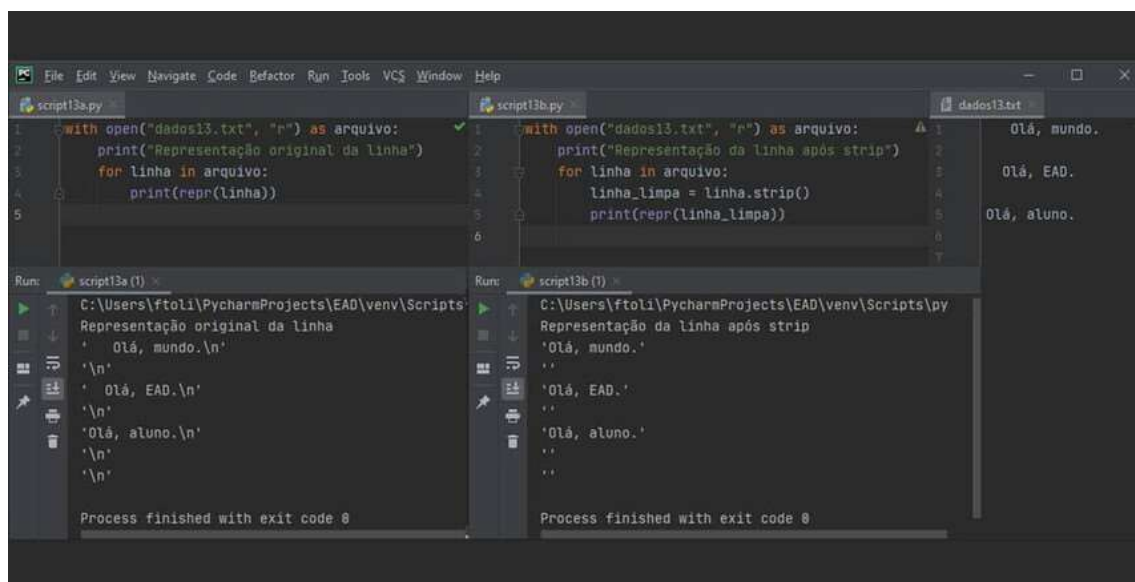
Run: script6

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

Process finished with exit code 0

Leitura do arquivo script5.

Dependendo do objetivo, esses caracteres são considerados lixo e podem atrapalhar o processamento que desejamos realizar. Para remover esses caracteres e também espaços em branco adicionais, o tipo `str` disponibiliza o método `strip()`. O método *strip* remove os caracteres do início e do final da linha. Observe o uso deste método no exemplo a seguir.



```
1 with open("dados13.txt", "r") as arquivo:
2     print("Representação original da linha")
3     for linha in arquivo:
4         print(repr(linha))
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Run: script13a

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts
Representação original da linha
' Olá, mundo.\n'
'\n'
' Olá, EAD.\n'
'\n'
'Olá, aluno.\n'
'\n'
'\n'
```

Process finished with exit code 0

```
1 with open("dados13.txt", "r") as arquivo:
2     print("Representação da linha após strip")
3     for linha in arquivo:
4         linha_limpa = linha.strip()
5         print(repr(linha_limpa))
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

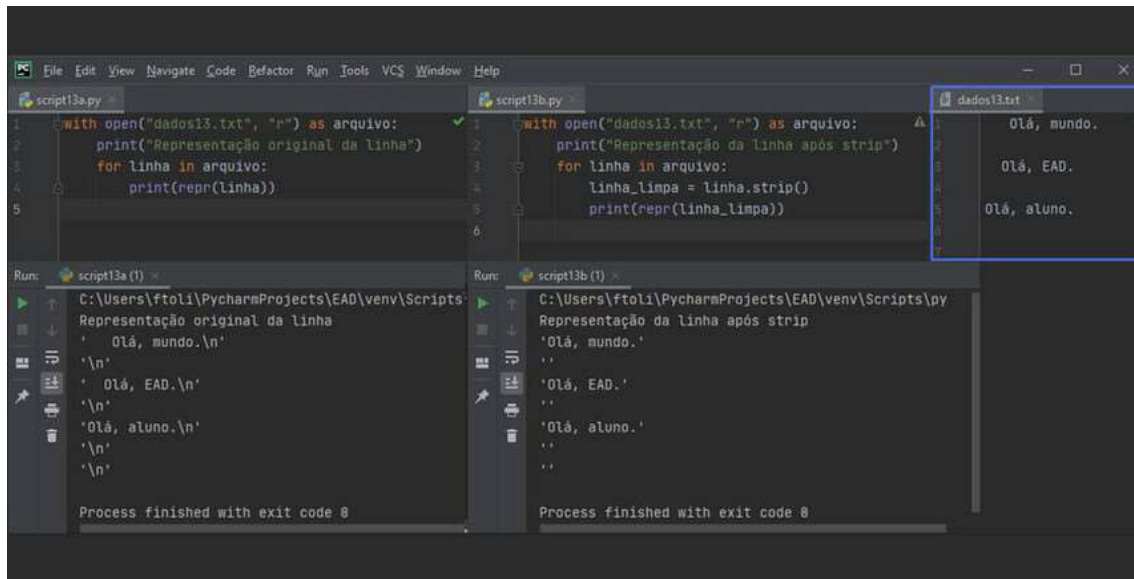
Run: script13b

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\py
Representação da linha após strip
'Olá, mundo.'
''
'Olá, EAD.'
''
'Olá, aluno.'
```

Process finished with exit code 0

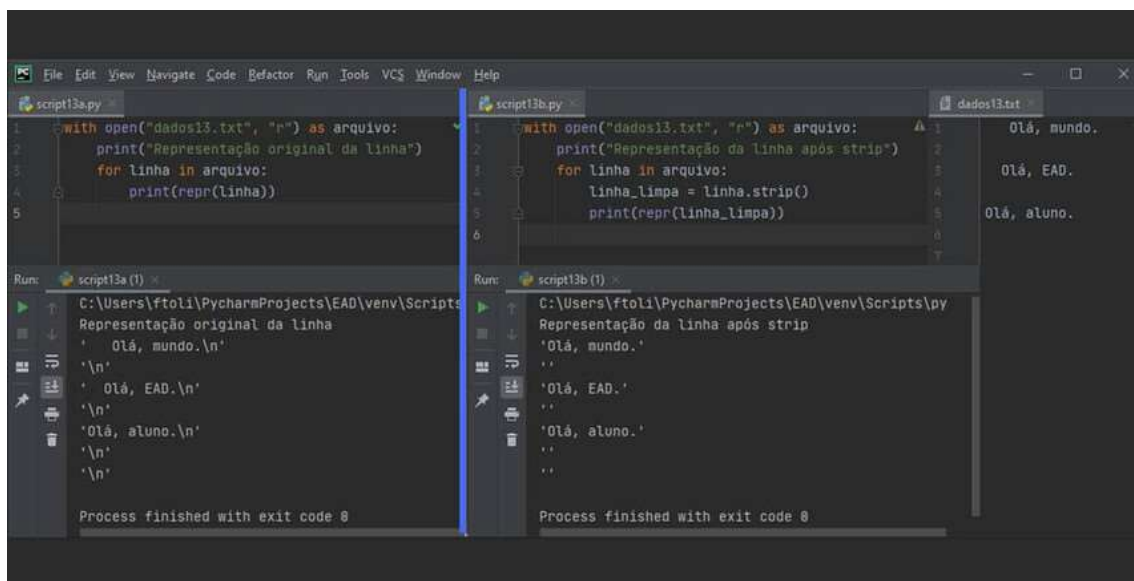
Scripts 13a e 13b, suas saídas e arquivo dados13.txt.

Temos os **scripts 13a e 13b**, suas respectivas saídas e seu arquivo dados13.txt.



Abertura do arquivo dados13.txt.

Abrimos o arquivo **dados13.txt**, exibido à direita da figura, e imprimimos o conteúdo de cada linha desse arquivo. Vamos utilizar o método `repr` para verificar o real conteúdo da string.



Separação do script em duas partes.

Separamos o script em duas partes, **a** e **b**, de forma a se perceber melhor o resultado da utilização do método `strip()`.

```
script13a.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Representação original da linha")
3     for linha in arquivo:
4         print(repr(linha))

Run: script13a (1)
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe
Representação original da linha
' Olá, mundo.\n'
'\n'
' Olá, EAD.\n'
'\n'
'Olá, aluno.\n'
'\n'
'\n'
Process finished with exit code 0

script13b.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Representação da linha após strip")
3     for linha in arquivo:
4         linha_limpa = linha.strip()
5         print(repr(linha_limpa))

Run: script13b (1)
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe
Representação da linha após strip
'Olá, mundo.'
''
'Olá, EAD.'
''
'Olá, aluno.'
''
''
Process finished with exit code 0

dados13.txt
1 Olá, mundo.
2
3 Olá, EAD.
4
5 Olá, aluno.
6
7
```

Abertura do arquivo em modo leitura.

Abrimos o arquivo em modo leitura, na primeira parte, **script13a**, utilizando o `with` na linha 1, iteramos sobre o objeto **arquivo** na linha 3 e imprimimos o conteúdo de cada linha na linha 4.

```
script13a.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Representação original da linha")
3     for linha in arquivo:
4         print(repr(linha))

Run: script13a (1)
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe
Representação original da linha
' Olá, mundo.\n'
'\n'
' Olá, EAD.\n'
'\n'
'Olá, aluno.\n'
'\n'
'\n'
Process finished with exit code 0

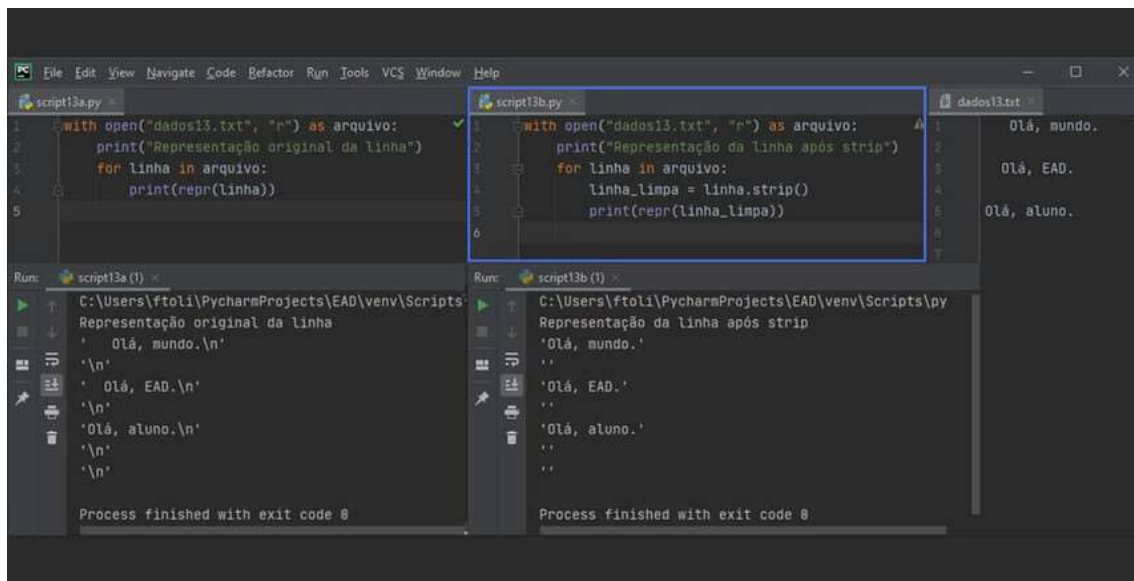
script13b.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Representação da linha após strip")
3     for linha in arquivo:
4         linha_limpa = linha.strip()
5         print(repr(linha_limpa))

Run: script13b (1)
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe
Representação da linha após strip
'Olá, mundo.'
''
'Olá, EAD.'
''
'Olá, aluno.'
''
''
Process finished with exit code 0

dados13.txt
1 Olá, mundo.
2
3 Olá, EAD.
4
5 Olá, aluno.
6
7
```

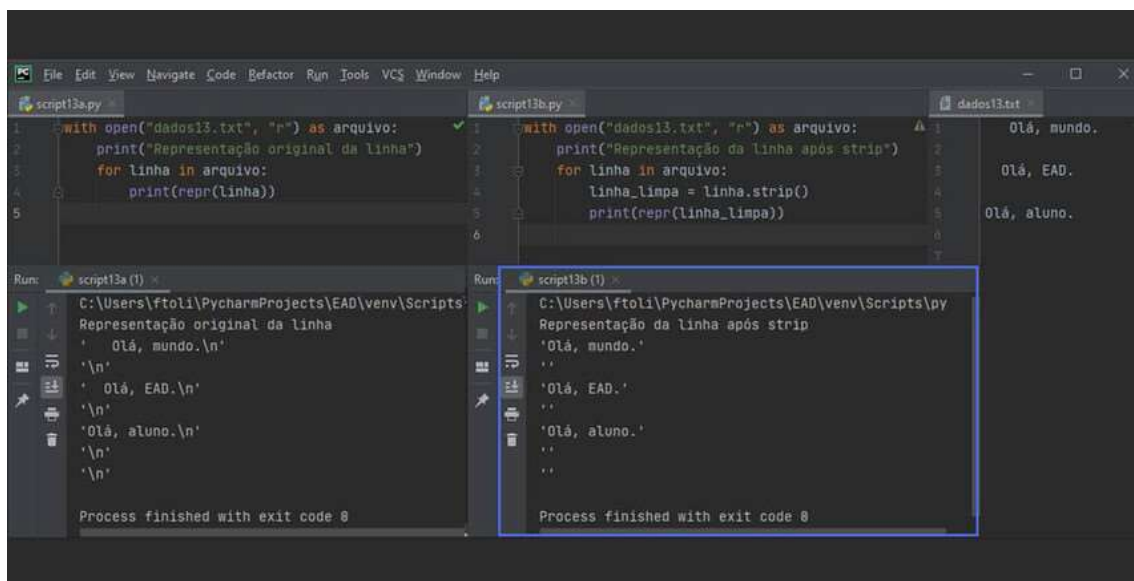
Presença dos espaços em branco e caracteres de nova linha.

Observe, no console abaixo do **script13a**, que os espaços em branco no início de cada linha e os caracteres de nova linha (`\n`) estão presentes na string.



Abertura e iteração do mesmo arquivo.

Abrimos, na segunda parte, **script13b**, o mesmo arquivo e iteramos da mesma forma. Porém, desta vez, “limpamos” a linha utilizando o método `strip`, na linha 4. Esse método retorna uma nova string, que é armazenada na variável `linha_limpa`. Em seguida, na linha 5, imprimimos a representação dessa variável.

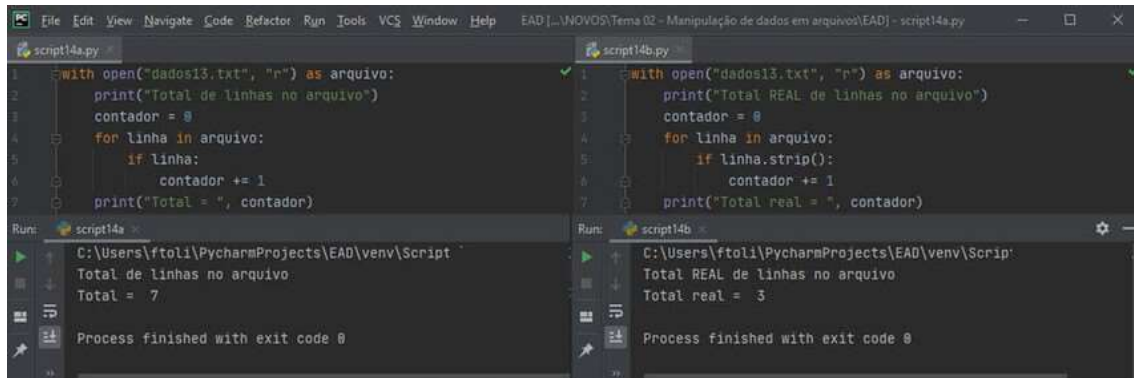


Saída do console.

Observe a saída do console e verifique que os caracteres em branco do início da linha e os caracteres de nova linha foram removidos.

Um exemplo real de utilização do `strip` é quando desejamos contar a quantidade de linhas com algum conteúdo em um arquivo. Dependendo do tamanho do arquivo, é inviável remover manualmente as linhas em branco. Para resolver esses problemas, podemos utilizar o método *strip* da seguinte forma:

Vamos abrir o mesmo arquivo do exemplo anterior, dados13.txt, e iterar sobre cada linha, incrementando um contador, que será impresso no final. Para melhor ilustrar, vamos mostrar como contar as linhas sem usar o método strip, script14a, com o strip script14b, veja:



```
script14a.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha:
6             contador += 1
7     print("Total = ", contador)

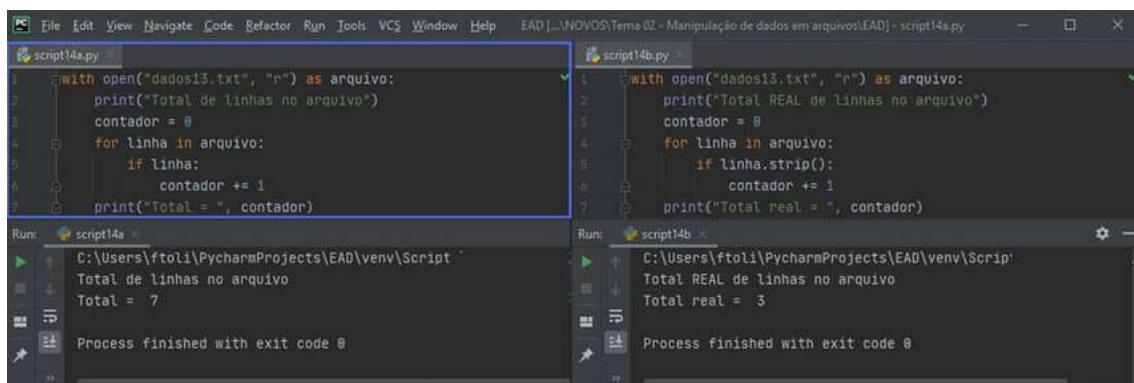
Run: script14a
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total de linhas no arquivo
Total = 7
Process finished with exit code 0

script14b.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total REAL de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha.strip():
6             contador += 1
7     print("Total real = ", contador)

Run: script14b
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total REAL de linhas no arquivo
Total real = 3
Process finished with exit code 0
```

Scripts 14a e 14b e suas respectivas saídas.

Temos os scripts 14a e 14b e suas respectivas saídas.



```
script14a.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha:
6             contador += 1
7     print("Total = ", contador)

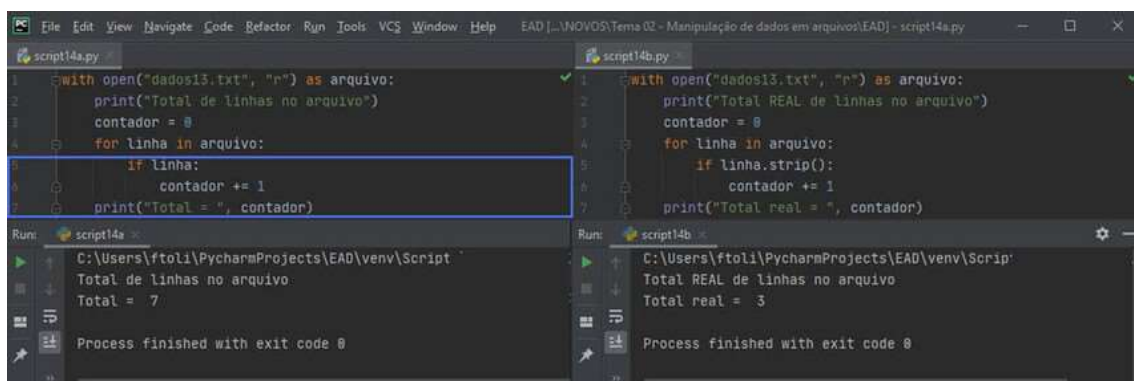
Run: script14a
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total de linhas no arquivo
Total = 7
Process finished with exit code 0

script14b.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total REAL de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha.strip():
6             contador += 1
7     print("Total real = ", contador)

Run: script14b
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total REAL de linhas no arquivo
Total real = 3
Process finished with exit code 0
```

Abertura do arquivo em modo leitura.

Abrimos, no primeiro script, o arquivo em modo leitura na linha 1, criamos e inicializamos uma variável inteira **contador**, linha 3, e iteramos seu conteúdo linha a linha, utilizando o loop for, linha 4.



```
script14a.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha:
6             contador += 1
7     print("Total = ", contador)

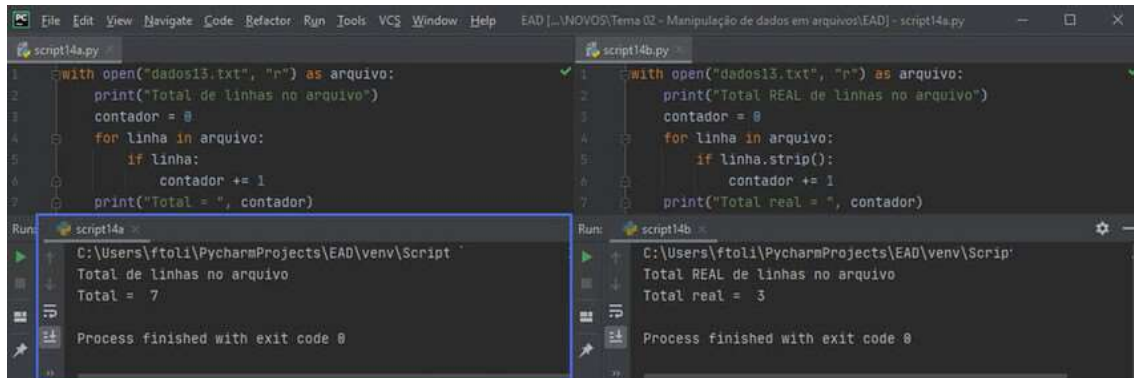
Run: script14a
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total de linhas no arquivo
Total = 7
Process finished with exit code 0

script14b.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total REAL de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha.strip():
6             contador += 1
7     print("Total real = ", contador)

Run: script14b
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total REAL de linhas no arquivo
Total real = 3
Process finished with exit code 0
```

Verificação da variável conter ou não conteúdo.

Verificamos, na linha 5, se a variável `linha`, do tipo `str`, contém algum conteúdo. Lembrando que uma variável do tipo `string` testa para falso apenas se seu conteúdo for vazio (" ou ""); caso a `string` contenha, pelo menos, um espaço, ela testará verdadeiro. Continuando, na linha 6, caso exista algum conteúdo, incrementamos o contador.



```
script14a.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha:
6             contador += 1
7     print("Total = ", contador)

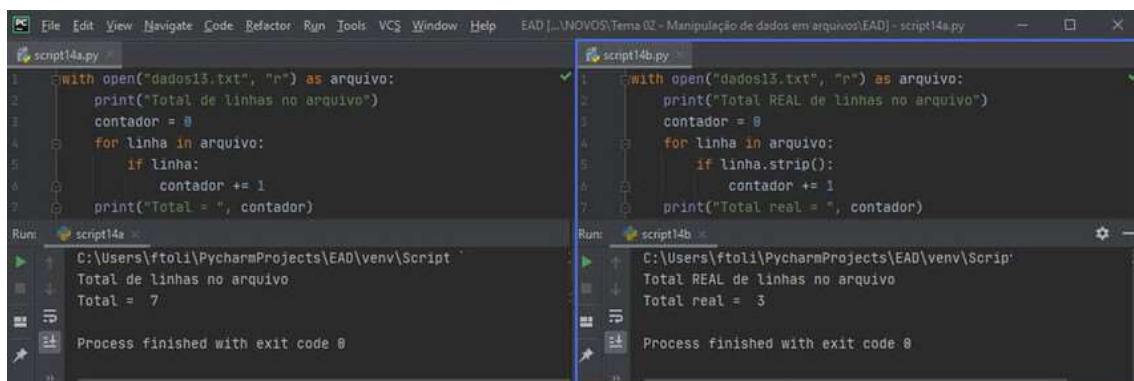
Run: script14a
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total de linhas no arquivo
Total = 7
Process finished with exit code 0

script14b.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total REAL de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha.strip():
6             contador += 1
7     print("Total real = ", contador)

Run: script14b
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total REAL de linhas no arquivo
Total real = 3
Process finished with exit code 0
```

Obtenção do valor 7 pelo contador.

O contador, para o primeiro script, obteve valor 7, indicando que o arquivo contém 7 linhas, conforme console abaixo do script14a.



```
script14a.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha:
6             contador += 1
7     print("Total = ", contador)

Run: script14a
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total de linhas no arquivo
Total = 7
Process finished with exit code 0

script14b.py
1 with open("dados13.txt", "r") as arquivo:
2     print("Total REAL de linhas no arquivo")
3     contador = 0
4     for linha in arquivo:
5         if linha.strip():
6             contador += 1
7     print("Total real = ", contador)

Run: script14b
C:\Users\ftoli\PycharmProjects\EAD\venv\Script
Total REAL de linhas no arquivo
Total real = 3
Process finished with exit code 0
```

Escrita do mesmo código.

Escrevemos, no segundo script, praticamente o mesmo código, iniciando pela abertura do arquivo na linha 1, e assim por diante. A exceção é a linha 5, que testa o conteúdo da variável `linha` após utilização do método `strip()`. Observe que, agora, contamos corretamente o número de linhas com algum conteúdo, 3.

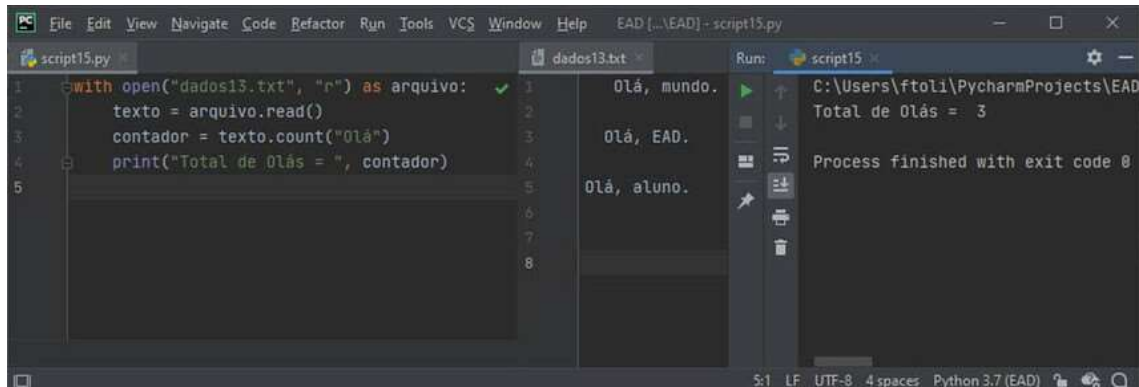
Métodos *count* e *split*

Outra atividade muito comum na manipulação de arquivos é a contagem do número de vezes que determinada palavra aparece. O Python disponibiliza o método `count` para strings, que recebe como parâmetro a palavra que desejamos contar e retorna o total de ocorrências dela. Sua sintaxe é a seguinte:

```
contagem = variavel_string.count(palavra)
```

Nela, a `variavel_string` é uma variável do tipo `str` e `palavra` é a string que desejamos contar.

Veja a seguir como utilizamos o método `count` para contar a quantidade de vezes em que aparece a palavra “Olá”, no arquivo `dados13.txt`:



The screenshot shows the PyCharm IDE with three panels. The left panel displays the code in `script15.py`:

```
1 with open("dados13.txt", "r") as arquivo:
2     texto = arquivo.read()
3     contador = texto.count("Olá")
4     print("Total de Olás = ", contador)
5
```

The middle panel shows the contents of `dados13.txt`:

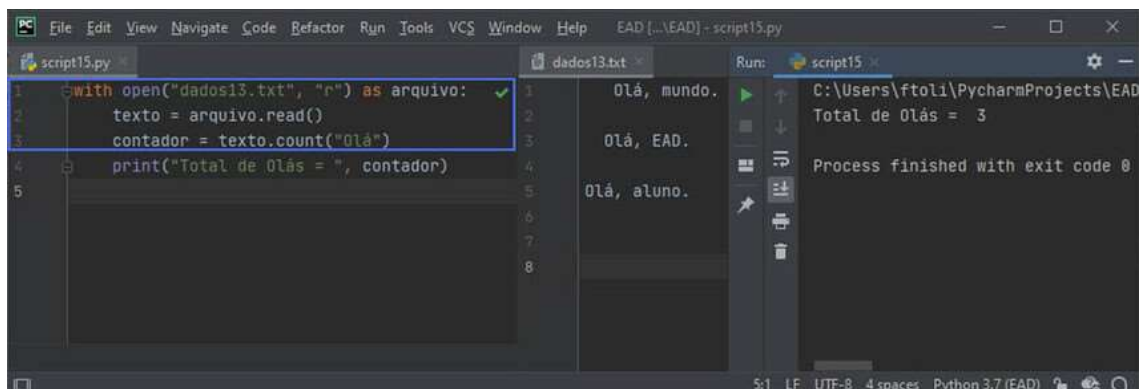
```
1 Olá, mundo.
2
3 Olá, EAD.
4
5 Olá, aluno.
6
7
8
```

The right panel shows the Run console output:

```
C:\Users\ftoli\PycharmProjects\EAD
Total de Olás = 3
Process finished with exit code 0
```

Scripts 15, saída e arquivo `dados13.txt`.

Temos os scripts 15, sua saída e arquivo `dados13.txt`.

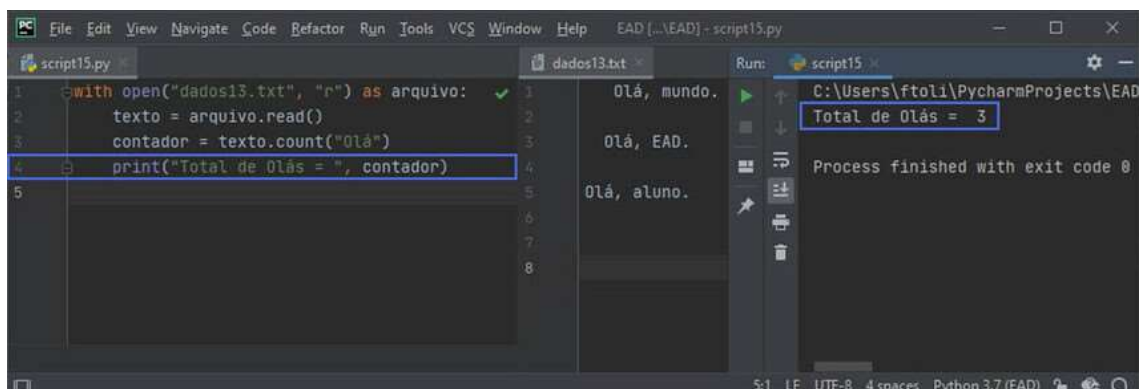


This screenshot is identical to the previous one, but with a blue box highlighting the first three lines of the Python code in `script15.py`:

```
1 with open("dados13.txt", "r") as arquivo:
2     texto = arquivo.read()
3     contador = texto.count("Olá")
```

Aplicação do método `read`.

Utilizamos, após abrir o arquivo na linha 1, o método `read`, linha 2, para retornar todo o conteúdo do arquivo como uma única string e armazená-lo na variável `texto`. Na linha 3 utilizamos o método `count` da variável `texto`, passando como argumento à string “Olá”.



This screenshot is identical to the previous one, but with a blue box highlighting the `print` statement in `script15.py`:

```
4     print("Total de Olás = ", contador)
```

Armazenamento do total de ocorrências de “Olá”.

O total de ocorrências da palavra “Olá” foi armazenado na variável **contador**, que foi impressa na linha 4. Observe que o total de ocorrência está correto: 3.

Apesar de ser muito simples a utilização do método *count* do tipo str, pode gerar alguns efeitos indesejáveis, pois esse método também conta as palavras que contêm parte da string passada como argumento.

Exemplo

Considere a frase: “Eu amo comer amoras no café da manhã”. Se utilizarmos o método *count*, com a string “amo” como argumento, o retorno será 2, pois o método irá considerar tanto a palavra amo quanto a palavra amoras.

Para contornar esse problema, podemos “quebrar” uma frase em palavras e depois verificar se cada palavra é igual à string que buscamos.

Isso nos leva a outro método muito utilizado em processamento de textos, o método *split()*, que é usado para quebrar uma string em partes menores, retornando uma lista com essas partes. Sua sintaxe é a seguinte:

```
lista_termos = variavel_string.split(separador)
```

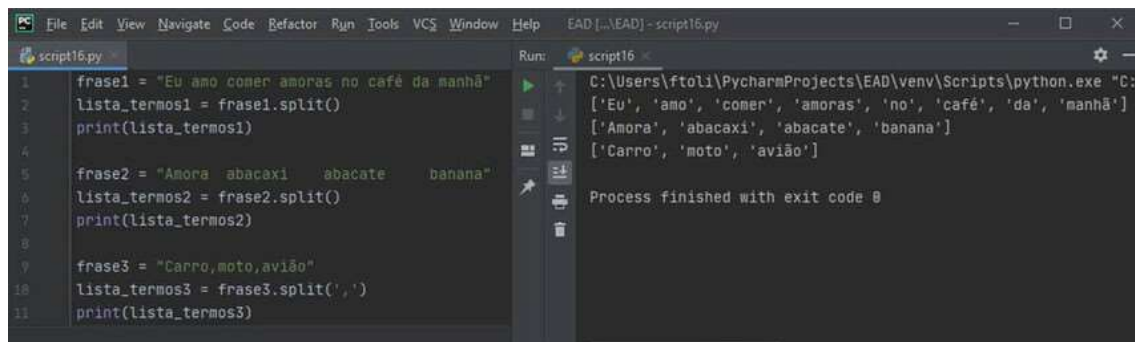
No exemplo, a *variavel_string* é uma variável do tipo str, e *separador* é uma string que desejamos utilizar como ponto de quebra do texto. O retorno desse método é uma lista de strings.

Digamos que desejamos usar o método *split* com separador ‘-’ na frase: “Amo futebol – Gosto de basquete”. O resultado seria uma lista em que o primeiro elemento é a string “Amo futebol ” e o segundo elemento é string “ Gosto de basquete”. Caso nenhum separador seja passado como argumento, o Python irá utilizar um ou mais espaços como separador padrão.

Dica

A string utilizada como separador **não** aparecerá em nenhum elemento da lista retornada.

No exemplo a seguir o método *split* é utilizado em três frases diferentes para mostrar o comportamento dele. Confira:

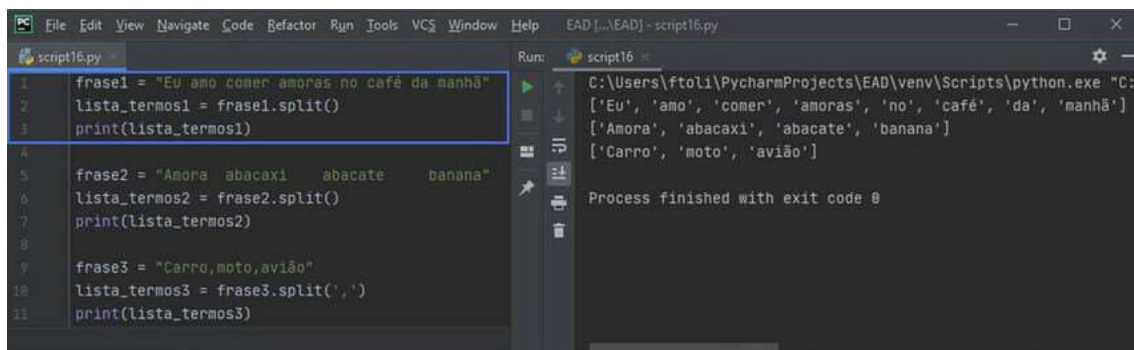


```
1 frase1 = "Eu amo comer amoras no café da manhã"
2 lista_termos1 = frase1.split()
3 print(lista_termos1)
4
5 frase2 = "Amora abacaxi abacate banana"
6 lista_termos2 = frase2.split()
7 print(lista_termos2)
8
9 frase3 = "Carro,moto,avião"
10 lista_termos3 = frase3.split(',')
11 print(lista_termos3)
```

The screenshot shows a Python script named `script16.py` in an IDE. The script defines three strings and splits them using the `split()` method. The first string is split by default (spaces), the second by default (spaces), and the third by commas. The output of the script is shown in the Run window, displaying three lists of words.

Script 16 e saída.

Temos o script 16 e sua saída.



```
1 frase1 = "Eu amo comer amoras no café da manhã"
2 lista_termos1 = frase1.split()
3 print(lista_termos1)
4
5 frase2 = "Amora abacaxi abacate banana"
6 lista_termos2 = frase2.split()
7 print(lista_termos2)
8
9 frase3 = "Carro,moto,avião"
10 lista_termos3 = frase3.split(',')
11 print(lista_termos3)
```

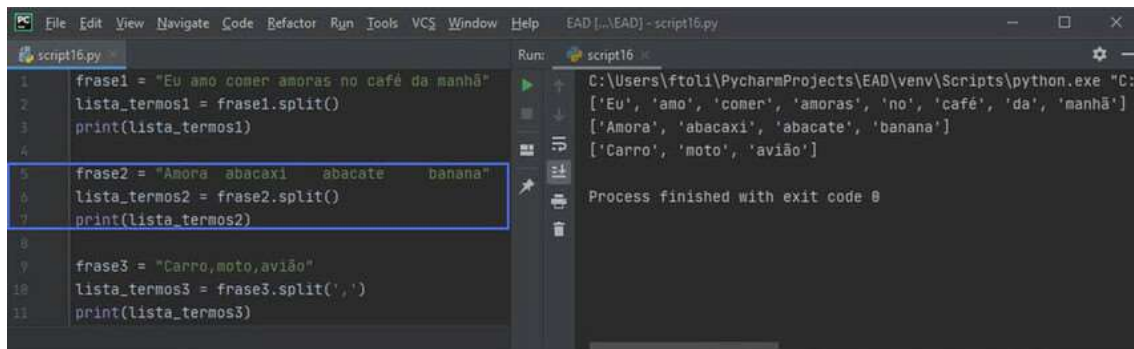
Run: script16

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/
['Eu', 'amo', 'comer', 'amoras', 'no', 'café', 'da', 'manhã']
['Amora', 'abacaxi', 'abacate', 'banana']
['Carro', 'moto', 'avião']

Process finished with exit code 0
```

Aplicação do método *split*.

Utilizamos o método *split* sem argumentos, linha 2, e imprimimos a lista retornada na linha 3. Observe pelo console que cada item da lista é uma palavra da frase.



```
1 frase1 = "Eu amo comer amoras no café da manhã"
2 lista_termos1 = frase1.split()
3 print(lista_termos1)
4
5 frase2 = "Amora abacaxi abacate banana"
6 lista_termos2 = frase2.split()
7 print(lista_termos2)
8
9 frase3 = "Carro,moto,avião"
10 lista_termos3 = frase3.split(',')
11 print(lista_termos3)
```

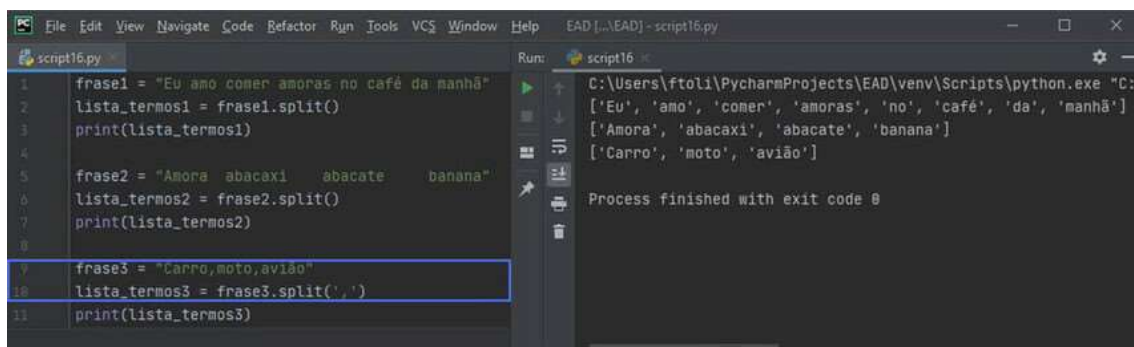
Run: script16

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/
['Eu', 'amo', 'comer', 'amoras', 'no', 'café', 'da', 'manhã']
['Amora', 'abacaxi', 'abacate', 'banana']
['Carro', 'moto', 'avião']

Process finished with exit code 0
```

Aplicação do método *split* e impressão da lista.

Utilizamos o método *split*, também, sem argumentos, linha 6, e imprimimos a lista retornada na linha 7. Observe pelo console que, mesmo havendo muitos espaços em branco entre as palavras da frase, a lista contém apenas as palavras, sem nenhum espaço adicional. O Python é esperto o suficiente para detectar vários espaços contínuos e tratá-los como um único separador.



```
1 frase1 = "Eu amo comer amoras no café da manhã"
2 lista_termos1 = frase1.split()
3 print(lista_termos1)
4
5 frase2 = "Amora abacaxi abacate banana"
6 lista_termos2 = frase2.split()
7 print(lista_termos2)
8
9 frase3 = "Carro,moto,avião"
10 lista_termos3 = frase3.split(',')
11 print(lista_termos3)
```

Run: script16

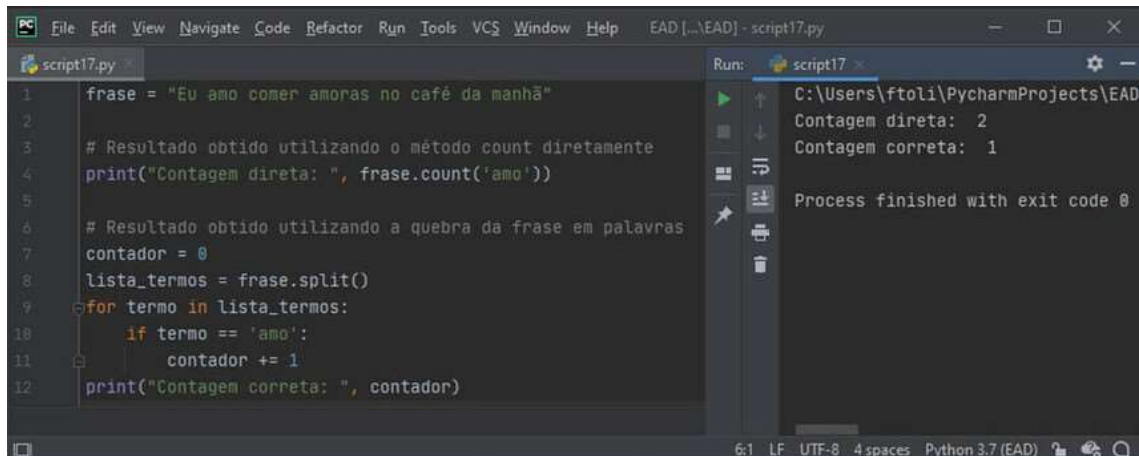
```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/
['Eu', 'amo', 'comer', 'amoras', 'no', 'café', 'da', 'manhã']
['Amora', 'abacaxi', 'abacate', 'banana']
['Carro', 'moto', 'avião']

Process finished with exit code 0
```

Aplicação do método *split* passando uma vírgula (,).

Temos, na linha 9, a última frase: "Carro,moto,avião". Desta vez, utilizamos o método *split* passando uma vírgula (,) como argumento, na linha 10. O resultado é uma lista contendo apenas as palavras, **sem o separador**, conforme podemos ver pelo console.

Agora que sabemos como funciona o método *split*, no próximo exemplo, vamos utilizar esse método para realizar a contagem da palavra "amo" na frase do exemplo anterior: "Eu amo comer amoras no café da manhã.". Vamos aproveitar para comparar os resultados obtidos pelos métodos *count* e *split*.



The screenshot shows the PyCharm IDE with a file named `script17.py`. The code in the editor is as follows:

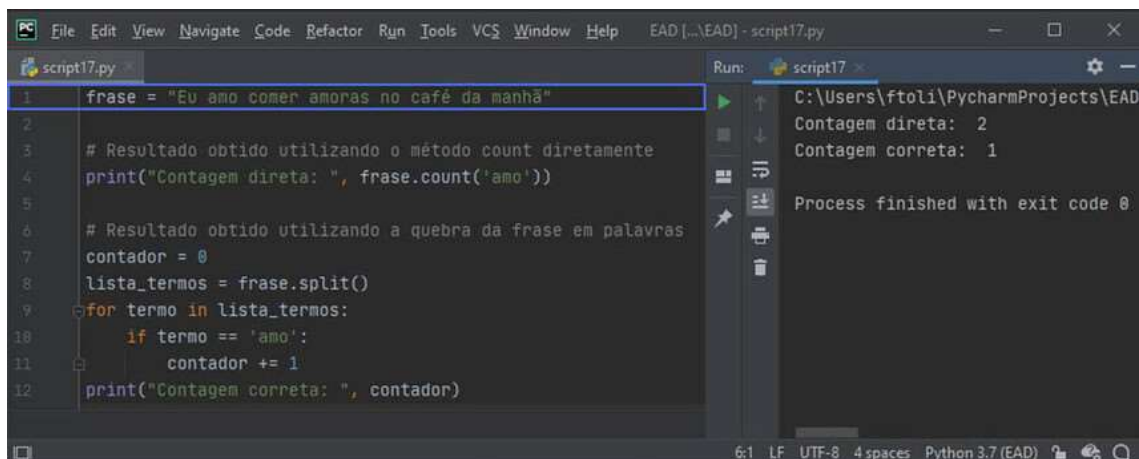
```
1 frase = "Eu amo comer amoras no café da manhã"
2
3 # Resultado obtido utilizando o método count diretamente
4 print("Contagem direta: ", frase.count('amo'))
5
6 # Resultado obtido utilizando a quebra da frase em palavras
7 contador = 0
8 lista_termos = frase.split()
9 for termo in lista_termos:
10     if termo == 'amo':
11         contador += 1
12 print("Contagem correta: ", contador)
```

The Run window on the right shows the output of the script:

```
C:\Users\ftoli\PycharmProjects\EAD
Contagem direta: 2
Contagem correta: 1
Process finished with exit code 0
```

Script 17 e saída.

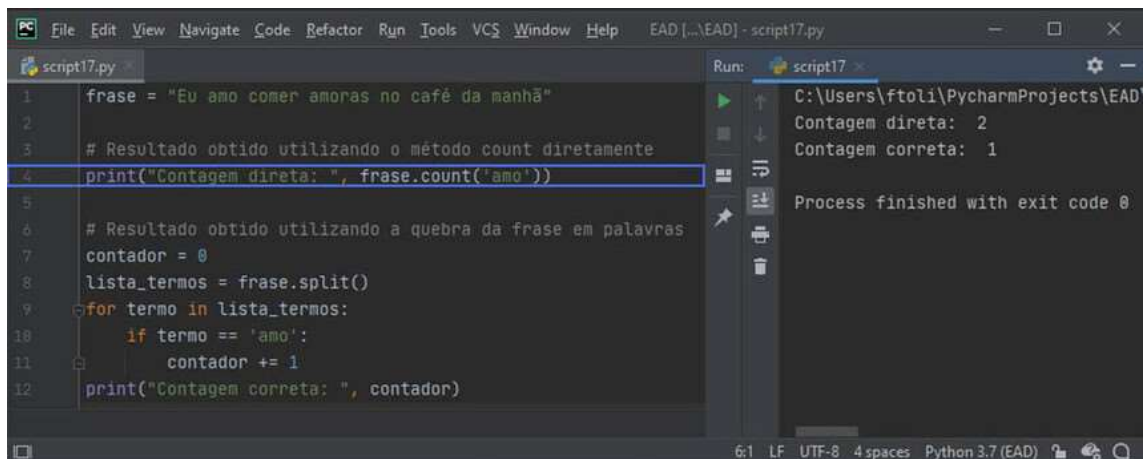
Temos o script 17 e sua saída.



This screenshot is similar to the previous one, but the first line of the code, `frase = "Eu amo comer amoras no café da manhã"`, is highlighted with a blue selection bar.

Definição da variável frase.

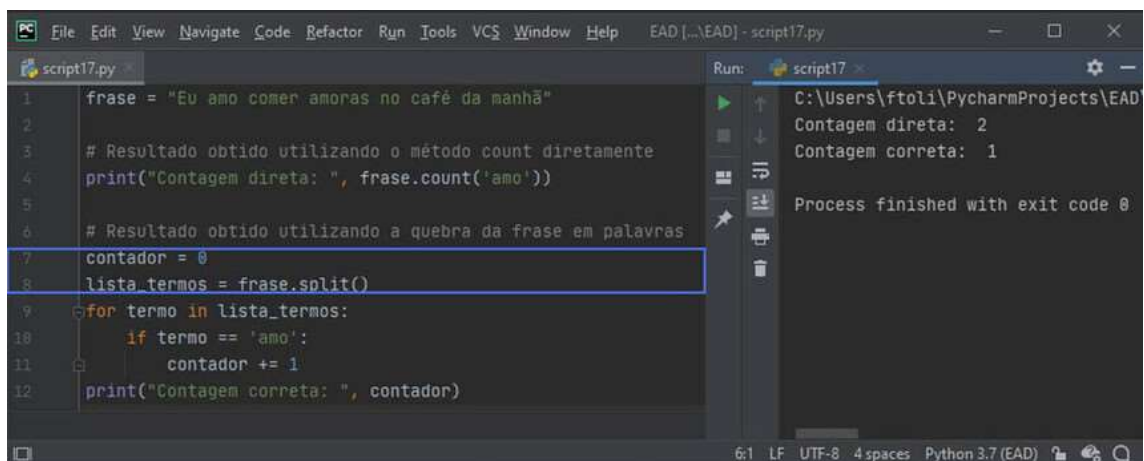
Definimos, na linha 1, nossa variável **frase** com o conteúdo descrito anteriormente.



The screenshot shows the PyCharm IDE with a Python script named `script17.py`. The script defines a string `frase = "Eu amo comer amoras no café da manhã"`. Line 4, which is highlighted with a blue selection box, uses the `count` method: `print("Contagem direta: ", frase.count('amo'))`. The right-hand pane shows the output of the script: `Contagem direta: 2` and `Contagem correta: 1`, followed by `Process finished with exit code 0`. The status bar at the bottom indicates the file encoding is UTF-8 and the Python version is 3.7.

Aplicação do método *count*.

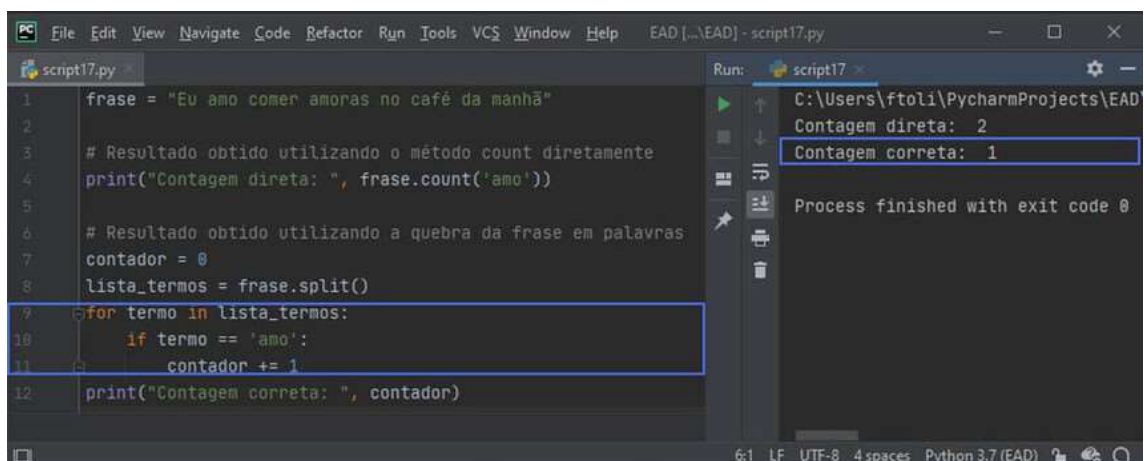
Utilizamos, na linha 4, o método *count* do tipo *str* para contar diretamente o número de ocorrências da string “amo” na frase. Pelo console, vemos que o resultado da contagem foi 2. Uma ocorrência pela palavra *amo* e outra pela palavra **amora**.



This screenshot shows the same PyCharm IDE environment. The script is identical to the previous one, but line 7, `contador = 0`, and line 8, `lista_termos = frase.split()`, are highlighted with a blue selection box. The output in the console remains the same: `Contagem direta: 2` and `Contagem correta: 1`.

Criação de um contador e aplicação do método *split*.

Criamos, para realizar a contagem da mesma frase usando o *split*, um contador, linha 7, e quebramos a frase em palavras utilizando o método *split*, linha 8.



This screenshot shows the PyCharm IDE with the script from the previous image. Lines 9 through 12 are highlighted with a blue selection box. These lines contain a `for` loop that iterates over the words in `lista_termos` (created by `split`), checks if the word is `'amo'`, increments the `contador`, and finally prints the result: `print("Contagem correta: ", contador)`. The console output is the same as in the previous image.

Iteração sobre cada palavra.

Iteramos, na linha 9, sobre cada palavra utilizando a variável **termo**. Para cada termo, comparamos se seu valor é igual a string “amo”, linha 10. Caso seja igual, incrementamos o contador, linha 11. Observe que desta vez alcançamos o resultado correto para a contagem, 1.

No script17.py, para realizar a contagem da palavra “amo” na frase “Eu amo comer amoras no café da manhã”, foi criada uma lista chamada `lista_termos`, através do uso do método *split* sobre a variável que continha a frase mencionada. Também foi criado um contador e houve uma iteração sobre `lista_termos`, realizando comparações entre os elementos desta e a palavra “amo”.

Será que não haveria uma maneira de realizar essa contagem de outra maneira, fazendo uso dos métodos de manipulação de strings apresentados até aqui?

Utilize o emulador de códigos para desenvolver outra solução para o problema da contagem da palavra “amo” na frase “Eu amo comer amoras no café da manhã.”:

```
frase = "Eu amo comer amoras no café da manhã."
```

```
lista_termos = frase.split()
```

A resolução está contida a seguir:

```
frase = "Eu amo comer amoras no café da manhã."
```

```
lista_termos = frase.split()
```

```
contagem = lista_termos.count("amo")
```

```
print("Contagem = ", contagem)
```

No código acima, na linha 2, temos a separação das palavras da frase por meio do uso do método *split* na variável `frase` (tipo `string`). A seguir, na linha 3, utilizamos o método *count* com o argumento “amo”, para obtermos a quantidade de ocorrências dessa string (“amo”) em `lista_termos`. Por fim, na linha 4, imprimimos o resultado no console do emulador.

Vamos avançar para mais um método!

Método *join*

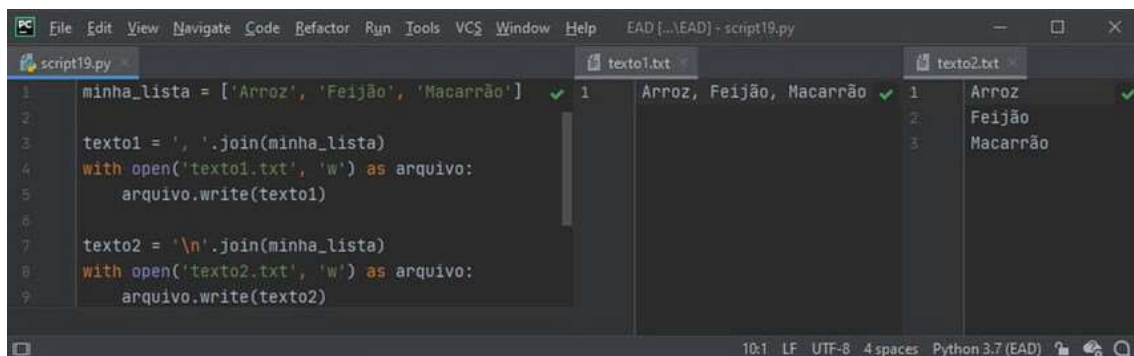
Assim como há a necessidade de quebrar uma frase em uma lista de palavras, podem existir situações em que ocorra o inverso, ou seja, temos uma lista de palavras ou frases e desejamos concatená-las em uma única string.

Para essa operação, utilizamos o método `join` do tipo `str`, que retorna uma única string com todos os elementos da lista concatenados, utilizando determinado conector. Sua sintaxe é a seguinte:

```
string_final = "conector".join(iteravel)
```

Em que “conector” é a string que será utilizada entre os elementos da lista que serão concatenados (ex. `,` `'`) e iteravel é um iterável, como uma lista ou tupla.

No exemplo a seguir, vamos unir os elementos de uma lista utilizando dois conectores diferentes: o conector vírgula (`,`) e o conector de nova linha (`\n`). Após a união, vamos gravar o conteúdo em um arquivo para mostrar o resultado:

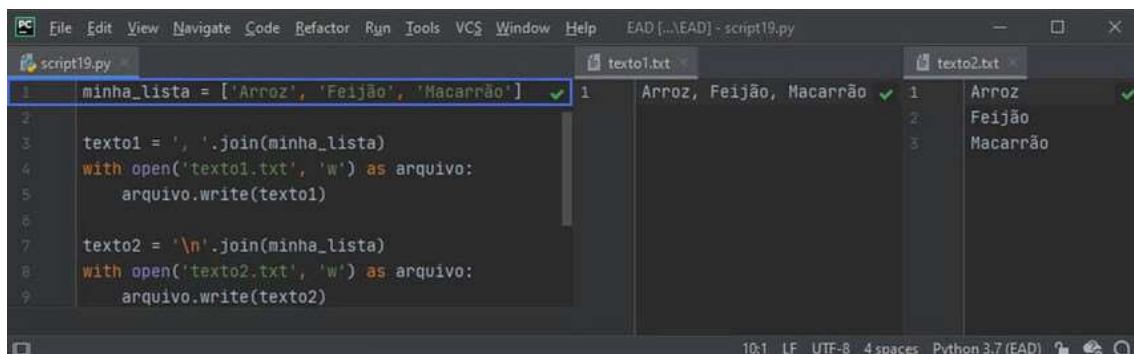


The screenshot shows a code editor with three files: `script19.py`, `texto1.txt`, and `texto2.txt`. The script in `script19.py` defines a list `minha_lista` with the elements `'Arroz'`, `'Feijão'`, and `'Macarrão'`. It then uses `join` with a comma as a separator to create `texto1`, and with a newline character as a separator to create `texto2`. Both strings are written to their respective text files. The output in `texto1.txt` is `Arroz, Feijão, Macarrão` and in `texto2.txt` is `Arroz`, `Feijão`, and `Macarrão` on separate lines.

```
1 minha_lista = ['Arroz', 'Feijão', 'Macarrão']
2
3 texto1 = ', '.join(minha_lista)
4 with open('texto1.txt', 'w') as arquivo:
5     arquivo.write(texto1)
6
7 texto2 = '\n'.join(minha_lista)
8 with open('texto2.txt', 'w') as arquivo:
9     arquivo.write(texto2)
```

Script 19, arquivos `texto1.txt` e `texto2.txt`.

Temos o script 19, arquivos `texto1.txt` e `texto2.txt`.



This screenshot is similar to the previous one but highlights the first line of the script, `minha_lista = ['Arroz', 'Feijão', 'Macarrão']`, which is the line where the list is created.

```
1 minha_lista = ['Arroz', 'Feijão', 'Macarrão']
2
3 texto1 = ', '.join(minha_lista)
4 with open('texto1.txt', 'w') as arquivo:
5     arquivo.write(texto1)
6
7 texto2 = '\n'.join(minha_lista)
8 with open('texto2.txt', 'w') as arquivo:
9     arquivo.write(texto2)
```

Criação da lista `minha_lista`.

Criamos, na linha 1, a lista `minha_lista`, que será utilizada nos dois exemplos.

```
1 minha_lista = ['Arroz', 'Feijão', 'Macarrão']
2
3 texto1 = ', '.join(minha_lista)
4 with open('texto1.txt', 'w') as arquivo:
5     arquivo.write(texto1)
6
7 texto2 = '\n'.join(minha_lista)
8 with open('texto2.txt', 'w') as arquivo:
9     arquivo.write(texto2)
```

texto1.txt

```
1 Arroz, Feijão, Macarrão
```

texto2.txt

```
1 Arroz
2 Feijão
3 Macarrão
```

Aplicação do método *join*.

Utilizamos, na linha 3, o método *join* com o conector `,` e atribuímos o resultado à variável **texto1**. O resultado dessa variável foi gravado no arquivo **texto1.txt**, que pode ser visto na imagem.

```
1 minha_lista = ['Arroz', 'Feijão', 'Macarrão']
2
3 texto1 = ', '.join(minha_lista)
4 with open('texto1.txt', 'w') as arquivo:
5     arquivo.write(texto1)
6
7 texto2 = '\n'.join(minha_lista)
8 with open('texto2.txt', 'w') as arquivo:
9     arquivo.write(texto2)
```

texto1.txt

```
1 Arroz, Feijão, Macarrão
```

texto2.txt

```
1 Arroz
2 Feijão
3 Macarrão
```

Junção dos elementos da mesma lista.

Fazemos, na linha 7, a junção dos elementos da mesma lista utilizando o conector `\n`. O resultado da junção foi gravado no arquivo **texto2.txt**, que pode ser visto à direita da imagem. Com isso, fomos capazes de colocar cada elemento da lista em uma linha do arquivo.

Formatação de strings

Manipulação de variáveis em strings

Até o momento, realizamos operações sobre strings pré-existent. No entanto, é muito comum precisarmos juntar valores de variáveis com strings.

Agora, veremos algumas funções relacionadas às strings, começando pela formatação de strings (*string formatting*).

A formatação de strings permite ajustar como uma string será exibida ou gravada em um arquivo, por exemplo. Ajustes como: número de casas decimais em *float*, exibição de datas, inclusão de variáveis e espaçamento são alguns dos exemplos.

Existem basicamente três formas de realizar a formatação de strings. São elas:

- Utilizando f-strings (*formatted string literals*).
- Utilizando o método `format()` das strings.
- Fazendo manualmente.

Veremos como utilizar f-strings, recurso adicionado na versão 3.6 do Python.

F-strings

O objetivo principal da criação das f-strings foi facilitar a formatação de strings.

Para definimos uma variável f-string, precisamos incluir, antes das aspas que definem uma string, a letra f (ou F), por exemplo:

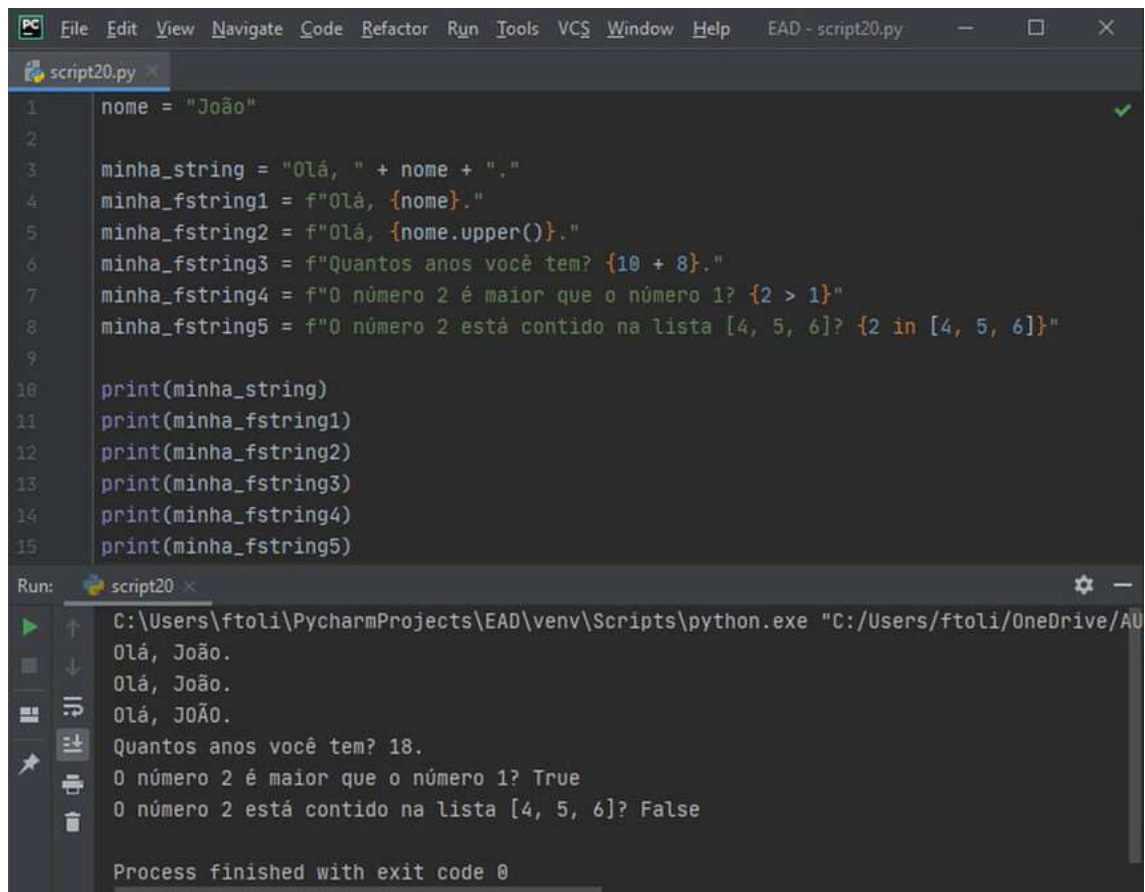
```
minha_string = f"Olá Mundo {expr}"
```

Dentro das f-string, podemos utilizar expressões em Python entre as chaves, delimitadas pelos caracteres `{` e `}`.

Essas expressões incluem variáveis ou literais. Inclusive, podemos fazer chamada para funções ou utilizar métodos de variáveis dentro desses delimitadores.

Todo o conteúdo entre os caracteres `{` e `}` é substituído pelo resultado da expressão e interpolado à string.

Compare a seguir a manipulação manual de strings com a utilização de f-string e veja alguns exemplos de uso de literais mais sofisticados dentro de f-strings:



The image shows a PyCharm IDE window with a Python script named `script20.py` and its execution output.

Script Code:

```
1 nome = "João"
2
3 minha_string = "Olá, " + nome + "."
4 minha_fstring1 = f"Olá, {nome}."
5 minha_fstring2 = f"Olá, {nome.upper()}."
6 minha_fstring3 = f"Quantos anos você tem? {10 + 8}."
7 minha_fstring4 = f"O número 2 é maior que o número 1? {2 > 1}"
8 minha_fstring5 = f"O número 2 está contido na lista [4, 5, 6]? {2 in [4, 5, 6]}"
9
10 print(minha_string)
11 print(minha_fstring1)
12 print(minha_fstring2)
13 print(minha_fstring3)
14 print(minha_fstring4)
15 print(minha_fstring5)
```

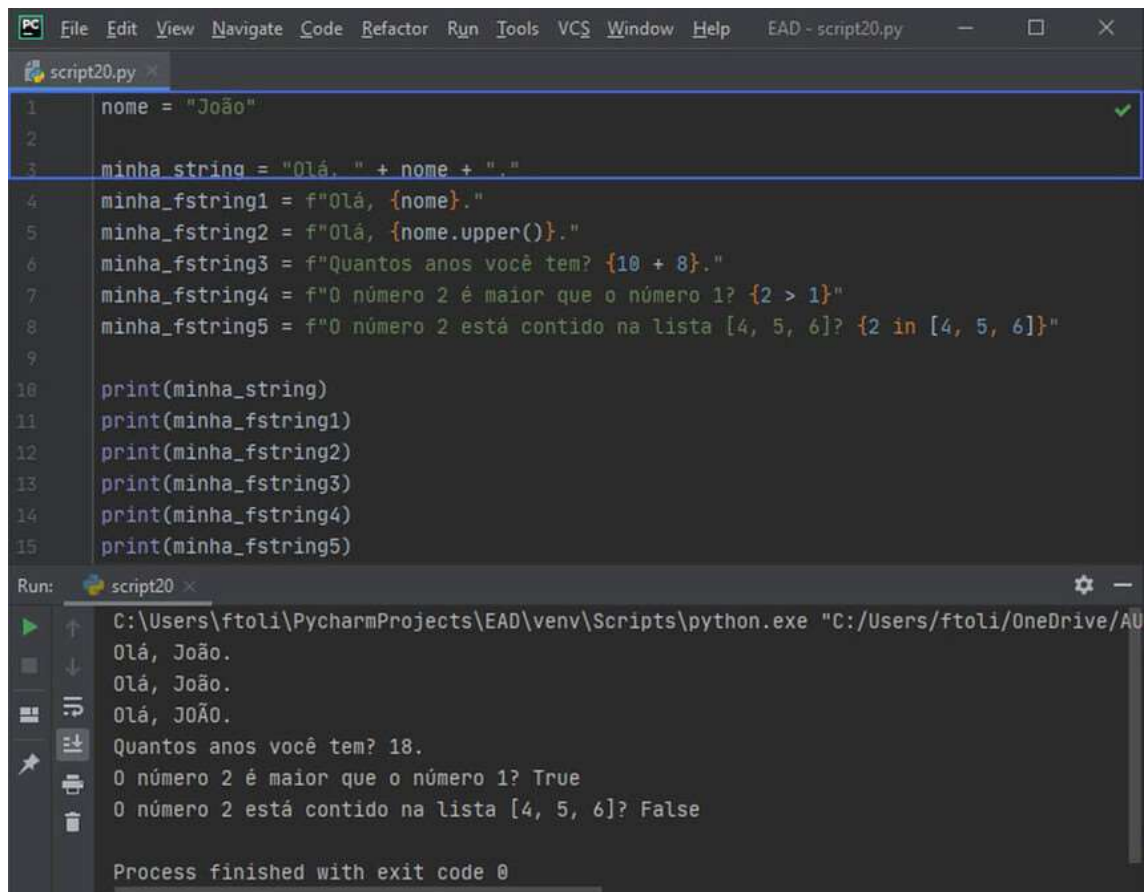
Run Output:

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/OneDrive/AU
Olá, João.
Olá, João.
Olá, JOÃO.
Quantos anos você tem? 18.
O número 2 é maior que o número 1? True
O número 2 está contido na lista [4, 5, 6]? False

Process finished with exit code 0
```

Script 20 e saída.

Temos o script 20 e sua saída.



The image shows a PyCharm IDE window with a Python script named `script20.py`. The script defines a variable `nome` and uses it in several string operations. The output window shows the results of these operations.

```
1 nome = "João"
2
3 minha_string = "Olá. " + nome + "."
4 minha_fstring1 = f"Olá, {nome}."
5 minha_fstring2 = f"Olá, {nome.upper()}."
6 minha_fstring3 = f"Quantos anos você tem? {10 + 8}."
7 minha_fstring4 = f"O número 2 é maior que o número 1? {2 > 1}"
8 minha_fstring5 = f"O número 2 está contido na lista [4, 5, 6]? {2 in [4, 5, 6]}"
9
10 print(minha_string)
11 print(minha_fstring1)
12 print(minha_fstring2)
13 print(minha_fstring3)
14 print(minha_fstring4)
15 print(minha_fstring5)
```

Run: `script20`

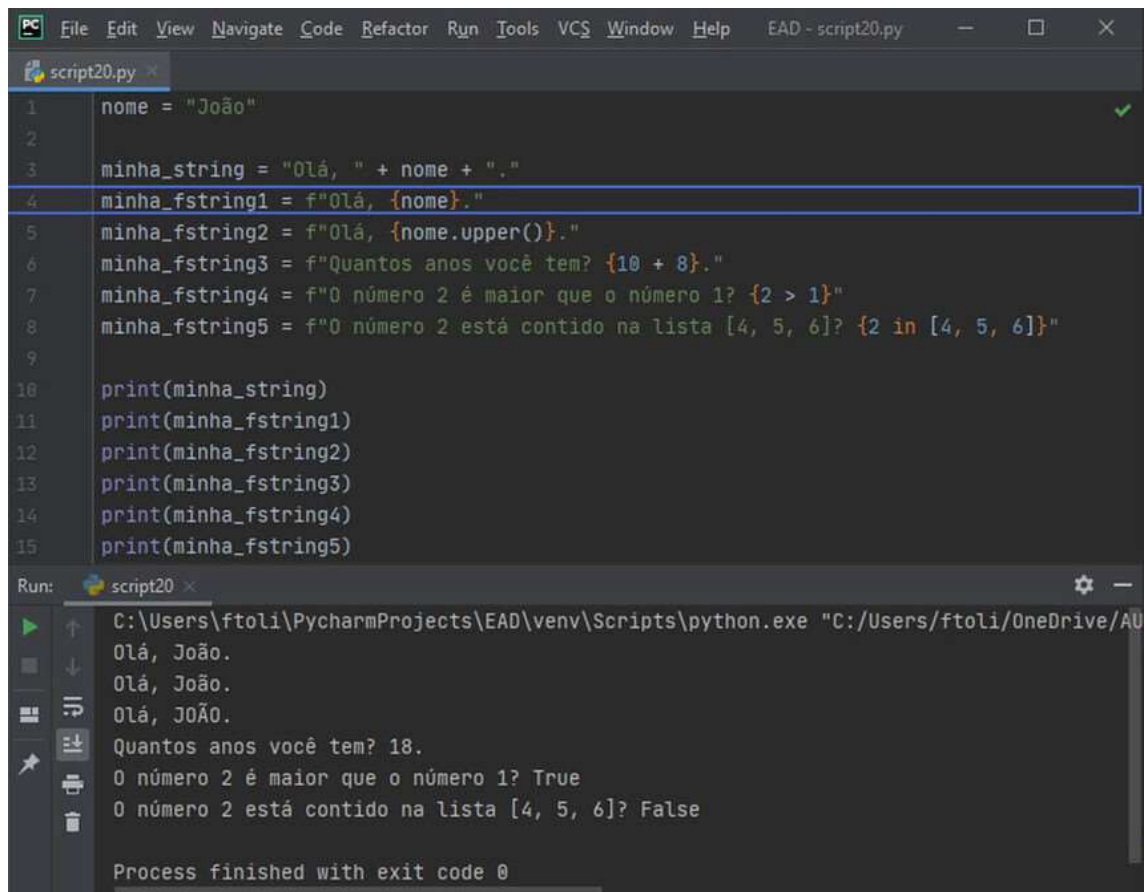
```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/OneDrive/AU
Olá, João.
Olá, João.
Olá, JOÃO.
Quantos anos você tem? 18.
O número 2 é maior que o número 1? True
O número 2 está contido na lista [4, 5, 6]? False

Process finished with exit code 0
```

Definição da variável `nome`.

Definimos, na linha 1, a variável **nome**, do tipo string, que será utilizada ao longo do script.

Mostramos, na linha 3, como incluir o valor da variável **nome** no meio de outra string, utilizando o processo manual de concatenação de strings por meio do operador '+'.
A linha 4 utiliza a formatação de strings com f-strings para incluir o valor da variável `nome` diretamente na string.



The image shows a PyCharm IDE window with a Python script named `script20.py`. The script defines a variable `nome` with the value "João" and creates several strings and f-strings. Line 4, `minha_fstring1 = f"Olá, {nome}."`, is highlighted. The script then prints each of these strings. Below the editor, the 'Run' console shows the output of the script, demonstrating how the f-string correctly inserts the value of `nome` into the string.

```
1 nome = "João"
2
3 minha_string = "Olá, " + nome + "."
4 minha_fstring1 = f"Olá, {nome}."
5 minha_fstring2 = f"Olá, {nome.upper()}."
6 minha_fstring3 = f"Quantos anos você tem? {10 + 8}."
7 minha_fstring4 = f"O número 2 é maior que o número 1? {2 > 1}"
8 minha_fstring5 = f"O número 2 está contido na lista [4, 5, 6]? {2 in [4, 5, 6]}"
9
10 print(minha_string)
11 print(minha_fstring1)
12 print(minha_fstring2)
13 print(minha_fstring3)
14 print(minha_fstring4)
15 print(minha_fstring5)
```

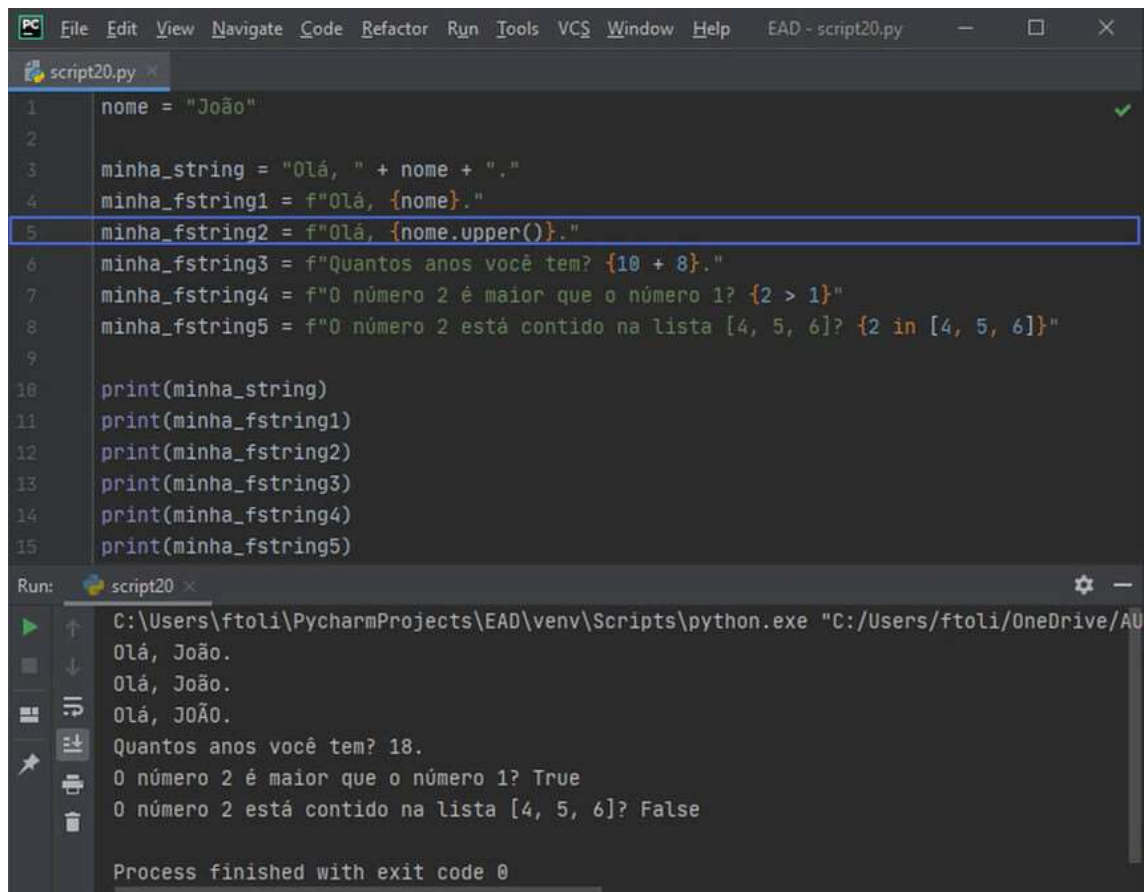
Run: `C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/OneDrive/AU`

```
Olá, João.
Olá, João.
Olá, JOÃO.
Quantos anos você tem? 18.
O número 2 é maior que o número 1? True
O número 2 está contido na lista [4, 5, 6]? False

Process finished with exit code 0
```

Aplicação da sintaxe f-string.

Utilizamos, na linha 4, a sintaxe da f-string para incluir o valor da variável `nome` também no meio de outra string. Observe que a IDE PyCharm já detectou que temos uma expressão entre as chaves e alterou a cor para destacar esse elemento. No console, o resultado das variáveis `minha_string` e `minha_fstring1` foi o mesmo, porém a sintaxe da f-string é muito mais clara e simples de ser utilizada e entendida.



The image shows a PyCharm IDE window with a Python script named `script20.py`. The script defines a variable `nome` with the value "João" and creates five f-strings. The fifth f-string, `minha_fstring2 = f"Olá, {nome.upper()}."`, is highlighted with a blue selection bar. Below the editor, the 'Run' console shows the output of the script. The output displays the first three f-strings, with the third one showing "Olá, JOÃO." in all caps. It also shows the results of two conditional checks: "0 número 2 é maior que o número 1? True" and "0 número 2 está contido na lista [4, 5, 6]? False". The console ends with the message "Process finished with exit code 0".

```
1 nome = "João"
2
3 minha_string = "Olá, " + nome + "."
4 minha_fstring1 = f"Olá, {nome}."
5 minha_fstring2 = f"Olá, {nome.upper()}."
6 minha_fstring3 = f"Quantos anos você tem? {10 + 8}."
7 minha_fstring4 = f"0 número 2 é maior que o número 1? {2 > 1}"
8 minha_fstring5 = f"0 número 2 está contido na lista [4, 5, 6]? {2 in [4, 5, 6]}"
9
10 print(minha_string)
11 print(minha_fstring1)
12 print(minha_fstring2)
13 print(minha_fstring3)
14 print(minha_fstring4)
15 print(minha_fstring5)
```

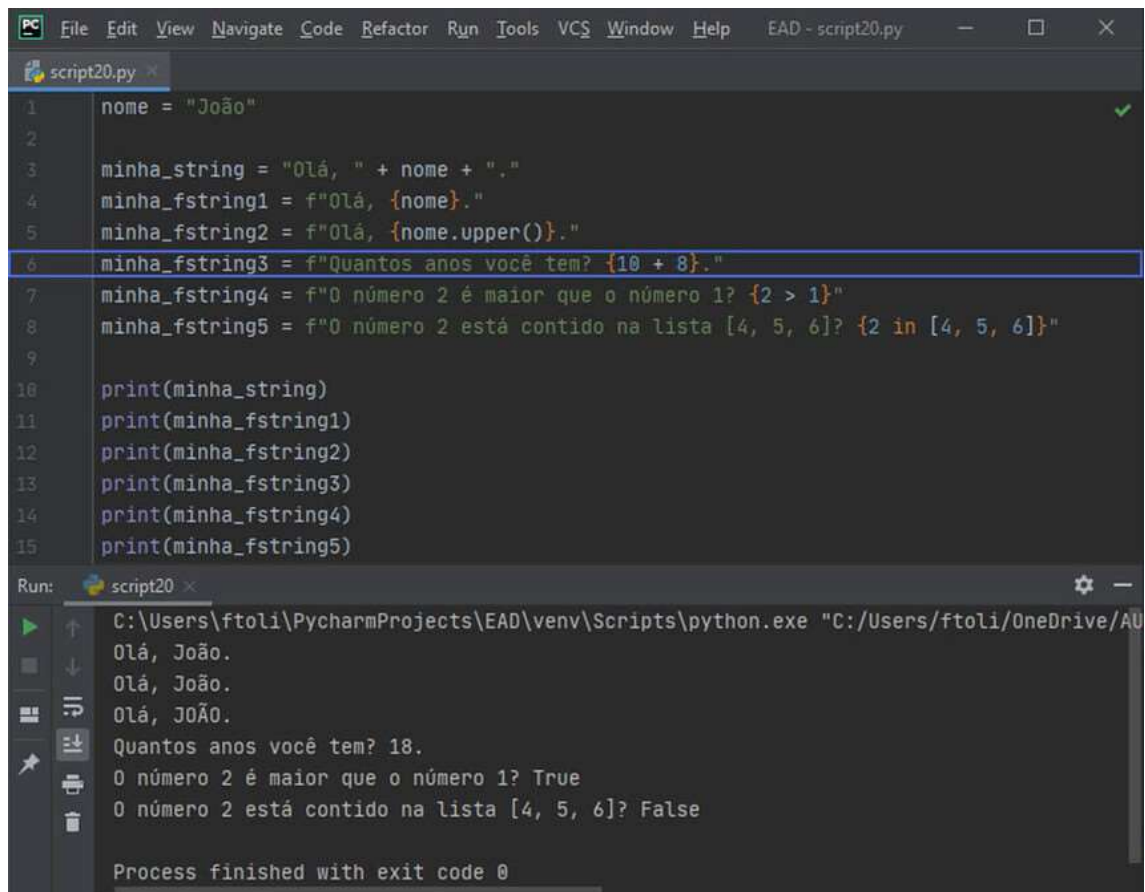
Run: script20 ×

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/OneDrive/AU
Olá, João.
Olá, João.
Olá, JOÃO.
Quantos anos você tem? 18.
0 número 2 é maior que o número 1? True
0 número 2 está contido na lista [4, 5, 6]? False

Process finished with exit code 0
```

Aplicação do método *upper*.

Temos, na linha 5, o mesmo exemplo da linha anterior, porém chamamos o método *upper* do tipo str para colocar todas as letras em maiúsculo. Isso foi feito diretamente na expressão!



The image shows a PyCharm IDE window with a Python script named `script20.py`. The script defines a variable `nome` with the value "João" and creates five f-strings. The third f-string, `minha_fstring3`, includes an arithmetic expression `{10 + 8}` inside curly braces. The script then prints each of these strings. Below the editor, the 'Run' console shows the output of the script, confirming that the expression `10 + 8` was correctly evaluated to 18.

```
1 nome = "João"
2
3 minha_string = "Olá, " + nome + "."
4 minha_fstring1 = f"Olá, {nome}."
5 minha_fstring2 = f"Olá, {nome.upper()}."
6 minha_fstring3 = f"Quantos anos você tem? {10 + 8}."
7 minha_fstring4 = f"O número 2 é maior que o número 1? {2 > 1}"
8 minha_fstring5 = f"O número 2 está contido na lista [4, 5, 6]? {2 in [4, 5, 6]}"
9
10 print(minha_string)
11 print(minha_fstring1)
12 print(minha_fstring2)
13 print(minha_fstring3)
14 print(minha_fstring4)
15 print(minha_fstring5)
```

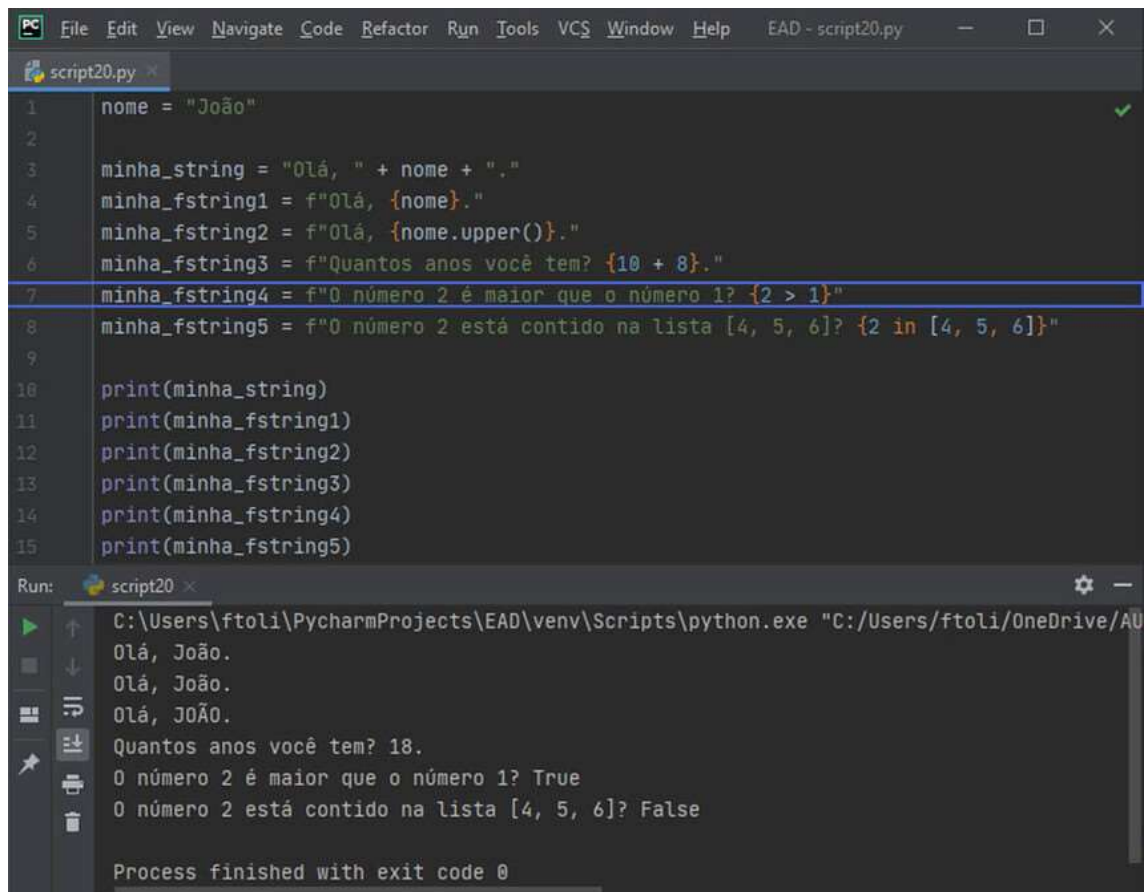
Run: script20 ×

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/OneDrive/AU
Olá, João.
Olá, João.
Olá, JOÃO.
Quantos anos você tem? 18.
O número 2 é maior que o número 1? True
O número 2 está contido na lista [4, 5, 6]? False

Process finished with exit code 0
```

Aplicação de soma na expressão entre chaves.

Utilizamos, na linha 6, uma soma na expressão entre chaves, que foi calculada corretamente, conforme console.



The image shows a PyCharm IDE window with a Python script named `script20.py`. The script defines a variable `nome` with the value "João" and creates five f-strings. Line 7, which contains the boolean comparison `{2 > 1}`, is highlighted with a blue line. Below the editor, the 'Run' console shows the output of the script, including the result of the boolean comparison: `0 número 2 é maior que o número 1? True`. The process finished with exit code 0.

```
1 nome = "João"
2
3 minha_string = "Olá, " + nome + "."
4 minha_fstring1 = f"Olá, {nome}."
5 minha_fstring2 = f"Olá, {nome.upper()}."
6 minha_fstring3 = f"Quantos anos você tem? {10 + 8}."
7 minha_fstring4 = f"0 número 2 é maior que o número 1? {2 > 1}"
8 minha_fstring5 = f"0 número 2 está contido na lista [4, 5, 6]? {2 in [4, 5, 6]}"
9
10 print(minha_string)
11 print(minha_fstring1)
12 print(minha_fstring2)
13 print(minha_fstring3)
14 print(minha_fstring4)
15 print(minha_fstring5)
```

Run: script20 ×

```
C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/OneDrive/AU
Olá, João.
Olá, João.
Olá, JOÃO.
Quantos anos você tem? 18.
0 número 2 é maior que o número 1? True
0 número 2 está contido na lista [4, 5, 6]? False

Process finished with exit code 0
```

Aplicação do comparador booleano >.

Utilizamos, na linha 7, o comparador booleano >, que também foi avaliado corretamente.

```

1 nome = "João"
2
3 minha_string = "Olá, " + nome + "."
4 minha_fstring1 = f"Olá, {nome}."
5 minha_fstring2 = f"Olá, {nome.upper()}."
6 minha_fstring3 = f"Quantos anos você tem? {10 + 8}."
7 minha_fstring4 = f"O número 2 é maior que o número 1? {2 > 1}"
8 minha_fstring5 = f"O número 2 está contido na lista [4, 5, 6]? {2 in [4, 5, 6]}"
9
10 print(minha_string)
11 print(minha_fstring1)
12 print(minha_fstring2)
13 print(minha_fstring3)
14 print(minha_fstring4)
15 print(minha_fstring5)

```

Run: script20

```

C:\Users\ftoli\PycharmProjects\EAD\venv\Scripts\python.exe "C:/Users/ftoli/OneDrive/AU
Olá, João.
Olá, João.
Olá, JOÃO.
Quantos anos você tem? 18.
O número 2 é maior que o número 1? True
O número 2 está contido na lista [4, 5, 6]? False

Process finished with exit code 0

```

Aplicação do operador IN.

Utilizamos, na linha 8, o operador IN para verificar a pertinência de um número em uma lista, que, mais uma vez, foi avaliado corretamente. Observe como é fácil e intuitiva a utilização de f-string!

Veja agora algumas funcionalidades adicionais de formatação de string utilizando f-string, como a definição de largura de uma string, formatação de float e de datas:

```

1 from datetime import datetime
2
3 frutas = ["Jabuticaba", "Laranja", "Uva", "Banana"]
4 for fruta in frutas:
5     minha_fruta = f"Nome: {fruta:12} - Número de letras: {len(fruta): 3}"
6     print(minha_fruta)
7
8 print()
9
10 pi = 3.1415
11 meu_numero = f"O número PI é: {pi:1f}"
12 meu_numero_deslocado = f"O número PI deslocado é: {pi:6.1f}"
13 meu_numero_preciso = f"O número PI mais preciso é: {pi:.4f}"
14 print(meu_numero)
15 print(meu_numero_deslocado)
16 print(meu_numero_preciso)
17
18 print()
19
20 data = datetime.now()
21 minha_data = f"A data de hoje é {data}"
22 minha_data_formatada = f"A data de hoje formatada é {data:%d/%m/%Y}"
23 print(minha_data)
24 print(minha_data_formatada)

```

Run: script21

```

"C:\Users\ftoli\OneDrive\AULAS\PYTHON\Tema Car
Nome: Jabuticaba - Número de letras: 10
Nome: Laranja - Número de letras: 7
Nome: Uva - Número de letras: 3
Nome: Banana - Número de letras: 6

O número PI é: 3.1
O número PI deslocado é: 3.1
O número PI mais preciso é: 3.1415

A data de hoje é 2020-08-13 18:58:32.242817
A data de hoje formatada é 13/08/2020

Process finished with exit code 0

```

Script 21 e sua saída.


Para facilitar o entendimento, o script21 foi dividido em três trechos. O primeiro trecho, entre as linhas 3 e 6, trata da formatação de largura do conteúdo de expressões, que serve, principalmente, para alinhar conteúdo de texto. No segundo trecho, das linhas 10 a 16, mostramos como formatar floats. No terceiro trecho, das linhas 20 a 24, mostramos como formatar datas. Confira cada trecho citado:

Linhas 3 até 6

No primeiro trecho, definimos uma lista chamada **frutas** na linha 3 e, na linha 4, percorremos cada item dessa lista.

Para cada item, montamos a f-string `minha_fruta`, que contém o nome da fruta e o número de letras que a fruta tem. Destacamos essa linha a seguir:

```
minha_fruta = f"Nome: {fruta:12} - Número de letras: {len(fruta):3}"
```



Destaque f-string.

Para indicar a largura, ou melhor, o número de espaços que o conteúdo de uma variável deve ocupar, devemos utilizar a sintaxe `{variavel:n}`, onde temos o nome da variável, seguida de dois pontos (:) e o número de espaços (n) que se deve ocupar.

Caso o tamanho do conteúdo da variável seja menor que o número n, serão incluídos espaços em branco até completar esse tamanho. A posição dos espaços adicionados depende do tipo da variável. Para variáveis do tipo string, os espaços são adicionados à direita, enquanto para variáveis do tipo inteiro, os espaços são adicionados à esquerda.

Caso o tamanho do conteúdo da variável seja maior que o número n, a formatação será ignorada.

Retornando ao exemplo, desejamos imprimir o nome da fruta de forma que ela ocupe 12 espaços (`{fruta:12}`), e o número de letras da fruta deve ocupar apenas três espaços (`{len(fruta):3}`). Observe o resultado obtido no console à direita.

Linhas 10 até 16

No segundo trecho, definimos uma variável *float* na linha 10 e criamos três f-strings para exibir esse conteúdo.

A formatação com f-string nos permite um controle maior de como será exibido um número do tipo *float*, no qual podemos definir a largura e o número de casas decimais que devem ser exibidos. A sintaxe para formatar um *float* é a seguinte:

`{variavel_float:largura.precisao f}`

Pelo exemplo, temos o nome da variável do tipo *float* seguida de dois pontos (:), a largura total que o número deve ocupar, incluindo as casas decimais, e o ponto (separador de decimal), seguido de um ponto (.), o número de casas decimais (precisão) e a letra “f”, que deve estar junto à precisão. A largura é opcional.

Na primeira f-string, na linha 11, utilizamos a expressão {pi:.1f}, ou seja, queremos que seja exibido o valor da variável **pi** com uma casa decimal apenas. Como não especificamos a largura, ela será calculada de forma a acomodar toda a parte inteira do *float*.

Na f-string da linha 12, utilizamos a expressão {pi:6.1f}, que indica que o número deve ocupar seis espaços, sendo que, necessariamente, deve ter uma casa decimal.

Na última f-string, linha 13, utilizamos a expressão {pi:.4f}, para que seja exibido o número com quatro casas decimais. Observe, no console, como ficaram os resultados.

Linhas 20 até 24

No terceiro e último trecho, vamos mostrar como formatar datas em expressões f-string.

Na linha 20, definimos a variável **data** com a data atual, utilizando o método `datetime.now()`.

Na linha 21, criamos uma f-string para exibir o valor da variável **data** sem informar ao Python qual formatação ele deve utilizar ({data}). Com isso, a data foi impressa no formato padrão: 2020-08-13 10:50:32.262037.

Na linha 22, utilizamos a expressão {data:%d/%m/%Y}, que indica que desejamos exibir a data no formato “dia/mês/ano” (13/08/2020). Veja o resultado no console à direita.

Pesquise mais sobre o módulo *datetime* na documentação oficial para descobrir outras maneiras de formatar datas.

Vamos avançar!