---

title: Manipulação de strings em Python

layout: lesson

slug: manipulacao-de-strings-em-python

date: 2012-07-17

translation\_date: 2021-06-01

authors:

- William J. Turkel

- Adam Crymble

reviewers:

- Jim Clifford

editors:

- Miriam Posner

translator:

- Felipe Lamarca

translation-editor:

-

translation-reviewer:

-

difficulty: 2

review-ticket:

activity:

topics: [python]

abstract: "Esta lição é uma breve introdução às técnicas de manipulação de strings em Python."

redirect\_from:

original:

avatar\_alt: A man playing a guitar

---

{% include toc.html %}

Objetivos da lição

-----------------------

Essa lição é uma breve introdução às técnicas de manipulação de [strings](https://pt.wikipedia.org/wiki/Cadeia\_de\_caracteres) em Python. O conhecimento sobre manipulação de strings é crucial na maior parte das tarefas de processamento de texto. Se quiser experimentar as lições a seguir, você pode escrever e executar programas curtos como fizemos nas lições anteriores da série ou pode abrir um Python shell / Terminal para testá-los na linha de comando.

## Manipulando strings Python

Se já tiver tido contato com outras linguagens de programação, deve ter aprendido que é necessário \*declarar\* ou \*digitar\* variáveis antes de armazenar qualquer coisa nelas. Isso não é necessário ao trabalhar com strings em Python. Podemos criar uma string simplesmente colocando o conteúdo entre aspas com um sinal de igual (=):

```

mensagem = "Olá Mundo"

```

## Operadores de string: Adicionando e Multiplicando

Uma string é um tipo de objeto que consiste numa série de caracteres. A linguagem Python já sabe como lidar com várias representações poderosas e de uso geral, incluindo strings. Uma forma de manipular strings é utilizando \*operadores de string\*. Esses operadores são representados por símbolos associados à matemática, tais como +, -, \\*, / e =. Quando utilizados com strings, eles executam ações semelhantes, mas não iguais, aos seus correspondentes matemáticos.

### Concatenar

Esse termo significa unir strings. O processo é conhecido como \*concatenando\* strings e é executado utilizando o operador de soma (+). Note que é necessário ser explícito quanto ao local onde deseja que os espaços em branco ocorram, colocando-os também entre aspas simples.

Nesse exemplo, a strings "mensagem1" recebe o conteúdo "olá mundo":

```

mensagem1 = 'olá' + ' ' + 'mundo'

print(mensagem1)

-> olá mundo

```

### Multiplicar

Se quiser multiplas cópias duma strings, use o operador de multiplicação (\\*). Nesse exemplo, a string \*mensagem2a\* recebe o conteúdo "olá" três vezes; a string \*message2b\* recebe o conteúdo "mundo"; depois imprimimos (com a função \*print\*) as duas strings.

```

mensagem2a = 'olá ' \* 3

mensagem2b = 'mundo'

print(mensagem2a + mensagem2b)

-> olá olá olá mundo

```

### Append

E se quiser adicionar material ao final duma string sucessivamente? Há um operador especial para isso (+=).

```

mensagem3 = 'oi'

mensagem3 += ' '

mensagem3 += 'mundo'

print(mensagem3)

-> oi mundo

```

## Métodos de string: Encontrando, Modificando

Além dos operadores, Python vem pré-instalado com dezenas de métodos de string que permitem executar ações com strings. Utilizados sozinhos ou combinados, esses métodos podem fazer com strings qualquer coisa que imaginar. A boa notícia é que é possível consultar uma lista de Métodos de String no [site do Python](https://docs.python.org/2/library/stdtypes.html#string-methods), incluindo informações sobre como usar cada um corretamente. Para garantir que você tenha uma compreensão básica dos métodos de string, o que se segue é uma breve visão geral de alguns dos mais comumente usados:

### Comprimento (Length)

Você pode determinar o número de caracteres numa string usando `len`. Note que o espaço em branco conta como um caractere separado.

```

mensagem4 = 'olá' + ' ' + 'mundo'

print(len(mensagem4))

-> 9

```

### Encontrar (Find)

É possível buscar por uma substring numa string, e seu programa retornará o índice da posição inicial dessa substring. Isso é útil para fases posteriores de processamento. Note que os índices são numerados da esquerda para a direita e que a contagem começa na posição 0, não 1.

```

mensagem5 = "olá mundo"

mensagem5a = mensagem5.find("mun")

print(mensagem5a)

-> 4

```

Caso a substring não exista, o programa retornará o valor -1.

```

mensagem6 = "olá mundo"

mensagem6b = mensagem6.find("esquilo")

print(mensagem6b)

-> -1

```

### Minúsculas (Lower Case)

Por vezes é útil converter os caracteres duma string para letras minúsculas. Por exemplo, se padronizarmos o caso, será mais fácil para o computador reconhecer que "Às vezes" é o mesmo que "às vezes".

```

mensagem7 = "OLÁ MUNDO"

mensagem7a = mensagem7.lower()

print(mensagem7a)

-> olá mundo

```

O efeito oposto, isto é, tornar os caracteres maiúsculos, pode ser alcançado trocando `.lower` por `.upper`.

### Substituir (Replace)

Caso precise substituir uma substring ao longo da string, pode fazer isso através do método `replace`.

```

mensagem8 = "OLÁ MUNDO"

mensagem8a = mensagem8.replace("O", "pizza")

print(mensagem8a)

-> pizzaLÁ MUNDpizza

```

### Cortar (Slice)

Se quiser cortar (`slice`) partes do início ou do fim de uma string, pode fazer isso criando uma substring. O mesmo tipo de técnica também permite quebrar uma string longa em componentes mais fáceis de gerenciar.

```

mensagem9 = "Olá Mundo"

mensagem9a = mensagem9[1:7]

print(mensagem9a)

-> lá Mun

```

É possível substituir variáveis para os inteiros usados nesse exemplo.

```

loc\_inicial = 2

loc\_final = 7

mensagem9b = mensagem9[loc\_inicial:loc\_final]

print(mensagem9b)

-> á Mun

```

Isso torna muito mais fácil de utilizar esse método em conjunto com o método `find`, como no exemplo a seguir, que procura pela letra "d" nos primeiros seis caracteres de "Olá mundo" e corretamente informa que ela não está lá (-1). Essa técnica é muito mais útil em strings mais longas - documentos inteiros, por exemplo. Note que a ausência dum inteiro antes dos dois pontos significa que queremos começar no início da string. Poderíamos utilizar a mesma técnica para dizer ao programa para ir até o fim da string, não colocando nenhum número inteiro após os dois pontos. E lembre-se que as posições do índice começam a contar a partir do 0 ao invés de 1.

```

mensagem9 = "Olá Mundo"

print(mensagem9[:5].find("d"))

-> -1

```

Há vários outros, mas os métodos de string acima são um bom começo. Observe que, neste último exemplo, utilizamos colchetes no lugar de parênteses. Essa diferença na \*sintaxe\* sinaliza uma diferença importante. Em Python, parênteses geralmente são utilizados para \*passar um argumento\* para uma função. Então, quando vemos algo do tipo

```

print(len(mensagem7))

```

significa passar a string \*mensagem7\* para a função `len` e depois enviar o valor retornado para a instrução de impressão (\*print\*). Mesmo que a função possa ser chamada sem um argumento, geralmente é necessário incluir um par de parênteses vazios após o nome da função de qualquer modo. Vimos um exemplo disso também:

```

mensagem7 = "OLÁ MUNDO"

mensagem7a = mensagem7.lower()

print(mensagem7a)

-> olá mundo

```

Essa sentença diz ao Python que aplique a função `lower` à string \*mensagem7\* e armazene o valor retornado na string \*mensagem7a\*.

Os colchetes têm um propósito diferente. Se pensarmos numa string como uma sequência de caracteres e quisermos que seja possível acessar os conteúdos duma string através das suas localizações dentro da sequência, então precisamos, de alguma forma, informar ao Python uma localização dentro da sequência. É isso que os colchetes fazem: indicam a localização inicial e final dentro duma sequência, como vimos ao utilizar o método `slice`.

## Sequências de escape

O que você faz quando precisa incluir aspas numa string? Você não quer que o interpretador do Python entenda de forma incorreta e termine a string quando passar por um desses caracteres. Em Python, é possível adicionar uma barra (\) na frente das aspas de modo que ela não termine a string. Isso é conhecido como sequências de escape.

```

print('\"')

-> "

```

```

print('O programa imprimiu \"olá mundo\"')

-> O programa imprimiu "olá mundo"

```

Duas outras sequências de escape permitem imprimir tabulações e novas linhas:

```

print('olá\tolá\tolá\nmundo')

->olá olá olá

mundo

```

## Leituras sugeridas

- Lutz, \*Learning Python\*

- Capítulo 7: Strings

- Capítulo 8: Lists and Dictionaries

- Capítulo 10: Introducing Python Statements

- Capítulo 16: Function Basics

## Sincronização de Código

Para continuar com as lições futuras, é importante que tenha os ficheiros e programas corretos no seu diretório programming-historian. Ao final de cada capítulo você pode fazer o download do ficheiro zip do programming-historian para garantir que possui os códigos corretos. Observe que foram removidos os ficheiros desnecessários de lições anteriores. Seu diretório pode conter mais ficheiros e não há problema nisso!

- programming-historian-1 ([zip](https://programminghistorian.org/assets/python-lessons1.zip))