Revisitando Strings

MC102-2018s1-Aula12-180410

Arthur J. Catto, PhD
ajcatto@g.unicamp.br

10 de abril de 2018

1 Revisitando Strings

1.1 O modelo

Uma *string* é coleção sequencial e ordenada de caracteres. Os caracteres que compõem uma *string* são idenficados por um *índice* que indica sua posição na sequência.

Os índices podem ser positivos (indicando a posição dos caracteres a partir do começo da *string*) ou negativos (indicando a posição dos caracteres a partir do fim da *string*), como mostra o diagrama abaixo.

0	1	2	3	4	5	6	7	8	9	10
U	*m*	*a*		*s*	*t*	*r*	*i*	*n*	*g*	.
−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

Strings são representadas entre aspas simples, duplas ou "triplas". Nos dois primeiros casos, a *string* deve ficar toda contida numa única linha; no terceiro, ela pode ocupar um número arbitrário de linhas.

A *string* nula ou vazia não contém caractere algum e é representada por '' ou "" (duas aspas simples ou duplas, sem nada no meio).

```
In [3]: cadeia_a = ''
        print(cadeia_a)  # não vai mostrar nada...
        print(repr(cadeia_a))  # repr mostra as aspas delimitadoras
```

```
''
```

```
In [4]: cadeia_b = 'Uma string.'
        print(cadeia_b)
```

```
Uma string.
```

```
In [5]: cadeia_c = "Outra string."
        print(cadeia_c)
```

```
Outra string.
```

```
In [6]: cadeia_d = '''Uma cadeia
        com várias
        linhas'''
        print(cadeia_d)
```

```
Uma cadeia
com várias
linhas
```

1.2 Os operadores + e * O operador + concatena *strings* e o operador * repete (e ao mesmo tempo concatena) *strings*.

```
In [7]: string_a = 'a-'
        string_b = 'bl'
```

```
In [8]: string_c = string_b + string_a
        print(string_c)
```

```
bla-
```

```
In [9]: string_d = 3 * string_c
        print(string_d)
```

```
bla-bla-bla-
```

1.3 Métodos e funções para *strings*

1.3.1 Conversão para maiúsculas e minúsculas

```
In [10]: titulo = 'Laranja Madura na Beira da Estrada'
```

```
In [11]: print(titulo.lower())
```

```
laranja madura na beira da estrada
```

```
In [12]: print(titulo.upper())
```

```
LARANJA MADURA NA BEIRA DA ESTRADA
```

```
In [13]: print(titulo.capitalize())
```

```
Laranja madura na beira da estrada
```

1.3.2 Remoção de espaços à esquerda e à direita

```
In [14]: titulo = '    Laranja Madura na Beira da Estrada   '

In [15]: print(repr(titulo.lstrip()))

'Laranja Madura na Beira da Estrada   '


In [16]: print(repr(titulo.rstrip()))

'    Laranja Madura na Beira da Estrada'


In [17]: print(repr(titulo.strip()))

'Laranja Madura na Beira da Estrada'
```

1.3.3 Contagem e substituição de *substrings*

```
In [18]: titulo = 'Laranja Madura na Beira da Estrada'.lower()

In [19]: print(titulo.count('ra'))

4


In [20]: print(titulo.replace('ra', 'RA'))

laRAnja maduRA na beiRA da estRAda
```

1.3.4 Ajuste do comprimento da *string*

```
In [21]: titulo = 'Laranja Madura na Beira da Estrada'

In [22]: print(repr(titulo.ljust(50)))

'Laranja Madura na Beira da Estrada                '


In [23]: print(repr(titulo.rjust(50)))

'                Laranja Madura na Beira da Estrada'


In [24]: print(repr(titulo.center(50)))

'        Laranja Madura na Beira da Estrada        '
```

1.3.5 Localizar uma subcadeia

```
In [25]: titulo = 'Laranja Madura na Beira da Estrada'.lower()

In [26]: print(titulo.find('ra'))  ## acha primeira ocorrência do argumento a partir da esque

2


In [27]: print(titulo.rfind('ra'))  ## acha primeira ocorrência do argumento a partir da dire

30


In [28]: titulo = 'Laranja Madura na Beira da Estrada'.lower()

In [29]: print(titulo.find('xx'))  ## find e rfind retornam -1 se a subcadeia não for encontr
         print(titulo.rfind('xx'))

-1
-1


In [30]: titulo = 'Laranja Madura na Beira da Estrada'.lower()

In [31]: print(titulo.index('ra'))  ## o mesmo que find e rfind se a subcadeia for encontrada
         print(titulo.rindex('ra'))

2
30


In [32]: titulo = 'Laranja Madura na Beira da Estrada'.lower()

In [33]: print(titulo.index('xx'))  ## index e rindex causam um erro se a subcadeia não for e


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

         ValueError                                Traceback (most recent call last)

         <ipython-input-33-81a3cb646268> in <module>()
    ----> 1 print(titulo.index('xx'))  ## index e rindex causam um erro se a subcadeia não fo


         ValueError: substring not found


In [42]: print(titulo.rindex('xx'))


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

         ValueError                                Traceback (most recent call last)
```

```
    <ipython-input-42-4c71121bc196> in <module>()
----> 1 print(titulo.rindex('xx'))


ValueError: substring not found
```

1.3.6 Formatação

Este é um método com muitos recursos que não podem ser completamente explorados aqui, mas os exemplos a seguir dão uma rápida ideia do que é possível fazer.

```
In [43]: a = 10
         b = 3
         c = 7
         print('A média de {}, {} e {} é {}.'.format(a, b, c, (a + b + c) / 3))
```

```
A média de 10, 3 e 7 é 6.666666666666667.
```

Suponha que quiséssemos que todos os valores fossem exibidos com uma casa decimal.

```
In [44]: print('A média de {:.1f}, {:.1f} e {:.1f} é {:.1f}.'.format(a, b, c, (a + b + c) / 3
```

```
A média de 10.0, 3.0 e 7.0 é 6.7.
```

O mesmo efeito pode ser conseguido colocando um *f* à frente da *string* de formato e incluindo as expressões nas chaves correspondentes.

```
In [45]: print(f'A média de {a:.1f}, {b:.1f} e {c:.1f} é {(a + b + c) / 3:.1f}.')
```

```
A média de 10.0, 3.0 e 7.0 é 6.7.
```

Uma *string* de formatação pode ser manipulada como qualquer outra *string* de Python.

```
In [46]: mens_1 = f'A média de {a:.1f}, {b:.1f} e {c:.1f} é '
         mens_2 = f'{(a + b + c) / 3:.1f}.'
         print(mens_1 + mens_2)
```

```
A média de 10.0, 3.0 e 7.0 é 6.7.
```

1.3.7 Comprimento

A função `len` retorna o número de caracteres na *string* à qual ela é aplicada.

```
In [47]: mascote = 'tamanduá'
         print(mascote, len(mascote))
```

```
tamanduá 8
```

Note que os índices positivos válidos para uma *string* s são $0, 1, \ldots \text{len}\,(s) - 1$, enquanto os negativos são $-1, -2, \cdots - \text{len}\,(s)$.

```
In [48]: print(mascote[0], mascote[len(mascote)-1], mascote[-1], mascote[-len(mascote)])
```

```
t á á t
```

1.3.8 Fatiamento (*slicing*)

Slices (*fatias*) são subcadeias que podem ser extraídas pelo '*fatiador*', como no caso de listas.

```
In [49]: titulo = 'Laranja Madura na Beira da Estrada'.lower()
         fruta = titulo[:7]
         local = titulo[-7:]
         print(repr(fruta), repr(local))
```

```
'laranja' 'estrada'
```

O fatiador não dá erro caso receba um índice inválido. Quando isso acontece, ele usa um índice razoável.

```
In [50]: titulo = 'Laranja Madura na Beira da Estrada'.lower()
         sem_fruta = titulo[8:99]
         sem_local = titulo[-99:-8]
         print(repr(sem_fruta), repr(sem_local))
```

```
'madura na beira da estrada' 'laranja madura na beira da'
```

```
In [51]: nums = '0123456789'
         impares = nums[1::2]
         pares = nums[::2]
         inversa = nums[::-1]
         print(repr(nums), repr(impares), repr(pares), repr(inversa))
```

```
'0123456789' '13579' '02468' '9876543210'
```

1.3.9 Comparação de cadeias

Os operadores de comparação podem ser usados normalmente. A comparação é lexicográfica e baseia-se no código de caracteres ASCII.

```
In [52]: print("'menor' < 'maior'  é", 'menor' < 'maior')
```

```
'menor' < 'maior'  é False
```

```
In [53]: print("'maior' < 'Maior'  é", 'maior' < 'Maior')
```

```
'maior' < 'Maior'  é False
```

```
In [54]: print("'maior' == 'Maior'  é", 'maior' == 'Maior')
```

```
'maior' == 'Maior'  é False
```

```
In [55]: print("'maior' != 'Maior'  é", 'maior' != 'Maior')
```

```
'maior' != 'Maior'  é True
```

O código de um caractere na tabela ASCII é dado pela função `ord` e o caractere correspondente a um dado código é dado pela função `chr`.

```
In [56]: print(f"{ord('M'):3} {chr(77)}")
         print(f"{ord('m'):3} {chr(109)}")
         print(f"{ord('a'):3} {chr(97)}")
         print(f"{ord('e'):3} {chr(101)}")
```

```
 77 M
109 m
 97 a
101 e
```

1.4 *Strings* são imutáveis

Ao contrário do que acontece com listas, não é possível alterar o conteúdo de uma *string*. Qualquer tentativa nesse sentido causará um erro.
Para conseguir uma variante da *string* original primeiro é preciso criar uma cópia.

```
In [57]: titulo = 'Laranja Maduro na Beira da Estrada'
         print(titulo[13])
```

```
o
```

```
In [58]: titulo[13] = 'a'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-58-1be28549ced2> in <module>()
----> 1 titulo[13] = 'a'


TypeError: 'str' object does not support item assignment
```

```
In [59]: titulo_corr = titulo[:13] + 'a' + titulo[14:]
         print(titulo_corr)
```

```
Laranja Madura na Beira da Estrada
```

1.5 Percorrendo *strings* com *for*

```
In [60]: texto = 'Uma string.'
         for c in texto:
             print(c, end=' ')
         print()
```

```
U m a   s t r i n g .
```

```
In [61]: texto = 'Uma string.'
         for i in range(len(texto)):
             print(texto[i], end=' ')
         print()
```

```
U m a   s t r i n g .
```

```
In [62]: texto = 'Uma string.'
         for i in range(-1, -len(texto) - 1, -1):
             print(texto[i], end=' ')
         print()
```

```
. g n i r t s   a m U
```

1.6 Percorrendo *strings* com *while*

```
In [63]: texto = 'Uma string.'
         i = 0
         while i < len(texto):
             print(texto[i], end=' ')
             i += 1
         print()
```

```
U m a   s t r i n g .
```

```
In [64]: texto = 'Uma string.'
         i = -1
         while i >= -len(texto):
             print(texto[i], end=' ')
             i -= 1
         print()
```

```
. g n i r t s   a m U
```

1.7 Os operadores *in* e *not in*

Os operadores *in* e *not in* testam se uma *string* é subcadeia de outra.

```
In [65]: titulo = 'Laranja Madura na Beira da Estrada'.lower()
         fruta = 'banana'

         print("'banana' in titulo   ", fruta in titulo)
```

```
'banana' in titulo    False
```

```
In [66]: print("'banana' not in titulo   ", fruta not in titulo)
```

```
'banana' not in titulo    True
```

```
In [67]: subs = 'na Beira'

         print("'" + subs + "' in titulo   ", subs in titulo)
         print("'" + subs + "' not in titulo   ", subs not in titulo)
```

```
'na Beira' in titulo    False
'na Beira' not in titulo    True
```

```
In [68]: subs = 'na beira'

         print("'" + subs + "' in titulo   ", subs in titulo)
         print("'" + subs + "' not in titulo   ", subs not in titulo)
```

```
'na beira' in titulo    True
'na beira' not in titulo    False
```

1.8 Exemplos

1.8.1 Remover todas as vogais de uma linha de texto

Como uma *string* não pode ser modificada, temos que criar uma "cópia" selecionando os caracteres que devem ser incluídos nela.

```
In [34]: texto = 'É difícil não esquecer dos detalhes.'
         vogais = 'aáàãâeéêiíoóõôuú'
         vogais += vogais.upper()

         texto_sem_vogais = ''
         for c in texto:
             if c not in vogais:
                 texto_sem_vogais += c
         print(repr(texto))
         print(repr(texto_sem_vogais))
```

```
'É difícil não esquecer dos detalhes.'
' dfcl n sqcr ds dtlhs.'
```

1.8.2 Remover de uma cadeia todas as ocorrências de uma subcadeia dada

```
In [35]: texto = 'Laranja Madura na Beira da Estrada'.lower()
         print('    texto =', repr(texto))
         subs = 'ra'
         print('     subs =', repr(subs))
         resultado = texto

         continuar = True
         while continuar:
```

```
            ix = resultado.find(subs)
            if ix != -1:
                resultado = resultado[:ix] + resultado[ix + len(subs):]
            else:
                continuar = False
        print('resultado =', repr(resultado))

    texto = 'laranja madura na beira da estrada'
     subs = 'ra'
resultado = 'lanja madu na bei da estda'
```

1.8.3 Verificar se uma frase é palíndroma

Dada uma frase, verificar se ela é palíndroma, desconsiderando acentos, espaços e pontuação. Uma frase é palíndroma se ela puder ser lida igualmente nos dois sentidos.

Um esboço de solução com alto nível de abstração poderia ser:

- ler a frase
- eliminar caracteres a serem desconsiderados
- verificar se é palíndroma
- exibir o resultado da verificação

```
In [36]: # ler a frase original
         frase_ori = input('Digite uma frase: ')

Digite uma frase: Socorram-me, subi no ônibus em Marrocos!
```

```
In [37]: import string

         alfabeto = string.ascii_lowercase
         Alfabeto = string.ascii_uppercase
         acentos = 'áàãâéêíóõôúç'
         sem_acentos = 'aaaaeeiooouc'
         acentos += acentos.upper()
         sem_acentos = sem_acentos * 2
```

```
In [38]: # criar frase modificada,
         # omitindo caracteres a serem desconsiderados

         frase_mod = ''
         for c in frase_ori:
             if c in alfabeto:
                 frase_mod += c
             elif c in Alfabeto:
                 frase_mod += alfabeto[Alfabeto.index(c)]
             elif c in acentos:
                 frase_mod += sem_acentos[acentos.index(c)]
```

```
In [39]: # verificar se a frase modificada é palíndroma

         meio = len(frase_mod) // 2
```

```
        metade_esq = frase_mod[:meio]
        metade_dir_rev = frase_mod[:-meio-1:-1]

        eh_palindroma = (metade_esq == metade_dir_rev)
```

In [40]:
```
        # exibir o resultado da verificação
        if eh_palindroma:
            print("'" + frase_ori + "'", 'é palíndroma.')
        else:
            print("'" + frase_ori + "'", 'não é palíndroma.')
```

```
'Socorram-me, subi no ônibus em Marrocos!' é palíndroma.
```