

Operadores e Funções para Strings

Prof. Alberto Costa Neto
Programação em Python

Biblioteca (Library) de String

- Python tem várias **funções** que manipulam strings na **biblioteca string**
- Estas **funções** pertencem à própria string. Para chamá-las, basta adicionar um ponto '.' e a chamada a função após o nome da variável
- Estas **funções** não modificam a string original. Ao invés disso, criam e retornam uma nova string que contém o valor alterado

```
>>> cump = 'Ola Bob'
>>> cump_min = cump.lower()
>>> print(cump_min)
ola bob
>>> print(cump)
Ola Bob
>>> print('Bom Dia!'.lower())
bom dia!
>>>
```

```
>>> s = 'Ola Mundo'
>>> type(s)
<class 'str'>
>>> dir(s)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

<https://docs.python.org/3/library/stdtypes.html#string-methods>

str.partition(*sep*) ¶

Split the string at the first occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings.

str.replace(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

str.rfind(*sub*[, *start*[, *end*]])

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

str.rindex(*sub*[, *start*[, *end*]])

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

str.rjust(*width*[, *fillchar*])

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

str.rpartition(*sep*)

Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

Biblioteca String

`str.capitalize()`

`str.center(width[, fillchar])`

`str.endswith(suffix[, start[, end]])`

`str.find(sub[, start[, end]])`

`str.lstrip([chars])`

`str.partition(sep)`

`str.replace(old, new[, count])`

`str.rfind(sub[, start[, end]])`

`str.lower()`

`str.rstrip([chars])`

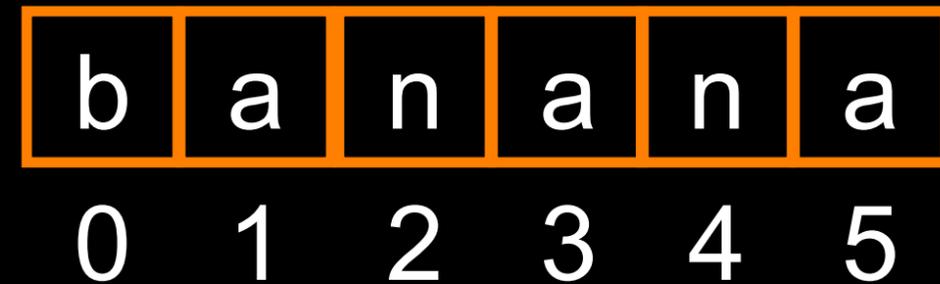
`str.split(sep=None, maxsplit=-1)`

`str.strip([chars])`

`str.upper()`

Buscando em uma String

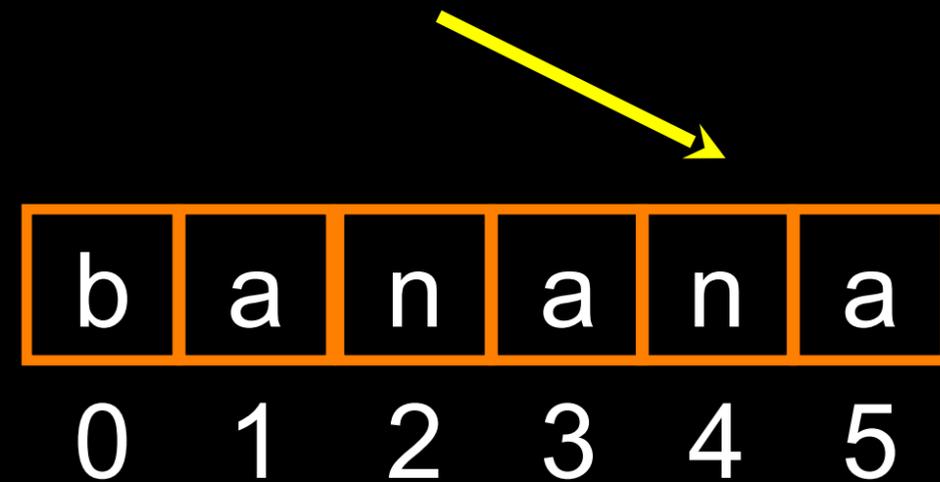
- Usamos `find()` para buscar uma substring dentro de outra string
- `find()` encontra a posição da primeira ocorrência da substring, iniciando a busca pelo índice 0
- Se a substring não for encontrada, `find()` retorna `-1`
- Lembre-se que as posições em strings começam em 0



```
>>> fruta = 'banana'
>>> pos = fruta.find('na')
>>> print(pos)
2
>>> pos_z = fruta.find('z')
>>> print(pos_z)
-1
```

Buscando em uma String começando da direita

- Usamos `rfind()` para buscar uma substring dentro de outra string
- `rfind()` encontra a posição da primeira ocorrência da substring, iniciando a busca pelo final (direita)
- Se a substring não for encontrada, `rfind()` retorna `-1`



```
>>> fruta = 'banana'
>>> pos = fruta.rfind('na')
>>> print(pos)
4
```

Transformando tudo para maiúsculas (UPPER CASE)

- Podemos criar uma cópia de uma string em **lower case** para **upper case**
- Com certa frequência, ao fazer buscas em uma String com a função **find()**, precisamos converter para **lower case** para que a busca ocorra corretamente

```
>>> cump = 'Ola Bob'
>>> maiu = cump.upper()
>>> print(maiu)
OLA BOB
>>> minu = cump.lower()
>>> print(minu)
ola bob
>>>
```

Busca e Substituição

- A função `replace()` trabalha como a operação de “localizar e substituir” de um editor de texto
- Ela substitui **todas as ocorrências** da **string de busca** pela **string substituta**

```
>>> cump = 'Ola Bob! Tchau Bob!'
>>> nstr = cump.replace('Bob', 'Jane')
>>> print(nstr)
Ola Jane! Tchau Jane!
>>> nstr = nstr.replace('e', 'io')
>>> print(nstr)
Ola Janio! Tchau Janio!
>>>
```

Extraindo Espaços em Branco

- É comum precisarmos remover espaços do início e/ou final de uma String
- `lstrip()` e `rstrip()` removem espaços em branco à esquerda e à direita, respectivamente
- `strip()` remove os brancos tanto no início como no final

```
>>> cumprimento = ' Ola Bob '
```

```
>>> cumprimento.lstrip()
```

```
'Ola Bob '
```

```
>>> cumprimento.rstrip()
```

```
' Ola Bob'
```

```
>>> cumprimento.strip()
```

```
'Ola Bob'
```

```
>>>
```

Prefixos e Sufixos

```
>>> linha = 'Por favor, tenha um bom dia!'
>>> linha.startswith('Por favor')
True
>>> linha.startswith('p')
False
>>> linha.endswith('!')
True
```

Analizando e Extraindo

12 19
↓ ↓

From alberto@ufs.br Sat Jan 5 09:14:16 2008

```
>>> dado = 'From alberto@ufs.br Sat Jan 5 09:14:16 2008'
>>> pos_arroba = dado.find('@')
>>> print(pos_arroba)
12
>>> pos_espaco = dado.find(' ', pos_arroba)
>>> print(pos_espaco)
19
>>> dominio = dado[pos_arroba + 1 : pos_espaco]
>>> print(dominio)
ufs.br
```