Strings em Python

Durante o curso já falamos e utilizamos strings em diversas ocasiões. Strings nada mais são que sequências de caracteres. Ao contrário dos tipos de dados primitivos que vimos nas primeiras aulas, como `int`, `float` e `bool`, a string é considerada uma estrutura pré-definida, composta por vários caracteres.

Na aula de hoje, veremos mais alguns detalhes sobre como podemos criar e manipular strings de diversas formas.

Criando strings

Strings podems ser criadas utilizando tanto aspas simples, `'`, quanto aspas duplas, `"`. Não há uma regra especificando qual tipo de aspa adotar. O guia do Python, PEP-8, diz que você deve escolher o tipo que for mais confortável para você. No entanto, é importante escolher um e adotá-lo sempre.

A exceção a essa regra é quando é necessário incluir aspas dentro da string. Nesse caso, há duas opções:

- Utilize o outro tipo de aspas para criar a string. Essa é a solução que gera um código mais limpo e fácil de ler;
- Utilize o caractere de escape `\` antes das aspas. Esse caractere de escape informa ao interpretador do Python que o caractere seguinte deve ser impresso como foi digitado.

Também é possível incluir uma string com múltiplas linhas. Nesse caso, utilizamos aspas triplas, `"""`. Lembre-se que, nesse caso, o Python entende toda quebra de linha e espaço que você inserir dentro da string!

Veja os exemplos abaixo:

```
texto = "Este é um texto com aspas duplas"
print(texto)
texto = 'Este é um texto com aspas simples'
print(texto)
texto = "Com 'aspas simples' no meio do texto, envolvemos com aspas duplas"
print(texto)
texto = 'Com "aspas duplas" no meio do texto, envolvemos com aspas simples'
print(texto)
texto = "Podemos também usar um \"caractere de escape\""
print(texto)

texto = """
Este é um exemplo
de texto com múltiplas linhas!

Veja que o Python captura todas as quebras de linha,
inclusive a primeira e a última!
"""

print(texto)
```

```
print(texto)
```

```
Este é um texto com aspas duplas
Este é um texto com aspas simples
Com 'aspas simples' no meio do texto, envolvemos com aspas duplas
Com "aspas duplas" no meio do texto, envolvemos com aspas simples
Podemos também usar um "caractere de escape"

Este é um exemplo
de texto com múltiplas linhas!

Veja que o Python captura todas as quebras de linha,
inclusive a primeira e a última!
```

Alguns detalhes técnicos sobre strings

Em Python, as strings são armazenadas na memória como caracteres individuais, dispostos em sequência. O benefício, no caso, é que strings podem ser acessadas tanto começando pelo primeiro caractere, quanto pelo último.

Nesse sentido, a string em Python funciona como uma lista. Basta indicarmos o índice do caractere que queremos exibir, lembrando que índices em Python iniciam no `0`.

```
texto = "Olá, mundo!"

print(texto[5])  # pega o sexto caractere, da esq. para dir.
print(texto[-3]) # pega o terceiro caractere, da dir. para esq.
```

```
m
d
```

Operadores de strings

Em Python Podemos operar com strings de três formas:

1. Operadores básicos: `*` e `+`;
2. Operadores de pertencimento: `in` e `not in`;
3. Operadores relacionais: `==`, `!=`, `>`, `<`, etc.

O operador `+` indica uma **concatenação de caracteres**, ou seja, vamos dispor um caractere após o outro. Já o operador `*` indica a **replicação** da string.

Os operadores de pertencimento funcionam exatamente como em listas. Nesse caso, podemos utilizar esses operadores para verificar se um caractere (ou uma sequência de caracteres) está contido na string.

Já os operadores relacionais comparam duas strings baseados no código da tabela ASCII de cada caractere, começando pelo caractere mais à esquerda. Para mais informações sobre os códigos da tabela ASCII, clique aqui.

```
print("Olá, " + "mundo!")
print("-" * 30)
print("Olá, " in "Olá, mundo!")
print("Olá, " not in "olá, mundo!") # Os caracteres "O" e "o" são diferentes!
print("abc" > "def")
print("oi" != "OI")
print("a" < "A")
```

```
Olá, mundo!
------------------------------
True
True
False
True
False
```

Strings como listas

Assim como os operadores de pertencimento e a notação de índices, podemos utilizar vários recursos de listas em strings. Alguns exemplos são indicados abaixo:

- Uso da função `len()` para identificar o tamanho da string;
- Uso da estrutura de repetição `for` para percorrer caracteres em uma string;
- Uso do "fatiamento" (*slicing*) de strings.

```
texto = "Olá, mundo!"
print(len(texto))

for caractere in texto:
  print(caractere)

print(texto[:3])
```

```
11
O
l
á
,

m
u
n
d
o
!
Olá
```

Algumas funções de strings

O bloco de código abaixo indica algumas funções úteis para manipular strings. Utilize esse guia como uma "cola" para os seus códigos. Todas as operações possíveis estão listadas aqui.

```
# Contar ocorrências de uma substring em uma string
```

```
print("olá, mundo!".count("o"))

# Localizar uma substring em uma string
# Indica o índice do primeiro caractere da substring
print("olá, mundo!".find("olá,"))
print("olá, mundo!".find("Olá,")) # Substring não é encontrada, retorna -1

# A função index também obtém o índice do primeiro caractere da substring
# No entanto, a função retorna uma exceção caso não localize a substring
# Não recomendo utilizar essa função!
print("olá, mundo!".index("olá,"))
# print("olá, mundo!".index("Olá,")) -> vai subir uma exceção do tipo ValueError

# Divide uma string, considerando um separador
# Retorna uma lista com as substrings obtidas
print("olá, mundo!".split(","))

# Une os elementos de uma lista de strings, adotando o separador definido
print(" - ".join(["1", "2", "3", "4", "5"]))

# Coloca todos os caracteres com letras maiúsculas ou minúsculas
# Gosto de usar essas funções para comparar strings e inputs de usuários
print("Olá, mundo".upper())
print("Olá, mundo".lower())

# Coloca o primeiro caractere em caixa alta
print("o rato roeu a roupa do rei de roma".capitalize())

# Coloca o primeiro caractere de cada palavra em caixa alta
print("o rato roeu a roupa do rei de roma".title())

# Troca o case (maiúscula por minúscula)
print("Olá!".swapcase())

# Identifica se a string é de algum dos tipos listados
print("oi".isupper())         # é maiúscula
print("oi".islower())         # é minúscula
print("Olá Mundo".istitle())  # tem o estilo de título
print("oi1".isalnum())        # possui apenas caracteres alfanuméricos
print("oi1".isalpha())        # possui apenas letras
print("oi1".isdigit())        # possui apenas números, 0-9
print("oi1".isnumeric())      # possui apenas números, em qualquer notação
                              # por exemplo, 三 é o ideograma chinês para 3
                              # a função isnumeric() retorna True
                              # a função isdigit() retorna False
print("   ".isspace())        # indica se possui apenas espaços

# Funções para organização de strings
print("Olá, mundo".ljust(40)) # alinha o texto à esquerda e preenche com espaços
                              # os demais caracteres
print("Olá, mundo".rjust(40)) # alinha o texto à direita e preenche com espaços
                              # os demais caracteres
print("Olá, mundo".center(40))# centraliza o texto e preenche com espaços os
                              # demais caracteres
print("  123  ".lstrip())     # remove espaços em branco à esquerda
print("  123  ".rstrip())     # remove espaços em branco à direita
print("  123  ".strip())     # remove espaços em branco à esquerda e à direita

# Identifica se a string começa ou termina com uma determinada substring
print("Olá, mundo!".startswith("Olá"))
print("Olá, mundo!".endswith("Olá"))
```

```
# Substitui uma substring dentro de uma string por outra substring
print("Olá, mundo!".replace("Olá, ", "olá, "))
```

```
2
0
-1
0
['olá', ' mundo!']
1 - 2 - 3 - 4 - 5
OLÁ, MUNDO
olá, mundo
O rato roeu a roupa do rei de roma
O Rato Roeu A Roupa Do Rei De Roma
oLÁ!
False
True
True
True
False
False
False
True
Olá, mundo
                                Olá, mundo
               Olá, mundo
123
  123
123
True
False
olá, mundo!
```

Formatação de strings

Muitas vezes nos deparamos com situações nas quais queremos exibir um resultado, utilizando dados contidos em variáveis para compor a nossa string.

O Python permite uma variedade enorme de formas de compor essa string. Eu tenho uma preferência pela **f-string**, ou *formatted string*. Na prática não há uma solução correta, porém as f-strings tendem a ser mais eficientes que as demais soluções, além de serem incrivelmente mais legívies que qualquer outra alternativa. Este artigo apresenta alguns argumentos pelos quais eu recomendo fortemente o uso de f-strings.

Para usar uma f-string, basta iniciar a string com um `f` antes das aspas. Para incluir valores de variáveis dentro de strings, basta posicionar os nomes das variáveis entre chaves dentro do corpo da string.

Uma observação: é uma boa prática utilizar f-strings **apenas** quando há a inclusão de uma variável. Caso seja incluída uma f-string sem uma variável, a impressão passada é que deveria ter uma variável naquele texto, porém alguém esqueceu de incluir.

Dentro de f-strings também é possível formatar a apresentação de dados numéricos. Isso é algo particularmente importante quando estamos lidando com números `float`.

Abaixo vamos colocar dois exemplos de como formatar números. Recomendo acessar essa página da documentação oficial do Python para mais informações sobre a formatação de strings.

```
nome = "Victor"
preco_total = 29.99
quantidade = 16

print("Dados da venda")  # veja que não há inclusão de variáveis, logo não
                         # usamos uma f-string
print(f"Nome: {nome}")

# A formatação abaixo indica que o número é inteiro e deve possuir 4
# caracteres. O que não for preenchido pelos números será preenchido por espaços
print(f"Quantidade comprada: {quantidade:4d}")
print(f"Preço total: {preco_total}")

# A formatação abaixo indica que o número possui até 4 caracteres, com
# 2 caracteres à direita do ponto, e é do tipo float
print(f"Valor unitário: {(preco_total / quantidade):4.2f}")
```

```
Dados da venda
Nome: Victor
Quantidade comprada:   16
Preço total: 29.99
Valor unitário: 1.87
```