PYTHON ACADEMY

STRINGS NO PYTHON

Ainda com dificuldade para manipular Strings no Python? Então chega mais que você vai DOMINAR Strings depois desse Post!

PYTHONACADEMY.COM.BR

Salve Salve Pythonista!

Nesse post você vai aprender tudo sobre as Strings no Python: - O que são strings - Como definimos strings no Python - Como funciona a indexação de strings - Operações entre strings - Formatação de strings com `format()` e **f-strings**

E muito mais! Então se prepara, faz seu café e vamos nessa!

Introdução

Uma string é um tipo de dados usado para armazenar informações de texto.

São considerados tipos simples pois seus valores são compostos por uma sequência de caracteres, podendo conter números, letras e pontuações.

As strings em Python são sempre colocadas entre aspas simples ou aspas duplas.

Para o Python **'Jornada Python'** é a mesma coisa que **"Jornada Python"**.

Agora vamos atribuir nossa string a uma variável e visualizar seu tipo de dado:

```
texto1 = 'Olá mundo'
texto2 = "Olá mundo"

print(texto1)
print(type(texto2))
```

Quando a saída de um type for `str`, saíba que é uma string:

```
Olá mundo
<class 'str'>
```

Também é possível criar Strings multilinhas utilizando três aspas, simples ou duplas:

```
multilinha_dupla = """
Is simply dummy text of the printing and typesetting industry
Lorem Ipsum has been the industry's standard dummy text ever since the
        1500s,
when an unknown printer took a galley of type and scrambled it to make
        a type."""

multilinha_simples = '''
Is simply dummy text of the printing and typesetting industry
Lorem Ipsum has been the industry's standard dummy text ever since the
        1500s,
when an unknown printer took a galley of type and scrambled it to make
        a type.'''
```

Na saída as linhas serão quebradas da mesma forma que está no código.

Índices e cortes (slicing)

Strings são sequências de caracteres, ou seja, uma sequência indexada onde podemos acessar qualquer caractere a partir de uma posição desejada.

Vamos a um pequeno exemplo onde iremos acessar o primeiro caractere de uma string para entendermos um conceito bem importante.

Veja:

```
nome = "Erick"

print(nome[0])
print(nome[1])
print(nome[2])
```

Resultando nos três primeiros caracteres:

```
E
r
i
```

> ⚠️ *Importante ressaltar que todo iterável do Python (Listas, Tuplas, Strings) são indexados começando-se pelo índice* `0` *e não pelo* `1` *.*

Caso tente acessar um índice inexistente, o seguinte erro será exibido:

```
IndexError: string index out of range
```

Mas não é somente com números positivos que acessamos os caracteres de uma strings, também temos a indexação negativa:

```
nome = "Erick"

print(nome[-1])
print(nome[-2])
print(nome[-3])
```

E a saída:

```
k
c
i
```

> *Utilizando a **Indexação Negativa**, o índice `-1` aponta para o último caracter da string; `-2` para o penúltimo; `-3` para o antepenúltimo e assim por diante.*

Use dois pontos para obter os caracteres de uma posição inicial até uma posição final:

```
nome = "Pythonista"

# Desde a posição de índice 3 (que é o "h")
# até a última posição da string
print(nome[3:])

# Desde a posição de índice -3 (que é o "s")
# até a última posição da string
print(nome[-3:])
```

Resultando em:

```
honista
sta
```

Também podemos retornar um intervalo de caracteres, especificando um índice inicial e final, separados por dois pontos.

Exemplo:

```
texto = "Python é uma linguagem de programação de alto nível"

print(texto[13:23])
```

E a saída será:

```
linguagem
```

Operações com strings

Juntar string pode ser algo essencial e saber como realizar essa operação é muito importante!

Em python podemos concatenar strings, ou seja, juntar duas ou mais strings em uma única, utilizando o operador `+`, desta forma:

```
var1 = 'variável'
var2 = 'string'

print(var1 + var2)
print(var1 + " " + var2)
```

E a saída será:

```
variávelstring
variável string
```

Que tal multiplicar uma string?

Use o operador `*` para isso. Exemplo:

```
linha = '-'

print(linha * 28)
print("Nosso código")
print(linha * 28)
```

Saída:

```
----------------------------
Nosso código
----------------------------
```

Formatação de Strings utilizando `format()`

Agora vamos supor que você queira concatenar uma variável que contém uma string com um número, assim:

```
var = 'Média: '

print(var + 10)
```

Se tentar executar esse código, o seguinte erro será mostrado:

```
TypeError: can only concatenate str (not "int") to str
```

Python só permite a concatenação utilizando o operador `+` quando os dois operandos são strings!

Para contornar este problema, podemos utilizar a formatação de strings.

Para isso existem, basicamente, duas formas: a primeira utilizando o método `format()` das strings e a outra - muito mais utilizada atualmente - é o conceito de F-Strings.

```
print('{} {}'.format(var, 1))
```

O método `.format()` pega os argumentos passados e insere dentro da string onde os marcadores `{}` estão posicionados, na ordem dos argumentos.

Resultado do código acima:

```
texto 1
```

Você pode utilizar esse recurso quantas vezes for necessário:

```
qtd_macas = 10
qtd_melancias = 3
qtd_uvas = 22.2

texto = 'Vou comprar {} maças, trocar {} melancias e comer {} uvas'
print(texto.format(qtd_macas, qtd_melancias, qtd_uvas))
```

Saída:

```
Vou comprar 10 maças, Trocar 3 melancias e comer 22.2 uvas
```

Também podemos definir as posições em que os argumentos serão inseridos:

```
qtd_macas = 10
qtd_melancias = 3
qtd_uvas = 22.2

texto = 'Vou comprar {2} maças, trocar {1} melancias e comer {0} uvas'
print(texto.format(qtd_macas, qtd_melancias, qtd_uvas))
```

Observe que realmente trocamos as posições em que os argumentos estão colocados:

```
Vou comprar 22.2 maças, Trocar 3 melancias e comer 10 uvas
```

> 💡 *Estou construindo o **DevBook**, uma plataforma que usa IA para criar ebooks técnicos — com código formatado e exportação em PDF. Não deixe de conferir!*

Formatação utilizando F-Strings

Esse conceito já foi abordado em um Post exclusivo aqui no Blog!

Se quiser conferir, termine primeiro este Post e depois clique aqui para dominar a formatação de strings com F-Strings!

> *Mas só depois de terminar este aqui hein!* 😅

Strings em Estruturas de repetição

Sabemos que as strings são iteráveis e por isso, podemos utilizar as estruturas de repetição `for` e `while` para percorrer seus itens.

Vamos a um exemplo com `for`:

```
txt = 'Python'

for item in txt:
    print(item)
```

E outro exemplo com `while`:

```
indice = 0
while indice < len(txt):
    print(txt[indice])
    indice += 1
```

Ambos produzem o mesmo resultado:

```
P
y
t
h
o
n
```

Uma pequena brincadeira com concatenação e estruturas de repetição:

```
prefixo = "MSCBGM"
sufixo = "ola"

for caractere in prefixo:
    print(caractere + sufixo)
```

Saída do código acima:

```
Mola
Sola
Cola
Bola
Gola
Mola
```

Perceba que o código iterou sobre os caracteres da variável `prefixo`, adicionando o sufixo `ola` ao final.

Se ainda não domina Estruturas de Repetição com `for` e `while`, se liga que nós temos artigos completos sobre o assunto: - Estruturas de Repetição utilizando `while` - Estruturas de Repetição utilizando `for`

Verificar se uma String pertence à outra String

Para verificar se uma determinada sequência de caracteres existe em uma string, basta usar o operador `in`, desta forma:

```
texto = "As strings são sequências de caracteres"

print('são' in texto)
```

Retornando:

```
True
```

Outro modo que podemos fazer essa verificação é utilizando o condicional `if` e o operador `in`, da seguinte maneira:

```
texto = "Tenha um ótimo dia"

if 'dia' in texto:
    print("A sequência de caractere existe na string!")
```

A saída será:

```
A sequência de caractere existe na string!
```

Também podemos verificar o oposto, isto é: se a sequência **não existe**, utilizando o operador `not in`, desta forma:

```
texto = "A tecnologia move o mundo!"

print('qualidade' not in texto)

if 'qualidade' not in texto:
    print("A palavra 'qualidade' realmente não existe dentro da
        string!")
```

A saída será `True` na primeira verificação, dado que `qualidade` realmente **não se encontra** na string em questão. A saída completa será:

```
True
A palavra 'qualidade' realmente não existe dentro da string!
```

Caracteres de saída (*Escape Characters*)

A princípio um caractere de escape, ou caracter de saída é utilizado para gerar uma interpretação alternativa ao texto que foi definido.

Em Python, eles são caracterizados pela barra inversa **** seguido por outro caractere.

Pode parecer confuso, mas vamos pra utilização que você vai entender!

Existem alguns caracteres que não podem ser inseridos em strings, pois será gerado um algum tipo de erro como por exemplo tentar adicionar aspas duplas em uma string que foi criada com aspas duplas, veja:

```
variavel = "Aprenda "o" Python "
print(variavel)
```

A saída do código acima gera um erro de sintaxe `SyntaxError` pois o Python finaliza a string após as aspas de fechamento após o texto “Aprenda”.

A maneira certa de conseguir que esse código tenha a mesma saída sem erro, é usar o caractere de saída `\"`, assim:

```
variavel = "Aprenda \"o\" Python"
print(variavel)
```

Dessa forma, o Python vai entender que você quer realmente definir aspas no texto, e não fechar uma string!

O resultado será:

```
Aprenda "o" Python
```

Este método também pode ser aplicado com aspas simples `''`. Exemplo:

```
variavel = 'Aprenda \'o\' Python'
print(variavel)
```

Acabamos de conhecer o caractere de saída **Single-Quote** (Citação única) e agora vamos conhecer outros caracteres!

Se você deseja adicionar uma contra barra sem que o Python interprete como um caractere de saída, basta utilizar duas contra barras para isso, assim:

```
variavel = "Contra \\' \\ barra"
print(variavel)
```

Resultando um uma única barra:

```
Contra \' \ barra
```

Vamos adicionar/quebrar uma linha dentro de uma string com `\n`:

```
variavel = "Desejo quebrar \n essa linha"
print(variavel)
```

Resultando na quebra de linha:

```
Desejo quebrar
 essa linha
```

O próximo é o `\t`, que nada mais é que um caractere que simboliza o TAB do seu teclado, confira:

```
variavel = "Quatro espaço \t para um tab"

print(variavel)
```

A saída será:

```
Quatro espaço    para um tab
```

Também temos o caractere de saída **backspace** `\b`. Você pode utilizá-lo para ter o mesmo efeito da tecla ***Backspace*** de seu teclado:

```
variavel = "Aa\b Bb\b Cc\b"
print(variavel)
```

Saída:

```
A B C
```

Veja que os caracteres `a`, `b` e `c` foram removidos da saída!

Representando Strings em Octal e Hexadecimal

Também temos caracteres especiais para tratamento de de valores Octais e Hexadecimais.

Os número Octais são representados por uma barra invertida e três números `\000`, dessa forma:

```
variavel = "\101 \102 \103 \104 - \060 \061 \062 \063"
print(variavel)
```

Resultando na seguinte saída:

```
A B C D - 0 1 2 3
```

Já para representar caracteres em formato **Hexadecimal**, utilizamos uma barra invertida seguida por um “x” e o número em Hexadecimal.

```
variavel = "\x61 \x62 \x63 \x64 - \x30 \x31 \x32 \x33"
print(variavel)
```

Resultando em:

```
a b c d - 0 1 2 3
```

Conclusão

Nesse post você viu vários conceitos importantes quando estamos manipulando strings em Python!

Ainda temos muito conteúdo para abordar sobre strings, então fique ligado na Python Academy que teremos mais artigos completíssimos em breve 😉

Se ficou com alguma dúvida, fique à vontade para usar o box de comentário aqui embaixo!

Terei enorme prazer em te responder!

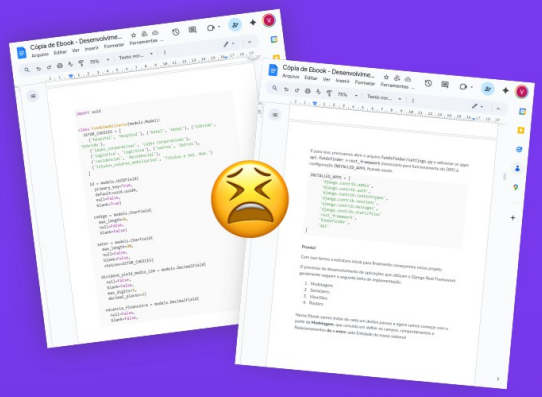
É isso meu caro, e até a próxima!

Não se esqueça de conferir!

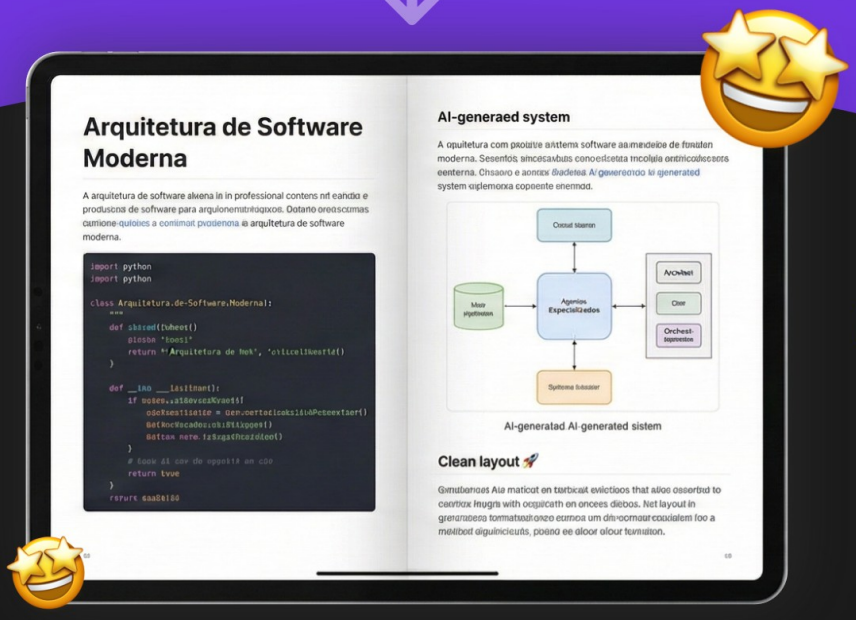
DevBook

Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!



Chega de formatar código no Google Docs



Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS