

# Aula 05

---

Tratamento de cadeias de caracteres  
(strings)



# Tópicos

---

- Introdução às Strings
- Operações Básicas com Strings
- Manipulação de Strings
- Formatando Strings
- Strings Multilinhas
- Strings como Iteráveis
- Métodos de Divisão e Junção
- Escape Characters e Raw Strings

# Introdução às String

---

# Introdução às String

---

- O que é uma string?
- Diferença entre strings e outros tipos de dados
- Como criar uma string em Python
- Notação de aspas simples e duplas

# O que é uma String?

---

- String é uma sequência de caracteres que pode incluir letras, números, símbolos e espaços. Em Python, strings são um dos tipos de dados mais comuns e são usadas para representar texto.
- Características:
  - Imutáveis: Uma vez criada, uma string não pode ser modificada. Se precisar alterar uma string, você deve criar uma nova string.
  - Índices: Cada caractere em uma string tem uma posição específica (ou índice) que começa em 0.

```
[15] palavra = "Python"  
      print(palavra[0]) # Saída: 'P'
```

⇒ P

# Diferença entre strings e outros tipos de dados

---

- Strings são usadas para representar texto ou sequências de caracteres, enquanto inteiros, flutuantes e booleanos são usados para representar números ou valores lógicos.

```
[16] # String
      frase = "123"

      # Inteiro
      numero = 123

      print(type(frase)) # Saída: <class 'str'>
      print(type(numero)) # Saída: <class 'int'>
```



```
<class 'str'>
<class 'int'>
```

# Notação de aspas simples e duplas

---

- Aspas Simples dentro de Aspas Duplas: Se sua string contém aspas simples, use aspas duplas para criar a string, e vice-versa.

```
[ ] frase = 'Ela respondeu "Sim, eu também gosto de Python!'"  
  
frase = "Ele disse 'Python é incrível!'"
```

- Aspas Duplas dentro de Aspas Simples: Da mesma forma, se sua string contém aspas duplas, use aspas simples para criar a string.

# Operações Básicas

---



# Operações básicas com String

---

- Concatenar strings (`+`)
- Repetir strings (`\*`)
- Acessar caracteres individuais (indexação)
- Fatiamento de strings (slicing)

# Concatenar String

---

- Concatenar strings significa juntar duas ou mais strings em uma única string. Em Python, isso pode ser feito usando o operador +.

✓  
0s

```
[17] str1 = "Hello"  
      str2 = "World"  
      resultado = str1 + " " + str2  
      print(resultado) # Saída: Hello World
```



Hello World

# Repetir String

---

- Você pode repetir uma string várias vezes utilizando o operador \*. Isso pode ser útil, por exemplo, para criar uma linha de caracteres repetidos.



```
str1 = "Ha"  
resultado = str1 * 3  
print(resultado)  # Saída: HaHaHa
```



HaHaHa

# Acessar caracteres individuais (indexação)

---

- Em Python, cada caractere em uma string pode ser acessado diretamente usando sua posição (índice). Os índices começam em 0, ou seja, o primeiro caractere está na posição 0, o segundo na posição 1, e assim por diante.

```
[19] str1 = "Python"
      primeiro_caractere = str1[0]
      ultimo_caractere = str1[-1]
      print(primeiro_caractere)  # Saída: P
      print(ultimo_caractere)   # Saída: n
```



P  
n

# Fatiamento de strings (slicing)

- Fatiamento é o processo de extrair uma parte da string, criando uma sub-string. Isso é feito utilizando a notação de colchetes `[]` com o formato `[inicio:fim:passo]`.

```
[20] str1 = "Python Programming"
      sub_string = str1[0:6] # Extrai "Python"
      print(sub_string) # Saída: Python

      # Outro exemplo de fatiamento com passo
      sub_string_com_passo = str1[0:12:2] # Extrai "Pto rg"
      print(sub_string_com_passo) # Saída: Pto rg
```



Python  
Pto rg

# Manipulação de String

---

# Manipulação de Strings

---

- Tamanho de uma string (função ``len()``)
- Métodos de string:
  - ``upper()``, ``lower()``, ``title()``, ``capitalize()``
  - ``strip()``, ``rstrip()``, ``lstrip()``
  - ``replace()``
  - ``find()``, ``index()``
  - ``count()``
- Verificar se uma string contém apenas números (``isdigit()``)
- Verificar se uma string contém apenas letras (``isalpha()``)

# Tamanho de uma string (Len())

---

- A função len() é uma função embutida em Python que retorna o número de itens em um objeto. Essa função pode ser usada em diferentes tipos de objetos, como strings, listas, tuplas, dicionários e outros tipos que suportam a contagem de elementos.

```
[14] frase1 = "Python é a linguagem de programação popular."  
      frase2 = "Python é divertido"  
  
      print(len(frase1))  
      print(len(frase2))
```

```
⇒ 44  
   18
```



# Métodos de String

---

- `upper()`: Converte todos os caracteres da string para maiúsculas.
- `lower()`: Converte todos os caracteres da string para minúsculas.
- `title()`: Converte a primeira letra de cada palavra em maiúscula e o restante em minúscula.
- `capitalize()`: Converte a primeira letra da string para maiúscula e o restante para minúscula.
- `strip()`: Remove espaços em branco do início e do fim da string.
- `rstrip()`: Remove espaços em branco do final da string.
- `lstrip()`: Remove espaços em branco do início da string.

# Remove espaços em branco numa string

- As funções removem espaços em branco dentro de uma string.

```
frase = "hello world"
print(frase.title())
print(frase.capitalize())
```

⇒ Hello World  
Hello world

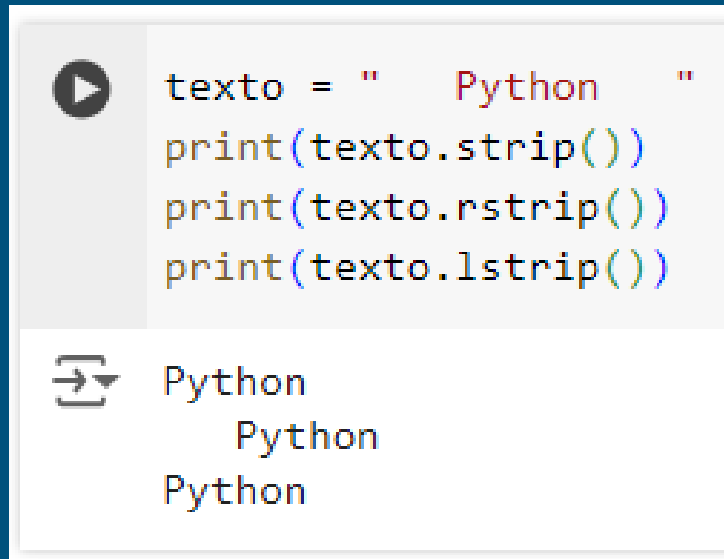
```
texto = "Python Programming"
print(texto.upper())
print(texto.lower())
```

⇒ PYTHON PROGRAMMING  
python programming

# Remove espaços em branco numa string

---

- As funções removem espaços em branco dentro de uma string.



```
texto = "  Python  "
print(texto.strip())
print(texto.rstrip())
print(texto.lstrip())
```

Python  
Python  
Python

The image shows a code execution environment. The top part contains a Python script with four lines: a string assignment and three print statements using strip, rstrip, and lstrip methods. The bottom part shows the output of these operations, which are the strings 'Python', 'Python', and 'Python' respectively, each on a new line. The first line of output is preceded by a right-pointing arrow icon.

# Métodos de String

---

- `replace()`: Substitui todas as ocorrências de uma substring por outra.
- `find()`: Retorna o índice da primeira ocorrência de uma substring. Retorna -1 se não encontrar.
- `index()`: Retorna o índice da primeira ocorrência de uma substring. Levanta um erro `ValueError` se não encontrar.
- `count()`: Conta o número de vezes que uma substring aparece na string.

# Funções replace()

- Substitui todas as ocorrências de uma substring por outra.
  - `string.replace(old, new, count)`
    - `old`: A substring que será substituída.
    - `new`: A substring que substituirá a antiga.
    - `count`: Número opcional de substituições a serem feitas.

```
frase = "banana, banana, banana"
nova_frase = frase.replace("banana", "maçã", 1)
print(nova_frase) # maçã, banana, banana
```

maçã, banana, banana

```
texto = "Python é ótimo"
novo_texto = texto.replace("ótimo", "fantástico")
print(novo_texto) # Python é fantástico
```

Python é fantástico

# Funções find() e index()

- find(): Retorna o índice da primeira ocorrência de uma substring.
- index(): Retorna o índice da primeira ocorrência de uma substring.



```
texto = "Python é divertido"
print(texto.find("divertido"))
print(texto.index("divertido"))
```




```
9
9
```

# Função count()

---

- Conta o número de vezes que uma substring aparece na string.
  - `string.count(substring, start, end)`
    - `substring`: A substring a ser contada.
    - `start`: Índice inicial opcional.
    - `end`: Índice final opcional.

```
 frase = "Python é a linguagem de programação mais popular. Python é ótimo."  
contagem = frase.count("Python", 0, 30)  
print(contagem) # 1
```

```
 1
```

# Verificar se uma String Contém

- O método `isdigit()` verifica se todos os caracteres na string são dígitos.
- O método `isalpha()` verifica se todos os caracteres na string são letras (a-z ou A-Z).



```
texto = "123a45"
if texto.isdigit():
    print("A string contém apenas números.")
else:
    print("A string contém caracteres não numéricos.")
```



A string contém caracteres não numéricos.



```
texto = "Python3"
if texto.isalpha():
    print("A string contém apenas letras.")
else:
    print("A string contém caracteres não alfabéticos.")
```



A string contém caracteres não alfabéticos.



# String como Iteráveis

---

# Operando com o 'for'

- A iteração sobre caracteres de uma string usando um loop for em Python permite percorrer cada caractere da string, um por um, executando operações ou verificações em cada um deles.

```
texto = "Programação"
vogais = "aeiouAEIOU"
contador = 0

for caractere in texto:
    if caractere in vogais:
        contador += 1

print('Total de vogais: ', contador)
```

⇒ Total de vogais: 4

```
texto = "Python"
for caractere in texto:
    print(caractere)
```

⇒ P  
y  
t  
h  
o  
n

# Operador 'in'

- O operador 'in' em Python é utilizado para verificar se um caractere ou uma substring (uma sequência de caracteres) está presente dentro de uma string. Ele retorna um valor booleano (True ou False).

```
▶ texto = "Python"
  if "P" in texto:
      print("A letra 'P' está presente na string.")
  else:
      print("A letra 'P' não está presente na string.")
```

⇒ A letra 'P' está presente na string.

```
▶ frase = "A programação em Python é interessante."
  if "Python" in frase:
      print("A palavra 'Python' está presente na frase.")
  else:
      print("A palavra 'Python' não está presente na frase.")
```

⇒ A palavra 'Python' está presente na frase.

# Métodos de Divisão e Junção

---

# Método split()

- O método split() em Python é usado para dividir uma string em várias partes, com base em um delimitador especificado. Ele retorna uma lista contendo as partes da string dividida.

```
▶ texto = "Python é uma linguagem poderosa"
partes = texto.split()
print(partes)
```

⇒ ['Python', 'é', 'uma', 'linguagem', 'poderosa']

```
▶ texto = "Maçã,Laranja,Banana,Uva"
frutas = texto.split(",")
print(frutas)
```

⇒ ['Maçã', 'Laranja', 'Banana', 'Uva']

```
▶ texto = "Python é uma linguagem poderosa"
partes = texto.split(" ", 2)
print(partes)
```

⇒ ['Python', 'é', 'uma linguagem poderosa']

# Método join()

- O método join() é usado para juntar (concatenar) uma sequência de strings (como uma lista ou tupla) em uma única string, usando um delimitador específico entre os elementos.

```
▶ palavras = ['Python', 'é', 'divertido']  
frase = " ".join(palavras)  
print(frase)
```

➡ Python é divertido

```
▶ frutas = ['Maçã', 'Laranja', 'Banana', 'Uva']  
lista_de_frutas = ", ".join(frutas)  
print(lista_de_frutas)
```

➡ Maçã, Laranja, Banana, Uva

```
▶ partes_caminho = ["C:", "Users", "Maria", "Documents", "arquivo.txt"]  
caminho = "\\".join(partes_caminho)  
print(caminho)
```

➡ C:\Users\Maria\Documents\arquivo.txt

# Escape Caracteres e Raw String

---

# As estruturas de repetição

---

- Os caracteres de escape são sequências de caracteres que não são interpretadas de maneira literal pelo Python. Eles começam com uma barra invertida (\) seguida de outro caractere. Esses caracteres permitem a inclusão de caracteres especiais em strings, como novas linhas, tabulações, aspas, etc.
- Uso de caracteres de escape:
  - \n: Nova linha.
  - \t: Tabulação.
  - \\: Barra invertida literal.
  - \': Aspa simples literal.
  - \": Aspa dupla literal.



# As estruturas de repetição

- Uso de caracteres de escape:

- `\n`: Nova linha.
- `\t`: Tabulação.
- `\\`: Barra invertida literal.
- `\'`: Aspa simples literal.
- `\"`: Aspa dupla literal.

```
# Exemplo com tabulação (\t)
string_com_tab = "Nome:\tMaria\nIdade:\t25"
print(string_com_tab)
```

```
Nome:   Maria
Idade:  25
```

```
# Exemplo com nova linha (\n)
string_com_nova_linha = "Olá, Mundo!\nBem-vindo ao Python."
print(string_com_nova_linha)
```

```
Olá, Mundo!
Bem-vindo ao Python.
```

```
# Exemplo com barra invertida literal (\\)
string_com_barra = "Este é um caminho de arquivo: C:\\Users\\Maria\\Documents"
print(string_com_barra)
```

```
Este é um caminho de arquivo: C:\Users\Maria\Documents
```

# O que são os “raw strings”

---

- As "raw strings" (strings brutas) são strings que tratam todos os caracteres da string literalmente, sem considerar caracteres de escape. Elas são definidas com um prefixo r antes da abertura das aspas da string. Isso é especialmente útil ao lidar com expressões regulares, caminhos de arquivos no Windows, ou qualquer outro caso em que os caracteres de escape possam causar problemas.
- O que são "raw strings" e quando usá-las (r'...')



```
# Exemplo com raw string  
caminho = r'C:\new_folder\teste'  
print(caminho)
```

Vamos praticar!  
Link



Vamos exercitar!  
Link

