Guia Python

String

Principais metodos de string

Em Python, as strings possuem diversos métodos embutidos que permitem manipulação, busca, formatação, e outras operações úteis. Abaixo, listo alguns dos principais métodos para strings:

1. Manipulação de Strings

- **`upper()`**: Converte todos os caracteres para maiúsculas.
- **`lower()`**: Converte todos os caracteres para minúsculas.
- **`capitalize()`**: Converte o primeiro caractere da string para maiúscula.
- **`title()`**: Converte o primeiro caractere de cada palavra para maiúscula.
- **`strip()`**: Remove espaços em branco do início e do fim da string.
- **`lstrip()`**: Remove espaços em branco do início da string.
- **`rstrip()`**: Remove espaços em branco do final da string.
- **`replace(old, new)`**: Substitui todas as ocorrências de uma substring por outra.
- **`split(separator)`**: Divide a string em uma lista, usando o separador especificado.
- **`join(iterable)`**: Junta uma lista de strings em uma única string, usando a string como separador.

2. Busca e Verificação

- **`find(substring)`**: Retorna o índice da primeira ocorrência da substring, ou -1 se não for encontrada.
- **`index(substring)`**: Semelhante a `find()`, mas gera uma exceção se a substring não for encontrada.
- **`startswith(prefix)`**: Verifica se a string começa com o prefixo especificado.
- **`endswith(suffix)`**: Verifica se a string termina com o sufixo especificado.
- **`count(substring)`**: Retorna o número de vezes que a substring aparece na string.

3. Formatação

- **`format()`**: Formata a string, substituindo placeholders `{}` por valores especificados.
- **`zfill(width)`**: Preenche a string com zeros à esquerda até atingir o comprimento especificado.
- **`center(width)`**: Centraliza a string em um espaço de largura especificada.
- **`ljust(width)`**: Alinha a string à esquerda em um espaço de largura especificada.
- **`rjust(width)`**: Alinha a string à direita em um espaço de largura especificada.

4. Outras Operações

- **`len(string)`**: Retorna o comprimento da string.
- **`isalnum()`**: Verifica se a string é alfanumérica.
- **`isalpha()`**: Verifica se a string contém apenas letras.
- **`isdigit()`**: Verifica se a string contém apenas dígitos.
- **`isspace()`**: Verifica se a string contém apenas espaços em branco.

Esses métodos são os mais comumente utilizados e cobrem uma ampla gama de necessidades ao trabalhar com strings em Python.

Maneiras de Reverter uma String

Em Python, strings são imutáveis, o que significa que não existe um método direto como `reverse()` para reverter uma string. No entanto, existem várias maneiras de reverter uma string de maneira eficiente.

1. **Usando Slicing** A maneira mais comum e simples de reverter uma string é utilizando slicing.

```
python texto = "Python" texto_revertido = texto[::-1] print(texto_revertido)
```

Saída: `nohtyP`

- O slicing `[::-1]` significa "pegue todos os caracteres da string, mas na ordem inversa".
- **Usando a Função `reversed()`** Outra maneira é usar a função `reversed()`, que retorna um iterador, e então, unir os caracteres de volta em uma string.

```
python texto = "Python" texto_revertido = ''.join(reversed(texto))
print(texto_revertido)
```

Saída: `nohtyP`

- Aqui, `reversed(texto)` retorna um iterador que percorre a string de trás para frente.
- `''.join()` é utilizado para unir os caracteres do iterador em uma nova string.
- **Usando um Loop Manual** Embora não seja a maneira mais eficiente, você também pode reverter uma string manualmente com um loop.

```
python texto = "Python" texto_revertido = "" for letra in texto:
texto_revertido = letra + texto_revertido print(texto_revertido)
```

Saída: `nohtyP`

Conclusão

Embora não exista um método `reverse()` específico para strings em Python, você pode facilmente reverter uma string usando slicing, a função `reversed()`, ou até mesmo um loop. O método mais comum e eficiente é o slicing `[::-1]`.