Revisão de Python: Strings

> ***Strings*** são usadas para armazenar uma coleção de caracteres. São **imutáveis**, o que significa que não podem ser alteradas após a criação.

- **Criação de Strings:** As strings podem ser criadas usando aspas simples ou duplas.

```
nome = 'Albert'
mensagem = "Olá, Mundo!"
```

> Python não possui um tipo de dado para caracteres individuais. Um caractere é simplesmente uma `string` de comprimento 1.

Convertendo Tipos de Dados

> Não é possível concatenar strings com outros tipos de dados. É necessário converter os outros tipos de dados em strings.

- **Conversão para String:** Outros tipos de dados podem ser convertidos em strings usando a função `str()`.

```
idade = 30
print("Olá, eu tenho " + str(idade) + " anos.")
```

- **Conversão de String para Número:** Strings que representam números podem ser convertidas em números usando as funções `int()` e `float()`.

```
numero = "10"
print(int(numero) + 5)
```

- **Número de base 2, 8, 10 e 16:** Strings que representam números em diferentes bases podem ser convertidas em números usando as funções `int()` e `float()`.

```
print(int('1001', 2))
print(int('12', 8))
print(int('123'))
print(int('AF', 16))
```

- A volta de um número para uma string em uma base específica pode ser feita usando a função `format()`.

```
print(format(9, 'b'))
print(format(9, 'o'))
print(format(9, 'd'))
print(format(9, 'x'))
```

String multi-linha

- Strings multi-linha podem ser criadas usando aspas triplas `'''` ou `"""`.

```
mensagem = '''Volta o cão arrependido
Com suas orelhas tão fartas
Com seu osso roído
E com o rabo entre as patas'''
print(mensagem)
```

> Embora seja usado para comentários de várias linhas, aspas triplas são usadas para criar strings multi-linhas. O que é um pouco confuso.

Caracteres de Escape

> São usados para representar caracteres especiais em strings.

- `\n` : Nova linha
- `\t` : Tabulação
- `\\` : Barra invertida
- `\'` : Aspas simples
- `\"` : Aspas duplas
- `\r` : Retorno de carro
- `\v` : Tabulação vertical
- `\xhh` : Caractere ASCII em hexadecimal
- `\uXXXX` : Caractere Unicode

> `\v` e `\r` podem não funcionar em todos os sistemas operacionais.

Métodos para Strings

- **Formatação:** `format()`, `f-strings`

```
nome = "Albert"
idade = 30
print("Olá, meu nome é {} e tenho {} anos.".format(nome, idade))
print(f"Olá, meu nome é {nome} e tenho {idade} anos.")
```

- **Formatação de Números:** `format()`
  - `{:.2f}` : Duas casas decimais
  - `{:.0f}` : Sem casas decimais
  - `{:10.2f}` : Duas casas decimais e 10 caracteres de largura
  - `{:.2e}` : Notação científica
  - `{:<10}` : Alinhamento à esquerda
  - `{:^10}` : Alinhamento centralizado
  - `{:>10}` : Alinhamento à direita
  - `{:0>10}` : Preenchimento com zeros
  - `{:x<10}` : Preenchimento com caracteres
  - `{:b}` : Número binário
  - `{:o}` : Número octal
  - `{:x}` `{:X}` : Número hexadecimal

- **Operadores de Comparação:** `==`, `!=`, `>`, `<`, `>=`, `<=`

```
print("abc" == "abc")
print("abc" != "abc")
print("abc" > "def")
print("abc" < "def")
```

- **Concatenação de Strings:** Duas strings podem ser concatenadas usando o operador de adição `+`.

```
a = "Olá"
b = "Mundo"
c = a + " " + b
print(c) # Saída: Olá Mundo
```

> Há um problema de desempenho ao usar o operador `+` para concatenar muitas strings. Isso ocorre porque as strings são imutáveis e cada vez que uma concatenação é feita, uma nova string é criada.

- **Repeticão de Strings:** Uma string pode ser repetida usando o operador de multiplicação `*`.

```
estrofe = '''Volta o cão arrependido
Com suas orelhas tão fartas
Com seu osso roído
E com o rabo entre as patas
'''
soneto = estrofe * 44
print(soneto)
```

- **O operador** `in` é usado para verificar se uma *substring* está presente em uma `string`.

```
print("abc" in "abcdef")
print("abc" not in "abcdef")
```

- **Iteração sobre uma String:** Uma string é uma sequência de caracteres, portanto, pode ser iterada usando um loop `for`.

```
for letra in "abc":
    print(letra)
```

- **Comprimento de uma String:** O comprimento de uma string pode ser obtido usando a função `len()`.

```
print(len("abc")) # Saída: 3
```

- **Acesso a Caracteres:** Um caractere de uma string pode ser acessado usando um índice entre colchetes `[]`.

```
s = "abc"
print(s[0]) # Saída: a
print(s[1]) # Saída: b
print(s[2]) # Saída: c
```

- **Alteração de Caixa:** `upper()`, `lower()`, `capitalize()`, `title()`, `swapcase()`

```
nome = "tutorial de Python"
print(nome.upper())
print(nome.capitalize())
print(nome.title())
print(nome.swapcase())
```

- **Métodos de Busca:**
  - `find()`, `index()`: Retorna o índice da primeira ocorrência de uma substring.
  - `count()`: Retorna o número de ocorrências de uma substring.

```
nome = "tutorial de Python"
print(nome.find("de"))
print(nome.index("de"))
print(nome.count("t"))
```

> `find()` e `index()` retornam `-1` se a substring não for encontrada, mas `find()` não gera uma exceção.

- **Métodos de Verificação:** `startswith()`, `endswith()`, `isalpha()`, `isdigit()`, `isalnum()`, `isspace()`

```
nome = "tutorial de Python"
print(nome.startswith("t")) # se inicia com "t"
print(nome.endswith("n")) # se termina com "n"
print(nome.isalpha()) # se só tem letras
print(nome.isdigit()) # se só tem números
print(nome.isalnum()) # se só possui letras e números
print(nome.isspace()) # se só tem espaços
```

> O que ocorre se a `string` for vazia?

- **Métodos de Substituição:** `replace()`, `strip()`, `lstrip()`, `rstrip()`

```
nome = "tutorial de Python"
print(nome.replace("Python", "Java")) # substitui "Python" por "Java"
print(nome.strip("t")) # remove "t" do início e do fim
print(nome.lstrip("t")) # remove "t" do início
print(nome.rstrip("n")) # remove "n" do fim
print(nome.strip()) # remove espaços do início e do fim
```

> Lembrando que *strings* são imutáveis, então esses métodos retornam uma nova string.

- **Encadeamento de Métodos:** Se um método retorna uma `string`, outro método pode ser chamado em seguida.

```
nome = "  tutorial de Python  "
print(nome.strip().replace("Python", "Java").upper())
```

- **Métodos de Separação:** `split()`, `partition()`, `rpartition()`

```
nome = "tutorial de Python"
print(nome.split()) # separa por espaços
print(nome.partition("de")) # separa na primeira ocorrência de "de"
print(nome.rpartition("de")) # separa na última ocorrência de "de"
```

- **split()** com separador personalizado

```
data = "10/05/2021"
x = data.split("/")
print(x) # Saída: ['10', '05', '2021']
```

- **split()** com separador personalizado quando há mais de um separador entre os elementos

```
data = "10//05//2021"
x = data.split("/")
print(x) # Saída: ['10', '', '05', '', '2021']
```

- **split()** com separador padrão quando há mais de um separador entre os elementos

```
data = "   10       05\t\t\t2021    "
x = data.split()
print(x) # Saída: ['10', '05', '2021']
```

- **Métodos de União:** `join()`

> O método `join()` é usado para unir uma lista de strings em uma única string. É muito útil para formatar saídas de dados e por apresentar melhor desempenho do que a concatenação de strings.

```
palavras = ["tutorial", "de", "Python"]
print(" ".join(palavras))
```

- Concatenação vs `join()` usando `%timeit`

```
palavras = ["tutorial", "de", "Python"]
%timeit x = " ".join(palavras)
%timeit x = palavras[0] + " " + palavras[1] + " " + palavras[2]
```

- **Métodos de Formatação:** `ljust()`, `rjust()`, `center()`, `zfill()`

```
nome = "Python"
print(nome.ljust(10)) # alinhado à esquerda
print(nome.rjust(10)) # alinhado à direita
print(nome.center(10)) # centralizado
print("123".zfill(5)) # preenchido com zeros
```

- **Métodos de Verificação de Caixa:** `islower()`, `isupper()`, `istitle()`

```
nome = "Python"
print(nome.islower())
print(nome.isupper())
print(nome.istitle())
```

- **r-strings:** `r''`, `r""`
- As *r-strings* são usadas para representar *strings* brutas. Caracteres de escape não são interpretados.

```
print(r"\n \t \r \\ ")
```

Exercícios

1. Escreva um programa que leia uma string e exiba o número de vogais e consoantes na string.
2. Escreva um programa que leia uma string e exiba o número de palavras na string.
3. Escreva um programa que leia uma string e exiba a string sem as vogais.
4. Escreva um programa que leia uma string e exiba a string sem as consoantes.
5. Escreva um programa que leia uma string como no exemplo a seguir e exiba e calcule a soma dos números na string. `1,20 R$ 1,50 R$ 2,00 R$ 3,00 R$ 4,00 R$ 5,00 R$ 6,00 R$ 7,00 R$ 8,00 R$ 9,00 R$ 10,00 R$`