

# Manipulação de Strings



# Strings

- Representam informação textual

```
nome = 'Maria Silva'
```

```
nacionalidade = 'brasileira'
```

```
nome_mae = 'Ana Santos Silva'
```

```
nome_pai = 'Jonas Nunes Silva'
```

# Acesso a conteúdo das Strings

- Acesso pode ser feito pelo nome da variável que contém a string

```
nome = 'Maria Silva'  
print(nome)
```

# Acesso a conteúdo das Strings

- Caracteres podem ser acessados pela sua posição dentro da String
- A primeira posição tem índice zero

```
nome = 'Maria Silva'
```

```
print(nome[0]) → M
```

```
print(nome[6]) → S
```

	0	1	2	3	4	5	6	7	8	9	10
nome	M	a	r	i	a		S	i	l	v	a

# Acesso a conteúdo das Strings

- É possível acessar também uma substring usando [início:fim]
- A substring retornada vai de início (inclusive) até fim – 1
- Se início for omitido, significa zero
- Se fim for omitido, significa len(string)

```
nome = 'Maria Silva'  
print(nome[2:5]) → ria  
print(nome[:5]) → Maria  
print(nome[6:]) → Silva
```

	0	1	2	3	4	5	6	7	8	9	10
nome	M	a	r	i	a		S	i	l	v	a

# Alteração

- O conteúdo de uma determinada posição de uma string não pode ser alterado – são sequências imutáveis

```
nome = 'Maria Silva'
```

```
nome[3] = 't'
```

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**TypeError: 'str' object does not support  
item assignment**

# Operadores

- Alguns operadores que podem ser usados em strings
  - in
  - len
  - +
  - \*

# in

- substring **in** string
  - Retorna True ou False

```
nome = 'Maria Silva'
```

```
print('M' in nome) → True
```

```
print('B' in nome) → False
```

```
print('m' in nome) → False
```

```
print('ria' in nome) → True
```

# len

- `len(string)`
  - Retorna a quantidade de caracteres da string

```
nome = 'Maria'
```

```
print(len(nome)) → 5
```

```
nome = 'Maria Silva'
```

```
print(len(nome)) → 11
```

# + (Concatenação)

- `string1 + string2`
  - Concatena duas strings

```
nome = 'Maria' + 'Silva'
```

```
print(nome) → MariaSilva
```

```
nome = 'Maria'
```

```
sobrenome = 'Silva'
```

```
nome_completo = nome + ' ' + sobrenome
```

```
print(nome_completo) → Maria Silva
```

# \* (Repetição)

- `string * int`
  - Repete a string `int` vezes

```
nome = 'Maria'
```

```
nome_repetido = nome * 2
```

```
print(nome_repetido) → MariaMaria
```

# Percorrendo uma String

- Os elementos de uma string podem ser acessados usando uma estrutura de repetição

```
nome = 'Maria Silva'  
i = 0  
while i < len(nome):  
    print(nome[i])  
    i += 1
```

```
nome = 'Maria Silva'  
for i in range(len(nome)):  
    print(nome[i])
```

```
nome = 'Maria Silva'  
for letra in nome:  
    print(letra)
```

# F-String

- Tipo especial de String que facilita a incorporação de variáveis
- É criada com f'...'
- Permite acessar expressões usando {...}
- Exemplo:

```
palavra = 'paralelepipedo'  
texto = f'A palavra {palavra} tem {len(palavra)} letras!'  
print(texto)
```

# F-String

- Nem sempre as expressões numéricas aparecem como gostaríamos

```
numerador = 5  
denominador = 3  
divisao = numerador / denominador  
print(f'{numerador} / {denominador} = {divisao}!')
```

→ **5 / 3 = 1.6666666666666667!**

# F-String

- É possível formatar as expressões, indicando o número de casas decimais

```
divisao = 5 / 3
```

```
print(f'{divisao}') → 1.6666666666666667
```

```
print(f'{divisao:.5f}') → 1.66667
```

```
print(f'{divisao:.2f}') → 1.67
```

# Algumas operações sobre Strings

- upper
- lower
- find
- strip

# upper

- `string.upper()`
  - Retorna a string com letras minúsculas substituídas por maiúsculas
  
- A string original não é modificada!

# upper

```
texto = 'Quem parte e reparte, fica com  
a maior parte'
```

```
print(texto.upper())
```

→ **'QUEM PARTE E REPARTE FICA COM A  
MAIOR PARTE'**

# lower

- `string.lower()`
  - Retorna a string com letras maiúsculas substituídas por minúsculas
  
- A string original não é modificada!

# lower

```
texto = 'Quem parte e reparte, fica com  
a maior parte'
```

```
print(texto.lower())
```

```
→ 'quem parte e reparte, fica com a  
maior parte'
```

# find

- `string.find(substring, inicio, fim)`
  - Retorna o índice da primeira ocorrência da **substring** dentro da **string**, a partir da posição **início**, até a posição **fim-1**
  - Retorna **-1** se a substring não for encontrada
  - **início** e **fim** são opcionais
  - Não é possível informar **fim** sem informar **início**

# find

```
texto = 'A humildade é o sólido fundamento de  
todas as virtudes'
```

```
print(texto.find('da'))           → 7
```

```
print(texto.find('da', 27, 50))  → 39
```

```
print(texto.find('da', 27, 35))  → -1
```

# strip

- `string.strip()`
  - Retorna uma string com os espaços do início e do fim da string original removidos
  - A string original não é modificada!

```
texto = '  Maria  '  
print(texto.strip()) → Maria
```

# Exemplo

- Programa para gerar a citação a partir de um nome
  - Ex.: Leonardo Gresta Paulino Murta → MURTA, L. G. P.

```
nome = input('Entre com um nome completo: ').strip()
```

```
iniciais = ''
```

```
inicio = 0
```

```
fim = nome.find(' ', inicio)
```

```
while fim != -1:
```

```
    iniciais += nome[inicio] + '. '
```

```
    inicio = fim + 1
```

```
    fim = nome.find(' ', inicio)
```

```
sobrenome = nome[inicio:len(nome)].upper()
```

```
print(sobrenome + ', ' + iniciais.upper().strip())
```

# Exercícios

1. Escreva um programa que lê uma frase, uma palavra antiga e uma palavra nova. O programa deve imprimir a frase com as ocorrências da palavra antiga substituídas pela palavra nova.
  - Exemplo:
    - Frase: “Quem parte e reparte fica com a maior parte”
    - Palavra antiga: “parte”
    - Palavra nova: “parcela”
    - Saída: “Quem parcela e reparte fica com a maior parcela”

# Exercícios

2. Faça um programa que lê uma frase e retorna o número de palavras que a frase contém. Considere que a palavra pode começar e/ou terminar por espaços.

3. Faça um programa que recebe uma frase e substitui todas as ocorrências de espaço por “#”.

# Exercícios

4. Faça um programa que decida se duas strings lidas do teclado são palíndromas mútuas, ou seja, se uma é igual à outra quando lida de traz para frente.

Exemplo: **amor** e **roma**.

5. Um anagrama é uma palavra que é feita a partir da transposição das letras de outra palavra ou frase. Por exemplo, “Iracema” é um anagrama para “America”. Escreva um programa que decida se uma string é um anagrama de outra string, ignorando os espaços em branco. O programa deve considerar maiúsculas e minúsculas como sendo caracteres iguais, ou seja, “a” = “A”.

# Exercícios

6. Faça um programa que leia o nome do usuário e mostre o nome de trás para frente, utilizando somente letras maiúsculas.

Exemplo:

Nome = Vanessa

Resultado gerado pelo programa:

ASSENAV

# Exercícios

7. Faça um programa que leia o nome do usuário e o imprima na vertical, em forma de escada, usando apenas letras maiúsculas.

Exemplo:

Nome = Vanessa

Resultado gerado pelo programa:

V

VA

VAN

VANE

VANES

VANESS

VANESSA

# Exercícios

8. Faça um programa que leia uma data de nascimento no formato dd/mm/aaaa e imprima a data com o mês escrito por extenso.

Exemplo:

Data = 20/02/1995

Resultado gerado pelo programa:

20 de fevereiro de 1995

# Exercícios

- 9. Faça um programa para justificar um texto com um número de colunas informado pelo usuário. Por exemplo, para o texto “Este é um exemplo de texto que vamos justificar usando o nosso programa.” quando justificado em 18 colunas, teríamos:

```
Este é um exemplo  
de texto que vamos  
justificar usando  
o nosso programa.
```

# Referências

- Slides feitos em conjunto com Aline Paes e Vanessa Braganholo

# Manipulação de Strings

