Manipulação de Strings

Vanessa Braganholo
vanessa@ic.uff.br

Strings

- Representam informação textual

```
nome = “Maria Silva”
nacionalidade = “brasileira”
nome_mae = “Ana Santos Silva”
nome_pai = “Jonas Nunes Silva”
```

Acesso a conteúdo das Strings

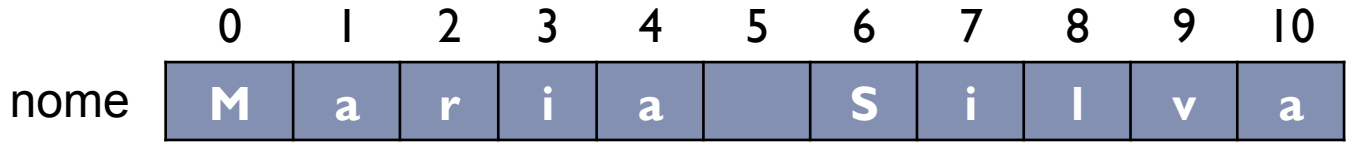
- Acesso pode ser feito pelo nome da variável que contém a string

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

Acesso a conteúdo das Strings

- String pode ser tratada como uma lista
- Caracteres podem ser acessados pela sua posição dentro da String

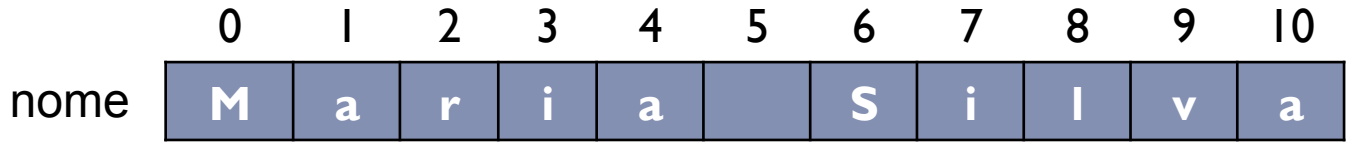
```
>>> nome = "Maria Silva"
>>> print(nome[0])
M
>>> print(nome[6])
S
```



Acesso a conteúdo das Strings

- Fatias também podem ser usadas

```
>>> nome = "Maria Silva"
>>> print(nome[:5])
Maria
>>> print(nome[6:])
Silva
```



Alteração

- Diferentemente das listas, o conteúdo das strings 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 atuam sobre sequências podem ser usados em strings
  - in
  - len
  - +
  - *

in

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

```
>>> nome = “Maria Silva”
>>> “M” in nome
True
>>> “B” in nome
False
>>> “m” in nome
False
```

len

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

```
>>> nome = "Maria"
>>> len(nome)
5
>>> nome = "Maria Silva"
>>> len(nome)
11
```

+ (Concatenação)

- string1 + string2
  - Concatena duas strings

```
>>> nome = “Maria” + “Silva”
>>> nome
MariaSilva
>>> nome = “Maria”
>>> sobrenome = “Silva”
>>> nome_completo = nome + sobrenome
>>> nome_completo
MariaSilva
```

* (Repetição)

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

```
>>> nome = “Maria”
>>> nome_repetido = nome * 2
>>> nome_repetido
MariaMaria
```

Percorrendo uma String

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

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

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

Operações sobre Strings

- count
- index
- find
- partition
- join
- split
- strip
- replace

count

- string.count(substring[, inicio[, fim]])
  - Retorna a quantidade de vezes que a **substring** aparece dentro da **string**, procurando a partir da posição **início** até a posição **fim -1**

count

```
>>> texto = “A humildade é o sólido
fundamento de todas as virtudes”
>>> texto.count(‘i’)
3
>>> texto.count(‘i’, 10)
2
>>> texto.count(‘i’, 10, 20)
1
>>> texto.count(‘da’)
3
```

index

- string.index(substring[, inicio[, fim]])
  - Retorna o índice da primeira ocorrência da **substring** dentro da **string**, a partir da posição **início**, até a posição **fim-1**

index

```
>>> texto = "A humildade é o sólido fundamento de todas as virtudes"
>>> texto.index('l')
6
>>> texto.index('l', 10)
18
>>> texto.index('da')
7
>>> texto.index('da', 27, 50)
39
>>> texto.index('da', 27, 35)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

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**
  - Diferentemente de **index**, retorna **-1** se a substring não for encontrada

find

```
>>> texto = “A humildade é o sólido
fundamento de todas as virtudes”
>>> texto.find(‘da’)
7
>>> texto.find(‘da’, 27, 50)
39
>>> texto.find(‘da’, 27, 35)
-1
```

partition

- string.partition(separador)
  - Divide uma string em 3 partes
    - O que vem antes do separador
    - O separador
    - O que vem depois do separador
  - Caso o separador não seja encontrado na string, retorna a string original seguida de duas strings vazias

- A string original não é modificada!

partition

```
>>> texto = “10/12/2015”
>>> texto.partition(‘/’)
(‘10’, ‘/’, ‘12/2015’)
>>> texto.partition(‘*’)
(’10/12/2015’, ‘’, ‘’)
```

join

- separador.join(sequencia)
  - Retorna uma string com todos os elementos da **sequencia** concatenados, mas com o **separador** entre cada elemento. Os elementos da sequencia devem ser strings.

- A string original não é modificada!

join

```
>>> '-'.join(('1', '2', '3', '4', '5'))
'1-2-3-4-5'
>>> ''.join(('par', 'te'))
'parte'
>>> 'A'.join(('1', '2'))
'1A2'
```

split

- string.split(separador)
  - Retorna uma lista com as substrings presentes entre cópias da string separador
  - Se separador não for especificado, é assumido sequências de caracteres em branco, tabs ou newlines

- A string original não é modificada!

split

```
>>> nome = “Maria Silva”
>>> nome.split()
[‘Maria’, ‘Silva’]
>>> ‘/usr/bin/python’.split(‘/’)
[‘’, ‘usr’, ‘bin’, ‘python’]
```

strip

- string.strip(ch)
  - Retorna uma cópia da string, removendo os caracteres de ch do início e do fim da string
  - Se ch não for especificado, retira caracteres em branco ou especiais
- string.rstrip(ch)
  - Retira caracteres à direita
- string.lstrip(ch)
  - Retira caracteres à esquerda

- A string original não é modificada!

strip

```
>>> "xxx yyy zzz xxx".strip("xy ")
'zzz'
>>> "  xxx  ".rstrip()
"  xxx"
>>> "  xxx  ".lstrip()
"xxx  "
>>> "  xxx  ".strip()
"xxx"
```

replace

- string.replace(velha, nova, n)
  - Retorna uma nova string com as **n** ocorrências da substring **velha** substituídas por **nova**
  - Se **n** não for definido, todas as substituições são feitas

- A string original não é modificada!

replace

```
>>> s = “Quem parte e reparte, fica com a maior parte”
>>> s.replace(“parte”, “parcela”)
“Quem parcela e reparcela, fica com a maior parcela”
>>> s.replace(“parte”, “parcela”, 2)
“Quem parcela e reparcela, fica com a maior parte”
```

Exercícios

1. Escreva uma função que recebe uma frase e uma palavra antiga e uma palavra nova. A função deve retornar uma string contendo a frase original, mas com a última ocorrência da palavra antiga substituída pela palavra nova. A entrada e saída de dados deve ser feita no programa principal.

- Exemplo:
  - Frase: "Quem parte e reparte fica com a maior parte"
  - Palavra antiga: "parte"
  - Palavra nova: "parcela"
  - Resultado a ser impresso no programa principal: "Quem parte e reparte fica com a maior parcela"

Exercícios

2. Faça uma função que recebe uma string que representa uma cadeia de DNA e gera a cadeia complementar. A entrada e saída de dados deve ser feita pelo programa principal.

- Exemplo:
  - Entrada: AATCTGCAC
  - Saída: TTAGACGTG

Exercícios

3. Faça uma função que recebe uma frase e retorna o número de palavras que a frase contém. Considere que a palavra pode começar e/ou terminar por espaços. A entrada e saída de dados deve ser feita no programa principal.

4. Faça uma função que recebe uma frase e substitui todas as ocorrências de espaço por "#", sem usar a função replace. Faça também uma função para realizar a entrada de dados. A saída de dados deve ser feita no programa principal.

Exercícios

5. 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**.

6. 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.

Referências

- Slides de Aline Paes

Manipulação de Strings

Vanessa Braganholo
vanessa@ic.uff.br