

# MC-102 — Aula 11

## Strings

Instituto de Computação – Unicamp

11 de Abril de 2018

# Roteiro

## 1 Strings

- Strings; operações, funções e métodos

## 2 Processamento de Texto

## 3 Exercícios

# Strings

- Strings em Python são listas imutáveis de caracteres.
- Strings são representadas por sequências de caracteres entre aspas simples ' ou entre aspas duplas ''.

```
>>> a = "Qwerty de Asdf"
>>> a
'Qwerty de Asdf'
>>> a = 'Qwerty de Asdf'
>>> a
'Qwerty de Asdf'
>>> c = 'Joe\'s Garage'
>>> c
"Joe's Garage"
```

# Strings

- Strings são listas imutáveis, portanto pode-se acessar posições de uma string da forma usual:

```
>>> a='querty'
>>> a
'querty'
>>> a[2]
'e'
>>> a[-1]
'y'
```

# Strings

- Strings são imutáveis:

```
>>> a='qwerty'
>>> a[2]
'e'
>>> a[2]='z'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

- O caractere '\n' pode fazer parte de uma string e ele só causa a mudança de linha no comando **print**.

```
>>> aa='abc\ndef'
>>> aa
'abc\ndef'
>>> print(aa)
abc
def
```

# Strings: operações, funções e métodos

- O operador `+` concatena 2 strings, e o operador `*` repete a concatenação (como em listas).

```
>>> 'qwerty'+'poiuy'
'qwertypoiuy'
>>> 3*'abc'
'abcbabcbabc'
```

- A função **slice** devolve a string entre duas posições dadas.

```
>>> a='qwerty'
>>> a[2:4]
'er'
```

- A string vazia é representada como `' '`.

# Strings como listas (imutáveis)

- Strings podem ser processadas como listas, podendo por exemplo ter seus elementos percorridos num laço **for**.
- Exemplo: Ler uma string e imprimir a inversa desta:

```
st = input("Digite um texto:")  
inv = ''  
for x in st:  
    inv = x + inv  
print(inv)
```

- Note que cada caractere, em ordem, é adicionado no início de **inv** de tal forma que o último caractere de **st** será o primeiro de **inv**.

# Strings: operações, funções e métodos

- A função **input** (já vista) lê e retorna uma string.
- O método **strip** retorna uma string sem os brancos e mudança de linhas no início e final de uma string.

```
>>> aa=' \n abcndef \n'
```

```
>>> aa
```

```
' \n abcndef \n'
```

```
>>> print(aa)
```

```
abcndef
```

```
>>> aa.strip()
```

```
'abcndef'
```



# Strings: operações, funções e métodos

- O operador **in** verifica se uma substring é parte de uma outra string.

```
>>> 'tho' in 'python'
True
>>> 'thor' in 'python'
False
```

- O método **find** retorna onde a substring começa na string.

```
>>> p='python'
>>> p.find('tho')
2
>>> p.find('thor')
-1
```

- O método **find** retorna -1 quando a substring não ocorre na string (veja o exemplo acima).

# Strings: operações, funções e métodos

- O método **split(sep)** separa uma string usando **sep** como separador. Retorna uma lista das substrings.

```
>>> a="1; 2 ; 3"
>>> a.split(';')
['1', ' 2 ', ' 3']
```

- O método **split()** separa usando espaço '\n' e tab como **sep**.

```
>>> b="ouviram do ipiranga margens"
>>> b.split()
['ouviram', 'do', 'ipiranga', 'margens']
```

- Note que podem haver substrings vazias no retorno de **split()**.

```
>>> a="1;2;;3"
>>> a.split(';')
['1', '2', '', '3']
```

# Strings: operações, funções e métodos

- O método **replace** serve para trocar todas as ocorrências de uma substring por outra em uma string.

```
>>> a="abcabcdnfgabc abc a b c"
>>> a.replace("abc","")
'dfg  a b c'
```

- Podemos usar a função **list** criar uma lista onde os itens da lista correspondem aos caracteres da string.

```
>>> a="abc\n;abc"
>>> list(a)
['a', 'b', 'c', '\n', ';', 'a', 'b', 'c']
```

- O método **join** recebe como parâmetro uma sequência ou lista, e retorna uma string com a concatenação dos elementos da sequência/lista.

```
>>> a="abc\n;abc"
>>> l = list(a)
>>> l
['a', 'b', 'c', '\n', ';', 'a', 'b', 'c']
>>> "".join(l)
'abc\n;abc'
```

# Processamento de Texto

- Como exemplo de funções com strings vamos implementar uma funcionalidade básica de processadores de texto: contar o número de palavras em um texto.

# Processamento de Texto

- Primeiramente removemos do texto todos os sinais de pontuação.
- Depois usamos a função split para separar as palavras.

```
st = input("Digite um texto:")
pontuacao = [".", ",", ":", ";", "!", "?"]
for pont in pontuacao: #remove os sinais de pontuação
    st = st.replace(pont, " ")

numPal = len(st.split()) #split devolve lista com palavras como itens
print("Num. palavras:", numPal)
```

# Exercício

- Escreva um programa que lê uma string, e imprime “Palindromo” caso a string seja um palindromo e “Nao Palindromo” caso contrário.
- OBS: Um palindromo é uma palavra ou frase, que é igual quando lida da esquerda para a direita ou da direita para a esquerda (espaços em brancos são descartados). Assuma que a entrada não tem acentos e que todas as letras são minúsculas
- Exemplo de palindromo: “saudavel leva duas”
- Faça uma nova versão que aceita como palindromo mesmo que as letras correspondentes sejam maiúsculas e minúsculas. Assim “Saudavel Leva DUas” deve ser também um palindromo.

# Exercício

- O usuário entra cinco números separados por brancos. Imprima a média deles.
- O usuário entra com vários números separados por branco ou vírgula, por exemplo “3,4 5 6, 9” . Imprima a média deles.