MC-102 — Aula 12
Strings

Eduardo C. Xavier

Instituto de Computação – Unicamp

26 de Outubro de 2020

Roteiro

Strings em Python

- Strings (tipo `str`) em Python são objetos imutáveis que representam uma sequencia 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'
```

Strings em Python

- Para que o caractere aspas simples ’ ou aspas duplas ’’ apareçam como parte da string, é necessário incluir antes deles uma barra (\).

```
>>> c = 'Joe\'s Garage'
>>> c
"Joe's Garage"
>>> s = "Maria disse \"Ave Maria\""
>>> s
'Maria disse "Ave Maria"'
```

- A string vazia é representada como ’ ’.

```
>>> s = ''
>>> s
''
```

Strings em Python

- O caractere especial \n faz o cursor pular uma linha.

```
>>> s = "Eu não sou daqui também marinheiro\nMas eu venho de longe ainda"
>>> print(s)
Eu não sou daqui também marinheiro
Mas eu venho de longe ainda
```

- O caractere especial \t corresponde a uma tabulação.

```
>>> s = "Eu não sou daqui\t Mas eu venho de longe ainda"
>>> print(s)
Eu não sou daqui         Mas eu venho de longe ainda
```

Strings em Python

- As formas de acesso a itens de uma lista que vimos anteriormente, também funcionam com strings.
- Por exemplo, podemos acessar caracteres específicos da string com um índice inteiro:

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

- Podemos usar **slicing** com strings:

```
>>> a = 'qwerty'
>>> a[2:5]
'ert'
```

Strings em Python

- Lembre-se que strings são **imutáveis** ao contrário de listas:

```
>>> l = ['a', 'b', 'c', 'd']
>>> l[1] = 'e'
>>> l
['a', 'e', 'c', 'd']
>>> s = 'abcd'
>>> s[1]
'b'
>>> s[1] = 'e'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Strings: formatação

- Vimos em aulas anteriores como formatar a saída de uma string incluindo números inteiros (%d) ou números ponto flutuante (%f).
- Podemos usar o método **format** de strings onde uma nova string é criada a partir da string base e da formatação especificada.
- Criamos uma string com *placeholders {}* que dizem como formatar o texto a ser inserido naquele ponto. Isto funciona basicamente como os *placeholders* %d e %f vistos anteriormente.

```
>>> s1 = "Um int %d e um float %.2f" %(5,3.1415)
>>> s1
'Um int 5 e um float 3.14'
>>> s2 = "Um int {} e um float {}".format(5, 3.1415)
>>> s2
'Um int 5 e um float 3.1415'
```

Strings: formatação

- Podemos usar indicadores de posição em para indicar qual elemento na chamada de **format** deverá entrar naquele ponto.

```
>>> s2 = "Um int {0}, um float {1} e uma str {2}".format(5, 3.1415, "chá")
>>> s2
'Um int 5, um float 3.1415 e uma str chá'
```

- Agora alterando a ordem.

```
>>> s2 = "Um int {1}, um float {2} e uma str {0}".format(5, 3.1415, "chá")
>>> s2
'Um int 3.1415, um float chá e uma str 5'
```

Strings: formatação

- Os especificadores %d e %f tem seus equivalentes :d e :f no método **format**.

```
>>> s2 = "Um int {:d}, um float {:f}".format(5, 3.1415)
>>> s2
'Um int 5, um float 3.141500'
```

- Em números float podemos especificar o número de casas decimais, como já visto.

```
>>> s2 = "Um int {:d}, um float {:.2f}".format(5, 3.1415)
>>> s2
'Um int 5, um float 3.14'
```

Strings: formatação

- Podemos especificar o alinhamento da formatação com :Nd, onde N é o número total de caracteres a ser impresso incluindo os do número.

```
>>> s2 = "Um int {:d}".format(54321)
>>> s2
'Um int 54321'
>>> s2 = "Um int {:10d}".format(54321)
>>> s2
'Um int       54321'
```

- O mesmo vale para números float com formatação :N.Mf.

```
>>> s2 = "Um float {:f}".format(3.141516)
>>> s2
'Um float 3.141516'
>>> s2 = "Um float {:10.2f}".format(3.141516)
>>> s2
'Um float       3.14'
```

Strings: operações, funções e métodos

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

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

- A função **len** devolve o tamanho da string.

```
>>> s = 'abcdef'
>>> len(s)
6
>>> s[0:6]
'abcdef'
```

Strings: operações, funções e métodos

- A função **input** (já vista) lê uma string do teclado e a devolve.
- 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 método **find** retorna onde uma determinada substring começa na string, e devolve -1 quando a substring não ocorre na string.

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

- Podemos especificar posições de início e fim para busca em **find**:

```
>>> s = 'ola abc ola'
>>> s.find('ola', 2)
8
>>> s.find('ola', 9,len(s))
-1
```

Strings: operações, funções e métodos

- O método **split('sep')** separa uma string usando a string **sep** como separador. O método devolve uma lista das substrings separadas:

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

- O método **split()** sem especificação do separador, executa a separação onde ocorre os caracteres: espaço, '\n', e tab ('\t').

```
>>> s = 'Ouviram\ndo\tipiranga as margens'
>>> s.split()
['Ouviram', 'do', 'ipiranga', 'as', '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.

```
>>> s = "abc teste abc teste abc"
>>> s.replace('abc', 'oi')
'oi teste oi teste oi'
```

- Podemos especificar o número máximo de substrings a sofrerem a troca.

```
>>> s = "abc teste abc teste abc"
>>> s.replace('abc', 'oi', 2)
'oi teste oi teste abc'
```

Strings: operações, funções e métodos

- Podemos usar a função **list** para transformar uma string em 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 sequencia/lista.

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

Strings: operações, funções e métodos

- Você pode testar se duas strings são iguais ou diferentes com os operadores == e !=

```
>>> s1 = 'mexirica'
>>> s2 = 'tangerina'
>>> s1 == s2
False
>>> s1 != s2
True
```

- O operador **in** verifica se uma string faz parte da outra.

```
>>> s1 = "eu não sou daqui também marinheiro"
>>> "não" in s1
True
>>> "mar" in s1
True
>>> "marinheira" in s1
False
```

Strings: operações, funções e métodos

Resumo de funções e métodos em strings:

- **len(s)**: devolve o tamanho de **s**.
- **s1 + s2**: devolve string que é concatenação de **s1** com **s2**.
- **s.strip()**: devolve string sem *white* caracteres do inicio e fim de **s**.
- **s.find(s2, begin=0, end=len(s))**: devolve a primeira posição de ocorrência de **s2** em **s**, e -1 caso não ocorra.
- **s.split(sep=")**: devolve lista com strings de **s** utilizando **sep** para quebra da string.
- **s.replace(s1, s2, max)**: devolve string onde todas (ou max) ocorrências de **s1** são trocadas por **s2**.
- **list(s)**: devolve uma lista onde cada item é um caractere de **s**.
- **s.join(l)**: devolve uma string resultante da concatenação de todos os itens da lista **l**.
- **s1 in s2**: testa se **s1** ocorre em **s2**.

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**.

Processamento de Texto

- Como exemplo de funções com strings vamos implementar a seguinte funcionalidade básica de processadores de texto:
  1. Contar o número de palavras em um texto.
- Assuma que o texto pode ter sinais de pontuação e usados sem espaços como por exemplo:

```
Não gosto de banana, laranja,uva;cebola e alho.
```

Processamento de Texto: Contar palavras

- Primeiramente removemos do texto todos os sinais de pontuação.
- Depois usamos a função split para separar as palavras.
- Finalmente o tamanho da lista é o total de palavras!

```
s = input()
pontuacao = [',', ';', '.', '!', '?', '...', ':']
for p in pontuacao:
  s = s.replace(p, ' ')#troca pontuacao por espaço

print('String sem pontuação: {}'.format(s))
l = s.split()#devolve lista com palavras como itens
print('Total de palavras: {}'.format(len(l)))
```

Exercício

- Escreva um programa que lê uma string de até 50 caracteres, e imprime “Palíndromo"caso a string seja um palíndromo e “Nao Palíndromo"caso contrário.
- OBS: Um palíndromo 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 as palavras são todas em minúsculas e sem acentos.
- Exemplo de palíndromo: saudavel leva duas.
- Faça uma nova versão que aceita como palíndromo mesmo que as letras correspondentes sejam maiúsculas e minúsculas. Assim “Saudavel Leva DUas” deve ser também um palíndromo.
- Faça uma nova versão que aceita como palíndromo mesmo que as letras possuam acentos. Assim “Saudável Leva DUas” deve ser também um palíndromo.

Exercício

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