MC-102 — Aula 09
Listas, Strings e tuplas

Instituto de Computação – Unicamp

3 de Setembro de 2019

Listas - o que a gente já sabe

- Uma lista: **a = [3, 6, "abc", [96.7, 55]]**
- a lista vazia **[]**
- acessar um elemento da lista **a[0]** (primeiro elemento), **a[-1]** ultimo, **a[7]** da erro
- tamanho de uma lista **len(a)**
- insere um elemento no fim da lista **a.append(45.6)**

Listas - o que a gente já sabe

- **for x in lista:** x assume o valor de cada um does elementos da lista
- **for i in range(len(lista)):** i assume os valores de 0 a tamanho da lista - 1, isto é os indices dos elementos da lista

Slice

- Listas em Python suportam uma operação conhecida como **slicing**, que consiste em obter uma sub-lista contendo os elementos de uma posição inicial até uma posição final de uma lista.
- identificador **a[ind1:ind2]** é uma sub-lista com os elementos de *ind*1 até *ind*2 − 1. de **a**

```
>>> notas= [4.5, 8.6, 9, 7.8, 7]
>>> notas[1:4]
```

- **a[:ind2]** indica a sublista do começo ate **ind2**
- **a[ind1:]** indica a sublista de **ind1** ate o final

Slice

- voce pode atribuir uma nova sublista para um slice
- o slice original e o novo não precisam ter o mesmo tamanho!

```
>>> notas= [4.5, 8.6, 9, 7.8, 7]
>>> notas[1:4] = [10.0, 10.0]
>>> notas
[4.5, 10, 10,  7]
```

Concatenacao

- A operação de soma em listas gera uma nova lista que é o resultado de "grudar" **lista2** ao final da **lista1**. Isto é conhecido como **concatenação** de listas.

lista1 + lista2

```
>>> lista1=[1,2,4]
>>> lista2=[27,28,29,30,33]
>>> x=lista1+lista2
>>> x
[1, 2, 4, 27, 28, 29, 30, 33]
>>> lista1
[1, 2, 4]
>>> lista2
[27, 28, 29, 30, 33]
```

- Veja que a operação de concatenação não modifica as listas originais.

in - testa por pertinencia

- o teste **elem in lista** retorna **True** se o elemento **elem** é membro da lista

```
>>> x=[40,30,10,40]
>>> 10 in x
>>> True
>>> 20 in x
>>> False
```

outros testes

- o == funciona para listas (todos os elementos tem que ser iguais)
- o > funciona para listas, mas testa so o primeiro, e se forem iguais o segundo, e se iguais o terceiro, etc. (Ordem lexicográfica, ou ordem do dicionário)

```
>>> x=[40,30,10,40]
>>> y =[103,2]
>>> x < y
>>> True
>>> y=[40,30,100, 1,1, 1,1 1]
>>> x < y
>>> True
>>> y=[9]
>>> x < y
>>> False
```

Mais informação sobre listas

- a pagina da documentação do Python sobre listas `https://docs.python.org/3/tutorial/datastructures.html#more-on-lists`

CUIDADO - listas não são copiadas

```
>>> a=[1,2,3]
>>> b=a
>>> a.append(47)
>>> a
[1, 2, 3, 47]
>>> b
[1, 2, 3, 47]
>>> b.append(99)
>>> b
[1, 2, 3, 47, 99]
>>> a
[1, 2, 3, 47, 99]
```

- **a** e **b** são a **mesma** lista (por causa da operação **b=a**)

CUIDADO - listas não são copiadas

```
>>> a=[1,2,3]
>>> b=[1,2,3]
>>> a.append(47)
>>> a
[1, 2, 3, 47]
>>> b
[1, 2, 3]
```

- listas que voce escreve no programa são listas "novas"

CUIDADO - como copiar uma lista

```
>>> a=[1,2,3]
>>> b=a[:]
>>> a.append(47)
>>> a
[1, 2, 3, 47]
>>> b
[1, 2, 3]
```

Objetos mutáveis e imutáveis

- Cada objeto criado em Python (um **int**, **float**, **list**, etc) é classificado como mutável ou imutável.
- Os objetos do tipo **int**, **float**, **string**, e **bool** são imutáveis. Isto significa que objetos deste tipo não podem ter seus valores alterados.
- Cada objeto criado está em uma posição de memória e possui um identificador único que pode ser obtido com a função **id()**

```
>>> a = 94
>>> id(a)
4297373664
>>> id(94)
4297373664
>>>
```

- A variável **a** está associada com o objeto **int** de valor 94, que possui o identificador 4297373664.

Objetos mutáveis e imutáveis

- Como um **int** é imutável, quando fazemos o incremento da variável **a**, o que ocorre na verdade é a criação de um novo objeto do tipo **int** que será associado com **a**.

```
>>> a = 94
>>> id(a)
4297373664
>>> id(94)
4297373664
>>> a = a + 1
>>> id(a)
4297373696
>>> id(95)
4297373696
>>>
```

Objetos mutáveis e imutáveis

- Objetos do tipo **list** são mutáveis (dicionários, que veremos no futuro, também são mutáveis). Isto significa que objetos deste tipo podem ter seus valores alterados.

```
>>> a=[]
>>> id(a)
4328743752
>>> a.append(1)
>>> a
[1]
>>> id(a)
4328743752
>>> a = a + [2]
>>> a
[1, 2]
>>> id(a)
4328743752
```

- No exemplo acima a lista cujo **id** é 4328743752, é alterada inicialmente de **[]** para **[1,2]**.

Objetos mutáveis e imutáveis

```
>>> a=[]
>>> id(a)
4360514376
>>> a.append(9)
>>> a
[9]
>>> id(a)    ## nao mudou
4360514376
>>> a=a+[5]  ### concatenação normal
>>> id(a)
4360808072   ### mudou
>>> a
[9, 5]
>>> a+=[88]  ### outra notação somar ao final
>>> id(a)
4360808072  ### nao mudou
>>>
```

Objetos mutáveis e imutáveis

- Objetos mutáveis e imutáveis possuem comportamento distintos quando usados em funções como veremos adiante.
- se objetos mutáveis são passados como parametros, algumas modificações (as que mantem o **id**) neles dentro da função são "sentidas" pelo objeto externo

```
def soma2(b):
    b.append(2)
    return b

def soma2x(b):
    b= b+[2]
    return b

a = [ 9,8,7]
soma2(a)
a
soma2x(a)
a
```

Strings - o que a gente já sabe

- Uma sequencia de caracteres entre "ou ' : **a ='quetrYY**
- o string vazio " ou **""**
- acessar um elemento do string **a[0]** (primeiro caracter), **a[-1]** ultimo, **a[7]** da erro
- string são como listas que não podem ser alterados. **a[1]="z"** da erro

Strings

- concatenação **str1+str2**
- o string vazio ” ou “”
- acessar um elemento do string **a[0]** (primeiro caracter), **a[-1]** ultimo, **a[7]** da erro
- string são como listas que não podem ser alterados. **a[1]=”z”** da erro

in

- o teste **sub in str** para string verifica se o **sub** é um substring (um trecho) do **str**

```
    >>> a= 'qwerty'
>>> 'er' in a
True
>>> 'ery' in a
False
```

Outros testes

- o == funciona em strings
- o > funciona em strings como em listas - ordem lexicográfica ou ordem do dicionário

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

- 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'
```

Mais informação sobre strings

- a pagina da documentação do Python sobre strings
  `https://docs.python.org/3.7/library/string.html`

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)
```

Processamento de Texto

Achar palavras em um texto.

- Fazer um programa que acha todas as posições de ocorrência de uma palavra (substring) em um texto (uma string).

```
Texto='a tete tetete'
Palavra='tete'

A resposta é [2, 7, 9]
```

Processamento de Texto

Ideia do algoritmo:

- Achamos a posição de ocorrência de **subst** em **st** e armazenamos esta na lista **pos**.
- Depois removemos toda parte inicial de **st** até o primeiro caractere onde encontramos **subst**:
  Exe: subst="abc" e st="dfg abcabc", vamos remover "dfg a" ficando st="bcabc".
- Como removemos uma parte inicial de **st** precisamos guardar o seu tamanho em **tamRemovido** pois próximas ocorrências estão deslocadas por este valor na string original.

```
>>> subst="abc"
>>> st="dfg abcabc"
>>> st.find(subst)
4                              #posição da 1o ocorrência
>>> st = st[4+1:]
>>> st
'bcabc'                       #removeu até a 1o ocorrência
>>> tamRemovido=4+1  #tamanho da parte removida
>>> st.find(subst)   #posição da 2o ocorrência
2
>>> pos = [4, tamRemovido+2] #posição da 2o ocorrência considerando o que foi removido
>>> pos
[4, 7]
>>>
```

Processamento de Texto

- Lembre-se que o **slicing st[pos:]** devolve uma string contendo todos os caracteres a partir da posição **pos**.
- Usamos isso para remover o início do texto até a primeira ocorrência da palavra.

```
st = input("Entre com um texto:")
subst = input("Entre com uma palavra:")
pos = []
tamRemovido = 0
while subst in st: #Enquanto houver uma ocorrência da palavra em st
  aux = st.find(subst) #acha posicao da 1o ocorrência
  pos.append(aux+tamRemovido) #inclui posição da palavra corrigido
  st = st[(aux+1):] #remove tudo até 1a letra da 1o ocorrência de subst
  tamRemovido = tamRemovido + aux + 1

print(pos)
```

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 virgula, por exemplo “3,4 5 6, 9” . Imprima a média deles.