UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE COMPUTAÇÃO

CIÊNCIA DA COMPUTAÇÃO

CK0211 - Fundamentos de Programação: Strings (cadeias de caracteres) e Listas

Emanuele Santos

Bibliografia: Ascencio, Cap. 6 e Cap. 9

Objetivos

- Aprender a trabalhar com strings e outras formas de manipular listas e entender melhor como elas funcionam

STRINGS
(CADEIAS DE CARACTERES) EM PYTHON

Variáveis do tipo string (str)

- Armazenam cadeias de caracteres como literais, nomes e textos em geral

```
nome = input('Qual o seu nome?')
```

- Podemos imaginar uma string como um vetor de caracteres, onde cada caractere ocupa uma posição

nome

E	m	a	n	u	e	l	e

Variáveis do tipo string (str)

- Uma string em Python tem um tamanho associado, assim como um conteúdo que também pode ser acessado caractere a caractere
- O tamanho de uma string pode ser obtido utilizando-se a função **len**
  - Essa função retorna o número de caracteres contidos na string

```
>>> nome = input('Qual o seu nome? ')
Qual o seu nome? Emanuele
>>> print(len(nome))
8
>>> letra = "A"
>>> print(len(letra))
1
```

Variáveis do tipo string (str)

- Podemos acessar o conteúdo da string, caractere a caractere
- Os caracteres podem ser acessados utilizando um número inteiro que guarda a sua posição (índice)

nome

0	1	2	3	4	5	6	7	← índice
E	m	a	n	u	e	l	e	← conteúdo

- Os índices de uma string sempre iniciam do **zero** e vão até **len(string)-1**

Variáveis do tipo string (str)

- Strings em Python são imutáveis: Uma vez criadas elas não podem ser alteradas

```
>>> nome = "Emanuele"
>>> print(nome[0])
E
>>> nome[0] = "e"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Variáveis do tipo string (str)

- Assim como em vetores, para acessar os caracteres de uma string devemos informar o índice do caractere entre colchetes [ ]

nome	0	1	2	3	4	5	6	7	
	E	m	a	n	u	e	l	e	

← índice
← conteúdo

```
>>> nome = “Emanuele”
>>> print(nome[0])
E
>>> print(nome[2])
a
>>> print(nome[8])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Operações com strings

- As variáveis do tipo string suportam as seguintes operações:
  - **Concatenação**: Juntar duas ou mais strings em uma nova string maior
  - **Composição**: Utilizar strings como modelos (ou máscaras) onde podemos inserir outras strings
  - **Fatiamento**: Utilizar apenas uma parte de uma string, ou uma fatia

Concatenação de strings

- O conteúdo de variáveis strings podem ser unidos ou concatenados.
- Para concatenar duas strings usamos o operador +
  - “AB” + “C” = “ABC”
- Um caso especial de concatenação é a repetição de uma string várias vezes
  - “A” * 3 = “AAA”

```
>>> s = “ABC”
>>> print(s + “C”)
ABCC
>>> print(s + “D” * 3)
ABCDDD
```

Composição de strings

- Nem sempre é prático juntar várias strings para construir uma mensagem

```
>>> X = 3
>>> Y = 7
>>> print(X, “+”, Y, “=”, X+Y, “.”)
3 + 7 = 10.
```

- Utilizamos a composição com marcadores de posição

```
>>> print(“%d + %d = %d.” % (X, Y, X+Y))
3 + 7 = 10.
```

Composição de strings: Tabela de marcadores

Marcador	Tipo
%d	Números inteiros
%s	Strings
%f	Números decimais

Composição com marcadores: números inteiros

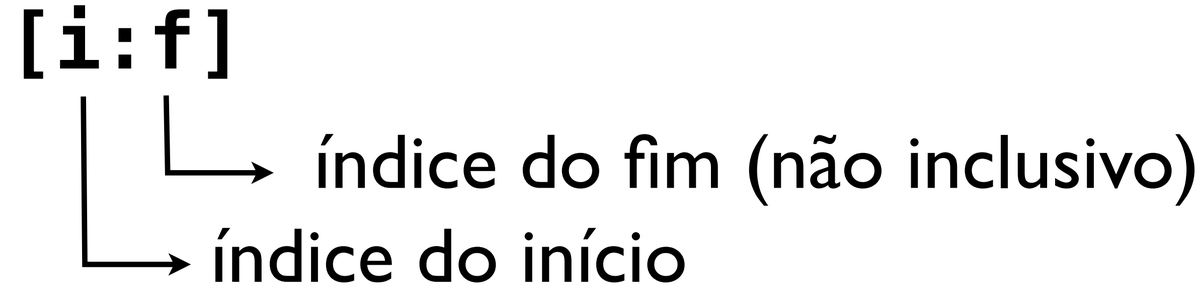
```
>>> idade = 25
>>> print(“%d” % idade)
25
>>> print(“%03d” % idade)
025
>>> print(“[%3d]” % idade)
[ 25]
>>> print(“[%-3d]” % idade)
[25 ]
```

Composição com marcadores: números reais

```
>>> print(“%f” % 5)
5.000000
#número decimal usando 2 casas decimais
>>> print(“%.2f” % 5)
5.00
#número decimal usando 5 posições e 2 casas decimais
>>> print(“%5.2f” % 5)
 5.00
#número decimal usando 10 posições e 2 casas decimais
>>> print(“%10.2f” % 5)
      5.00
```

Fatiamento de strings

- Podemos fatiar strings de modo a utilizar apenas parte de uma string
- O fatiamento funciona utilizando dois pontos : no índice de uma string para indicar um intervalo




```
>>> frase = “Eu estou aprendendo python”
# para pegar os dois primeiros caracteres
>>> print(frase[0:2])
Eu
```

Fatiamento de strings

```
>>> s = “ABCDEFGHI”
# podemos omitir o da esquerda para indicar
# o início da string
>>> print(s[:2])
AB
# assim como o da direita para indicar o final da string
>>> print(s[1:])
BCDEFGHI
# usando somente : indica a string completa
>>> print(s[:])
ABCDEFGHI
# índices negativos indicam posições a partir do fim
>>> print(s[-1:])
I
```

O que será mostrado?

```
s = “0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.- ”
i = 14
mensagem = s[i] + s[-9] + s[-1] + s[i] + s[-11:-9] + s[24] +
s[-9] + s[-1] + s[10] + s[25] + s[-12] + s[i] + s[23] +
s[13:15] + s[23] + s[13] + s[24] + s[-1] + s[25] + s[-5] +
s[-10] + s[17] + s[24] + s[23] + s[-3]
print(mensagem)
```

TRABALHANDO COM STRINGS

Verificação parcial de strings

- Quando você precisar verificar se uma storing começa ou termina com alguns caracteres, você pode usar os métodos **startswith** e **endswith**

```
>>> nome = "João da Silva"
>>> nome.startswith("João")
True
>>> nome.startswith("joão")
False
>>> nome.endswith("Silva")
True
```

- **startswith** e **endswith** consideram letras maiúsculas e minúsculas como letras diferentes

Verificação parcial de strings

- E se quisermos ignorar maiúsculas e minúsculas? Podemos converter a string para minúsculas ou maiúsculas antes de realizar a comparação
- **lower** retorna uma cópia da string com todos os caracteres minúsculos e **upper** retorna uma cópia da string com todos os caracteres maiúsculos

```
>>> s = "O Rato roeu a roupa do Rei de Roma"
>>> s_min = s.lower()
>>> print(s_min)
'o rato roeu a roupa do rei de roma'
>>> s_mai = s.upper()
>>> print(s_mai)
'O RATO ROEU A ROUPA DO REI DE ROMA'
>>> s_min.startswith(“o rato")
True
>>> s_mai.startswith(“O RATO")
True
```

Verificação parcial de strings

- Outra maneira de verificar se uma palavra pertence a uma storing é utilizando o operador **in**

```
>>> s = "Pedro Álvares Cabral"
>>> "Pedro" in s
True
>>> "Álvares" in s
True
>>> "Cabral" in s
True
>>> "o Á" in s
True
>>> "pedro" in s
False
```

Verificação parcial de strings

- Você também pode testar se uma string não está contida na outra usando **not in**

```
>>> s = "Todos os caminhos levam a Roma"
>>> "levam" not in s
False
>>> "Caminhos" not in s
True
>>> "AS" not in s
True
```

Verificação parcial de strings

- Você pode combinar **in** e **not in** com lower e upper para ignorar maiúsculas e minúsculas na comparação

```
>>> s = "Pedro leu um livro"
>>> "pedro" in s.lower()
True
>>> "LIVRO" in s.upper()
True
>>> "leu" not in s.lower()
False
>>> "revista" not in s.lower()
True
```

Contagem

- Para contar ocorrências de uma palavra ou de uma letra em uma storing, utilize o método **count**

```
>>> t = "um tigre, dois tigres, três tigres"
>>> t.count("tigre")
3
>>> t.count("tigres")
2
>>> t.count("t")
4
>>> t.count("z")
0
```

Pesquisa de strings

- Para pesquisar se uma string está dentro de outra e obter a posição da primeira ocorrência, você pode utilizar o método **find**

```
>>> s = "Python é massa"
>>> s.find("mas")
9
>>> s.find("ok")
-1
```

- Para pesquisar da direita para a esquerda, use **rfind**

```
>>> s = "Um dia de sol"
>>> s.rfind("d")
7
>>> s.find("d")
3
```

Pesquisa de strings

- Tando **find** como **rfind** suportam duas opções adicionais: início (start) e fim (end)
- Se você especificar início, a pesquisa começará a partir dessa posição
- Analogamente, se você especificar fim, a pesquisa terminará nessa posição

```
>>> t = "um tigre, dois tigres, três tigres"
>>> t.find("tigres")
15
>>> t.rfind("tigres")
28
>>> t.find("tigres", 7) # início=7
15
>>> t.find("tigres", 30) # início=30
-1
>>> t.find("tigres", 0, 10) # início=0 fim=10
-1
```

Pesquisa de strings

- **find** e **rfind** retornam apenas a primeira ocorrência. Como fazer para retornar todas as ocorrências?

```
t = "um tigre, dois tigres, três tigres"
i = 0
while i > -1:
    i = t.find("tigre", i)
    if i >= 0:
        print("Posição: %d” % i)
        i += 1
```

Exercício 1

- Escreva um programa que leia duas strings. Verifique se a segunda string ocorre dentro da primeira e mostre a posição de início.
- Exemplo:

```
Digite a primeira string: ASDFGHJKL
Digite a segunda string: HJ
Resultado: HJ encontrado na posição 5 de ASDFGHJKL
```

Exercício 2

- Escreva um programa que leia uma string e mostre quantas vezes cada caractere aparece nessa string. Não diferencie maiúsculas e minúsculas
- Exemplo:

```
Digite uma string: TtAaCCc
T: 2x
A: 2x
C: 3x
```

COMO FUNCIONAM AS LISTAS EM PYTHON

Listas em Python

- Lista de três inteiros

```
>>> [1, 7, -2]
```

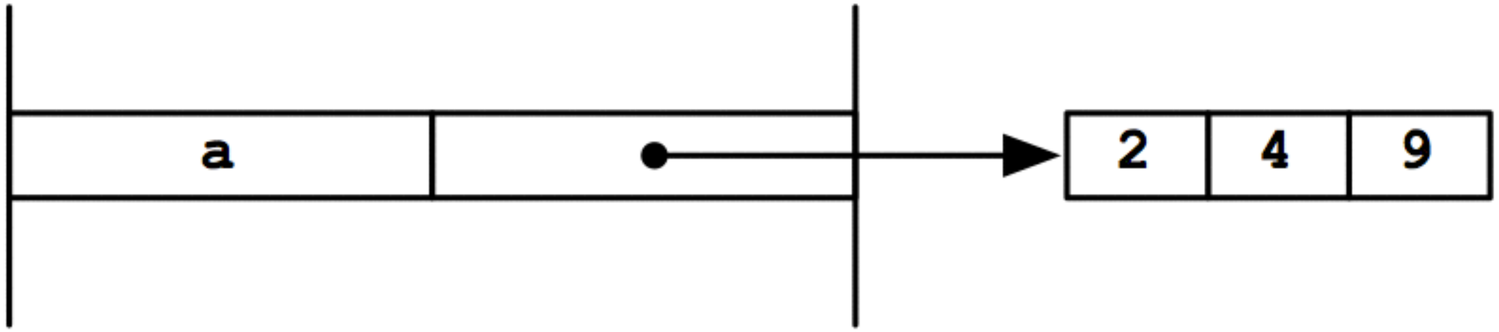
- Cujo diagrama abstrato da representação na memória, é:

Listas em Python

- Podemos atribuir uma lista a uma variável

```
>>> a = [2, 4, 9]
```

- Quando atribuímos uma lista a uma variável, fazemos a ligação do nome da variável ao endereço de memória da estrutura:

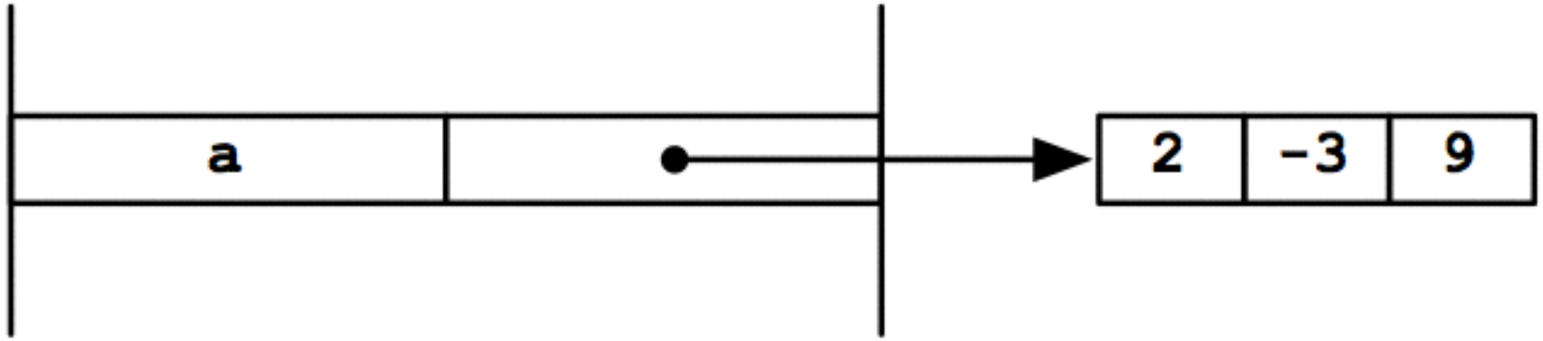


Modificações de listas

- Listas são estruturas de dados mutáveis
  - Podemos alterar os valores armazenados como seus elementos

```
a[1] = -3
```

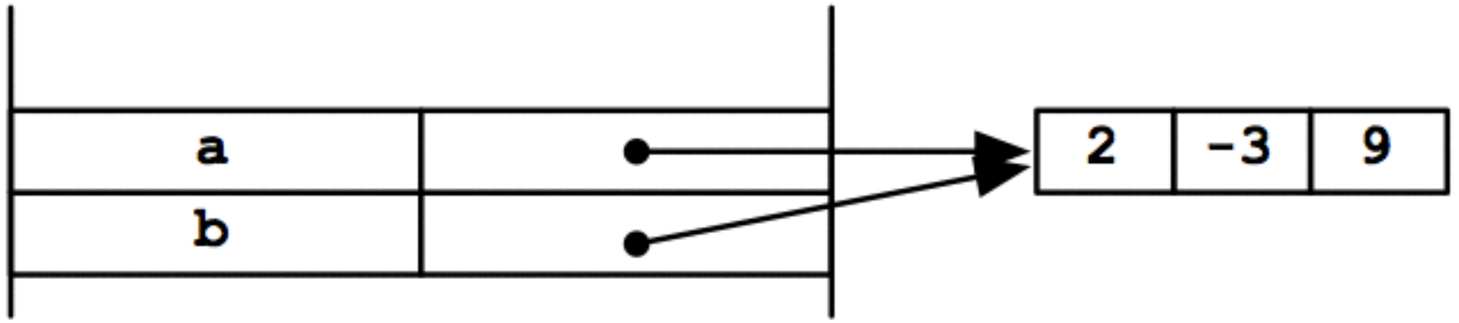
- O comando acima atribui -3 como o segundo elemento de a



Modificações de listas

- A mutabilidade de listas gera muitas consequências com as quais precisamos ter cuidado
- É sempre bom utilizar os diagramas para ajudar a entender o que realmente está acontecendo
- Continuando o exemplo anterior

```
>>> b = a
```



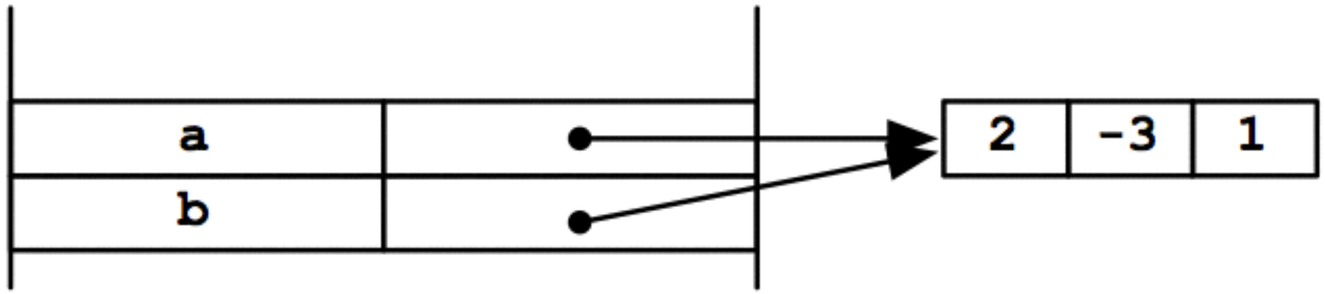
Modificações de listas

- Agora podemos referenciar partes da lista através da variável **b**

```
>>> b[0]
2
>>> b[2] = 1
```

- Note que como **a** e **b** apontam para a mesma lista, ao mudar **b** também mudamos **a**!

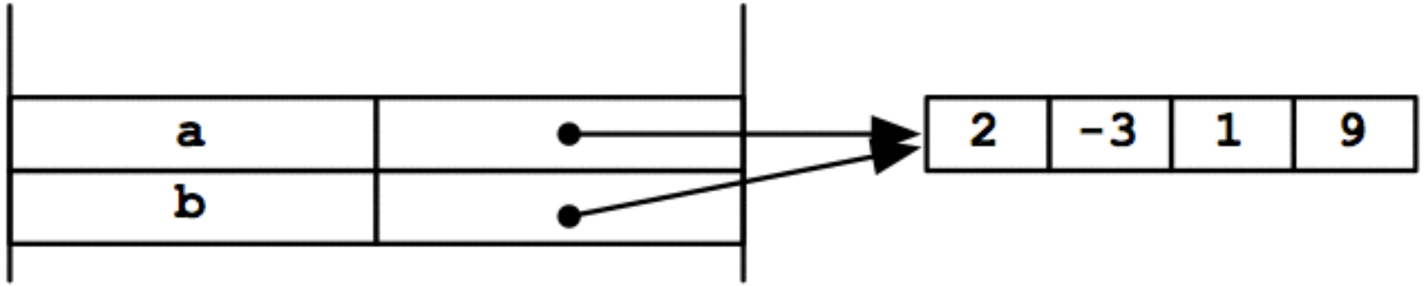
```
>>> a
[2, -3, 1]
```



Modificações de listas

- Conceitualmente, um vetor após criado, não pode ter seu tamanho alterado
- No entanto, em Python, podemos aumentar ou diminuir o tamanho de listas, adicionando e removendo elementos

```
# adicionando elementos ao final da lista
>>> a.append(9)
```




```
>>> b
[2, -3, 1, 9]
```

Modificações de listas

- Outros métodos de listas

```
# inserindo elemento x na posição i
# list.insert(i,x)
>>> a.insert(1,5)
# remover o primeiro elemento x que encontrar na lista
# list.remove(x)
>>> a.remove(5)
# list.pop() remove o último elemento da lista e o retorna
>>> x = a.pop()
# list.pop(i) remove o elemento da posição i e o retorna
>>> x = a.pop(0)
```

http://docs.python.org/3/tutorial/datastructures.html

Fatiamento de listas

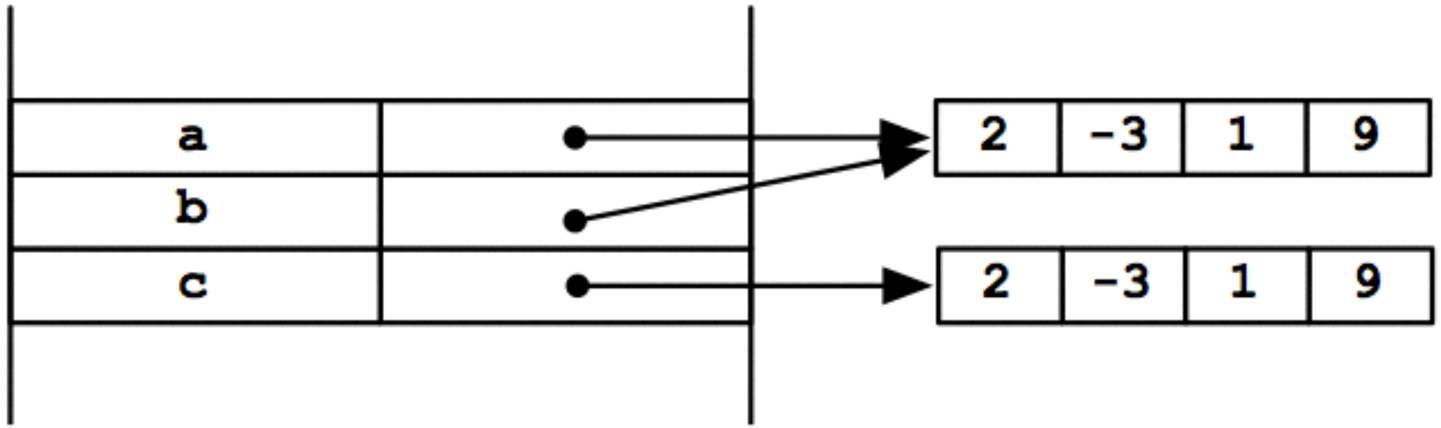
- Podemos também fatiar uma lista, da mesma forma que fizemos com strings:

```
>>> L = [5,4,3,2,1]
>>> L[0:2]
[5,4]
>>> L[:2]
[5,4]
>>> [1:]
[4,3,2,1]
>>> L[0:5]
[5,4,3,2,1]
>>> L[:]
[5,4,3,2,1]
>>> L[-2:]
[2,1]
```

Copiando listas

- Duas maneiras diferentes de copiar os elementos de uma lista:

```
>>> c = list(a)
# ou
>>> c = a[:]
```



Copiando listas

- Para adicionar um elemento à lista e ao mesmo tempo obter uma cópia da lista

```
>>> a + [1]
```

- O operador + cria novas listas que contem os elementos de ambos operandos

Exercícios

1. Mostre o diagrama da memória após os comandos abaixo

```
>>> a = [5, 6]
>>> b = [1, 2]
>>> c = b + a
```

2. Dada a lista L abaixo, e as variáveis x = 1 e y = 3 escreva o resultado dos comandos:

L = [2, 9, 4, 8, 3, 7, 11, 10, 1, 13]

a) print(L[x+1]) b) print(L[x+2]) c) print(L[x+y])

d) print(L[L[x+y]]) e) print(L[L[0] + L[4]]) f) print(L[L[L[8]]])

Exercício

- Escreva um algoritmo que calcula o produto escalar **s** de dois vetores e teste o seu programa com os vetores v = [4, 6, -1] e w = [5, -4, -4]

$$s = \sum_{i=1}^{n} v_i w_i$$