# Processamento da Informação
# – Teoria –

## Strings

Semana 04

Prof. Jesús P. Mena-Chalco

15/05/2013

# Das aulas anteriores...

Vimos vários tipos de dados: int, float, long, string

Stings são qualitativamente diferentes dos outros tipos pois são compostas de pedaços menores (caracteres): **Tipo de dado composto**.

```
>>> x = “mensagem”
>>> print x
mensagem
```

# Uma string é uma sequência

Uma string (cadeia) é uma sequência de caracteres.

Podemos acessar aos caracteres com o operador colchete

```
>>> fruta = "banana"
>>> letra = fruta[1]
```

# Uma string é uma sequência

Uma string (cadeia) é uma sequência de caracteres.

Podemos acessar aos caracteres com o operador colchete

```
>>> fruta = "banana"
>>> letra = fruta[1]
>>> print letra
```
**a**

# Uma string é uma sequência

A primeira letra (“b”) tem a posição 0. A segunda letra (“a”) tem a posição 1, ...

```
>>> fruta = "banana"

>>> print fruta[0]
 b
>>> print fruta[1]
 a
>>> print fruta[2]
 n
```

Índices

# Uma string é uma sequência

A primeira letra (“b”) tem a posição 0. A segunda letra (“a”) tem a posição 1, ...

```
>>> fruta = "banana"

>>> print fruta[0]
 b
>>> print fruta[1]
 a
>>> print fruta[2]
 n
>>> print fruta[1.5]
```

# Uma string é uma sequência

A primeira letra (“b”) tem a posição 0. A segunda letra (“a”) tem a posição 1, ...

```
>>> fruta = "banana"

>>> print fruta[0]
 b
>>> print fruta[1]
 a
>>> print fruta[2]
 n
>>> print fruta[1.5]
TypeError: string indices must be integers
```

No **índice**:
Podemos usar qualquer expressão, incluindo variáveis e operadores,
Entretanto, o valor do índice deve ser inteiro.

# Comprimento

A função **len** retorna o número de caracteres de uma string.

```
>>> fruta = "banana"
>>> len(fruta)
6
```

# Comprimento

A função **len** retorna o número de caracteres de uma string.

```
>>> fruta = "banana"
>>> len(fruta)
6
```

Para pegar a última letra de uma string, podemos tentar realizar a seguinte operação:

```
>>> comprimento = len(fruta)
>>> ultima = fruta[comprimento]
IndexError: string index out of range
```

Índice fora do intervalo

# Comprimento

A função **len** retorna o número de caracteres de uma string.

```
>>> fruta = "banana"
>>> len(fruta)
6
```

Para pegar a última letra de uma string, podemos tentar realizar a seguinte operação:

```
>>> comprimento = len(fruta)
>>> ultima = fruta[comprimento-1]
```

# Percorrendo uma string com um laço

Várias operações envolvem processamento de strings, considerando um caractere de cada vez.

Exemplo para percorrer e imprimir cada letra da variável fruta:

```
indice = 0
while indice < len(fruta):
    letra = fruta[indice]
    print letra
    indice = indice + 1
```

# Percorrendo uma string com um laço

Várias operações envolvem processamento de strings, considerando um caractere de cada vez.

Exemplo para percorrer e imprimir cada letra da variável fruta:

```
indice = 0
while indice < len(fruta):
    letra = fruta[indice]
    print letra
    indice = indice + 1
```

Se fruta = “banana”

Então o resultado será:

b
a
n
a
n
a

# Percorrendo uma string com um laço

Várias operações envolvem processamento de strings, considerando um caractere de cada vez.

Exemplo para percorrer e imprimir cada letra da variável fruta:

```
indice = len(fruta)-1
while indice >= 0 :
    letra = fruta[indice]
    print letra
    indice = indice - 1
```

Imprime as letras de trás para frente

Se fruta = “banana”

Então o resultado será:

a
n
a
n
a
b

# Percorrendo uma string com um laço

Também podemos usar o laço **for** para percorrer uma string.

```
for letra in fruta:
    print letra
```

A cada vez através do laço, o próximo caractere da string é atribuído à variável **letra**. O laço continua até que não reste mas caracteres.

# Percorrendo uma string com um laço

Exemplo de concatenação de strings para a geração de uma série abecedário:

```
prefixos = "JKLMNOPQ"
sufixo = "ack"

for letra in prefixos:
    print letra + sufixo
```

# Percorrendo uma string com um laço

Exemplo de concatenação de strings para a geração de uma série “abecedário”:

```
prefixos = "JKLMNOPQ"
sufixo = "ack"

for letra in prefixos:
    print letra + sufixo
```

```
Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack
```

# Fatias de strings

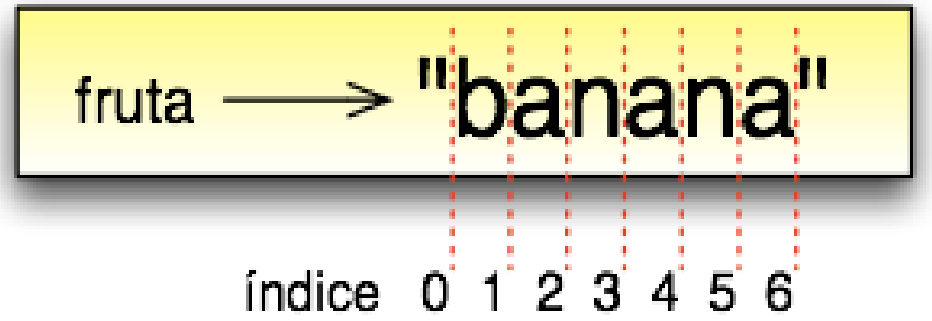
Um segmento de uma string é chamado de uma fatia (subsequência).

Selecionar uma fatia é similar a selecionar um caractere.

```
>>> s = "Pedro, Paulo e Maria"
>>> print s[0:5]
Pedro
>>> print s[7:12]
Paulo
>>> print s[16:21]
Maria
```

# Fatias de strings

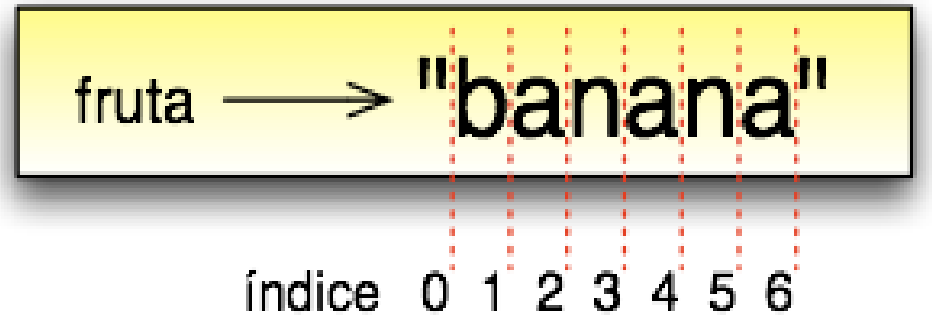
O operador **[m:n]** retorna a parte da string do “m-ésimo” caractere ao “n-ésimo” caractere, incluindo o primeiro mas excluindo o último. (esse comportamento não é intuitivo)




Fruta[0:3] → “ban”
Fruta[3:3] → “”

# Fatias de strings

O operador **[m:n]** retorna a parte da string do “m-ésimo” caractere ao “n-ésimo” caractere, incluindo o primeiro mas excluindo o último. (esse comportamento não é intuitivo)




fruta[0:3] → “ban”
fruta[3:3] → “”
fruta[6:3] → “”

Se o primeiro índice é maior ou igual ao segundo índice, o resultado será uma string vazia

# Uma função `find`

O que faz a seguinte função?

```
def find(cadeia, letra):
    indice = 0
    while indice < len(cadeia):
        if cadeia[indice] == letra:
            return indice
        indice = indice + 1
    return -1
```

# Uma função `find`

O que faz a seguinte função?

```
def find(cadeia, caractere):
    indice = 0
    while indice < len(cadeia):
        if cadeia[indice] == caractere:
            return indice
        indice = indice + 1
    return -1
```

A função procura o índice
do **caractere** na **cadeia**.
Se o caractere não é encontrado,
a função retorna -1.

# Iterando e contando

O seguinte programa conta o número de vezes que a letra **'a'** aparece na string fruta:

```
fruta = "banana"
contador = 0
for letra in fruta:
    if letra == 'a'
        contador = contador + 1
print contador
```

# Iterando e contando

Generalizando o procedimento para contar o número de vezes que aparece uma **letra** em uma **cadeia**.

```
def contar_vezes(cadeia, letra)
  contador = 0
  for l in cadeia:
      if l == letra
          contador = contador + 1
  return contador
```

# O operador `in`

A palavra in é um operador booleano que considera duas strings e retorna True se a primeira aparece como uma substing na segunda:

```
>>> 'a' in 'banana'
True

>>> 'caqui' in 'banana'
False

>>> 'ana' in 'banana'
True
```

# O operador `in`

A seguinte função imprime todas as letras da **palavra1** que aparece também na **palavra2**:

```
def procura_letras(palavra1, palavra2):
    for letra in palavra1:
        if letra in palavra2:
            print letra
```

# O operador `in`

A seguinte função imprime todas as letras da **palavra1** que aparece também na **palavra2**:

```
def procura_letras(palavra1, palavra2):
    for letra in palavra1:
        if letra in palavra2:
            print letra
```

```
>>> procura_letras('apples', 'oranges')
a
e
s
```

# Atividade em aula

**Questao 1:** Crie uma função que receba duas palavras e retorne True se uma das palavras é o reverso da outra:

```
def reverso(palavra1, palavra2):
```

Exemplo:

'pots' é reverso de 'stop'

'livres' é reverso de 'servil'

# Atividade em aula

```
1  def reverso(palavra1, palavra2):
2      if len(palavra1) != len(palavra2):
3          return False
4      i = 0
5      j = len(palavra2)-1
6      while j>=0:
7          if palavra1[i] != palavra2[j]:
8              return False
9          i = i+1
10         j = j-1
11     return True
```

# Atividade em aula

```
1  def reverso(palavra1, palavra2):
2      if len(palavra1) != len(palavra2):
3          return False
4      i = 0
5      j = len(palavra2)-1
6      while j>=0:
7          if palavra1[i] != palavra2[j]:
8              return False
9          i = i+1
10         j = j-1
11     return True
```

# Atividade em aula

```
>>> reverso('pots', 'stop')
1
>>> reverso('livres','servil')
1
>>> reverso('livres','brinquedo')
0
```

# Atividade em aula

**Questão 2:** Crie uma função que receba duas palavras e retorne True caso a primeira palavra seja um prefixo da segunda:

Cabeçalho: def prefixo (palavra1, palavra2):

Exemplo: 'uf' é prefixo de 'ufabc'

# Atividade em aula

Solução com erro...

```
1  def prefixo(palavra1, palavra2):
2      if len(palavra1) > len(palavra2):
3          return False
4      if palavra1 in palavra2:
5          return True
6      else:
7          return False
```

```
>>> prefixo("uf","ufabc")
1
>>> prefixo("uf","abcuf")
1
```

# Atividade em aula

```
1  def prefixo(palavra1, palavra2):
2      if len(palavra1) > len(palavra2):
3          return False
4      i=0
5      while i<len(palavra1):
6          if palavra1[i]!=palavra2[i]:
7              return False
8          i=i+1
9      return True
```

```
>>> prefixo("uf","ufabc")
1
>>> prefixo("uf","abcuf")
0
>>> prefixo("ufabc","uf")
0
```

# Atividade em aula

**Questão 3:** Crie uma função que receba duas palavras e retorne True caso a primeira palavra seja um sufixo da segunda:

Cabeçalho: def sufixo (palavra1, palavra2):

Exemplo: 'abc' é sufixo de 'ufabc'

# Atividade em aula

```
def sufixo(palavra1, palavra2):
    if len(palavra1) > len(palavra2):
        return False
    j=len(palavra1)-1
    while j>=0:
        if palavra1[j]!=palavra2[len(palavra2)-len(palavra1)+j]:
            return False
        j=j-1
    return True
```

```
>>> sufixo("uf","abcuf")
1
>>> sufixo("cuf","abcuf")
1
>>> sufixo("auf","abcuf")
0
```