Strings

Prof. Alberto Costa Neto

Programação em Python

Tipo de Dados String

- Uma string é uma seqüência de caracteres
- Uma literal string usa apóstrofos ou aspas para delimitar os caracteres
- Para strings, + significa “concatenar”
- Mesmo que uma string contenha apenas números, ela continua sendo uma string
- Podemos converter uma string contendo caracteres numéricos em valor numérico usando int() ou float()

```
>>> str1 = "Bom"
>>> str2 = 'dia'
>>> bom = str1 + str2
>>> print bom
Bomdia
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call last):  File "<stdin>", line 1, in <module>TypeError: cannot concatenate 'str' and 'int' objects
>>> x = int(str3) + 1
>>> print x
124
>>>
```

Lendo e Convertendo

- Preferimos ler dados usando strings e então analisar e converter os dados conforme seja necessário
- Isto nos dá mais controle sobre situações de erro e/ou entrada do usuário inválida
- Números de entrada crus(raw) devem ser convertidos a partir das strings

```
>>> nome = raw_input('Nome:')
Nome:Alberto
>>> print nome
Alberto
>>> apple = raw_input('Enter:')
Enter:100
>>> x = apple - 10
Traceback (most recent call last):  File "<stdin>", line 1, in <module>TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print x
90
```

Detalhes Internos de Strings

- Podemos acessar individualmente qualquer caractere em uma String usando o índice entre colchetes
- O valor do índice deve ser um inteiro e inicia-se em 0 (zero)
- O valor do índice pode ser obtido de uma expressão numérica

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruta = 'banana'
>>> letra = fruta[1]
>>> print letra
a
>>> n = 3
>>> w = fruta[n - 1]
>>> print w
n
```

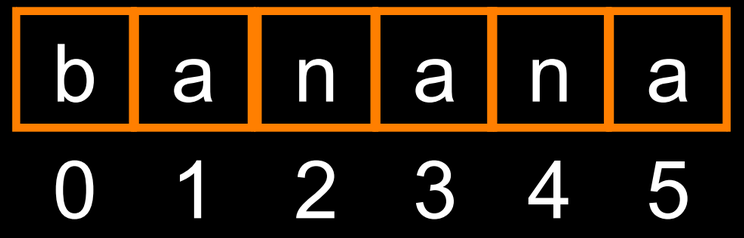
Um Caractere muito distante

- Caso tente acessar um índice além do final da String, você irá obter erro.
- Então tenha cuidado ao calcular os índices e construir fatias (slices)

```
>>> str = 'abc'
>>> print str[5]
Traceback (most recent call last):  File "<stdin>", line 1, in <module>IndexError: string index out of range
>>>
```

Strings têm comprimento (Length)

- Existe uma função built-in len que nos dá o comprimento de uma string




```
>>> fruta = 'banana'
>>> print len(fruta)
6
```

Função len

```
>>> fruta = 'banana'
>>> x = len(fruta)
>>> print x
6
```

Uma função é um bloco de código fonte armazenado que usamos.

Uma função recebe uma entrada e produz uma saída.

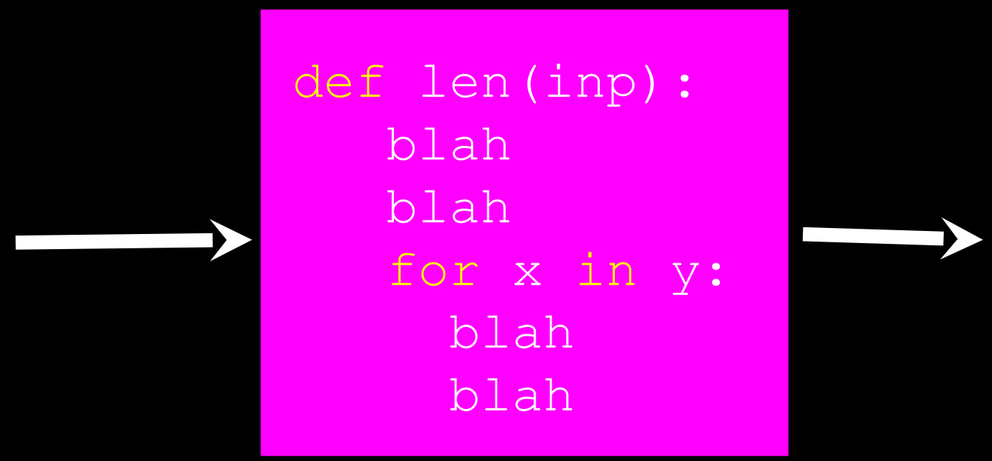
Guido escreveu este código

Função len

```
>>> fruta = 'banana'
>>> x = len(fruta)
>>> print x
6
```

Uma função é um bloco de código fonte armazenado que usamos.

Uma função recebe uma entrada e produz uma saída.



Laços e Strings

- Podemos usar o comando while (com uma variável de iteração) e a função len para criar um laço para acessar cada caractere de uma String individualmente

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

```
0 b
1 a
2 n
3 a
4 n
5 a
```

Laços e Strings

- Um laço determinado for é muito mais elegante
- A variável de iteração é totalmente controlada pelo laço for

```
fruta = 'banana'
for letra in fruta:
    print letra
```

```
b
a
n
a
n
a
```

while x for

```
fruta = 'banana'
for letra in fruta:
    print letra
```

- O laço for é muito mais elegante
- Porque a variável de iteração é totalmente controlada pelo laço for

```
fruta = 'banana'                    b
indice = 0                          a
while indice < len(fruta):          n
    letra = fruta[indice]           a
    print indice, letra             n
    indice = indice + 1             a
```

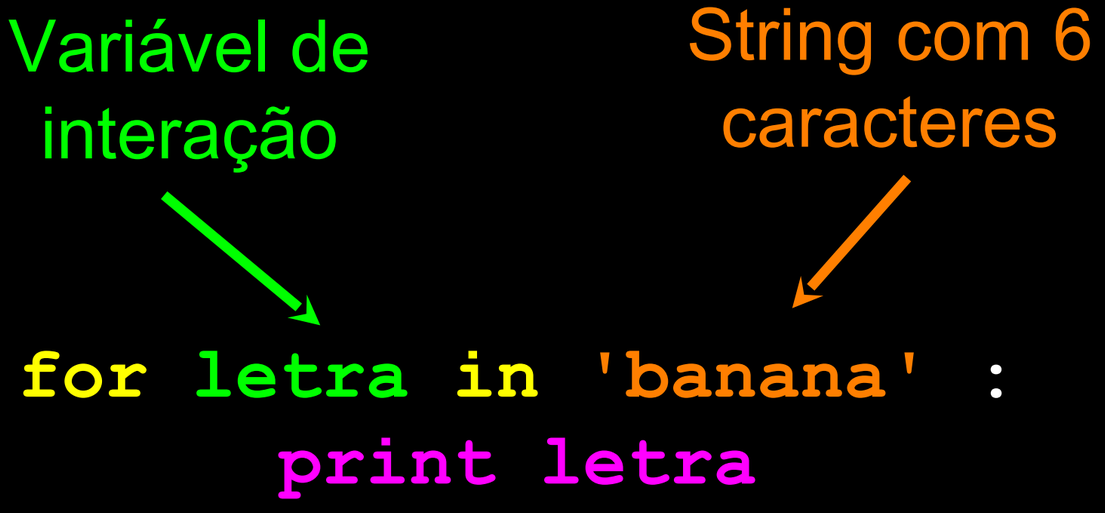
Laços e Contagem

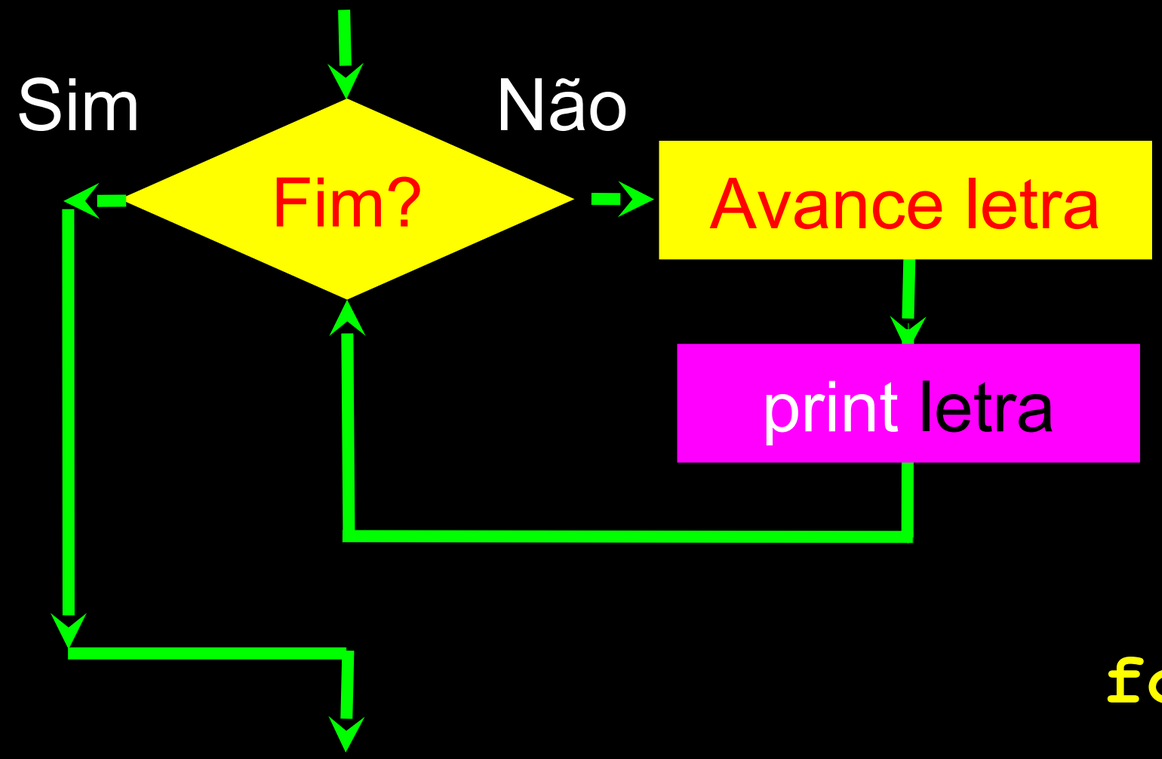
- Este é um laço que acessa cada letra da String e conta o número de vezes que o caractere 'a' é encontrado

```
palavra = 'banana'
cont = 0
for letra in palavra :
    if letra == 'a' :
        cont = cont + 1
print cont
```

Mais detalhes sobre o in

- A variável “itera” através da seqüência (conjunto ordenado)
- O bloco (corpo) de código é executado uma vez para cada valor na (in) seqüência
- A variável de iteração move-se por todos os valores na (in) seqüência



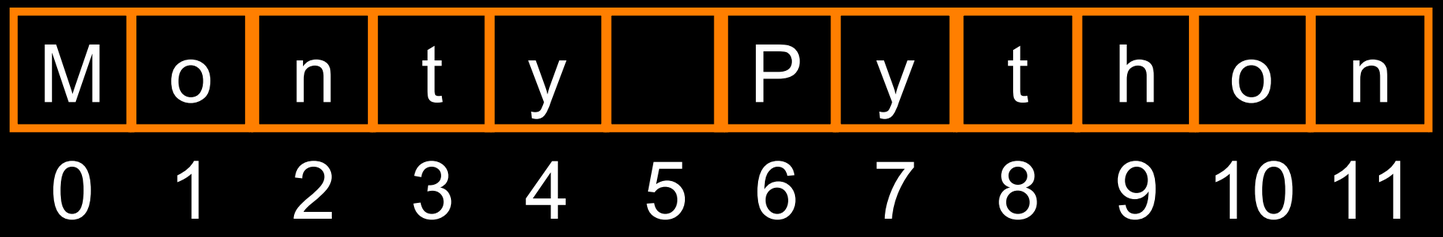


b	a	n	a	n	a

```
for letra in 'banana' :
    print letra
```

A variável de iteração “itera” pela string e o bloco (corpo) do código é executado uma vez para cada valor na (in) seqüência

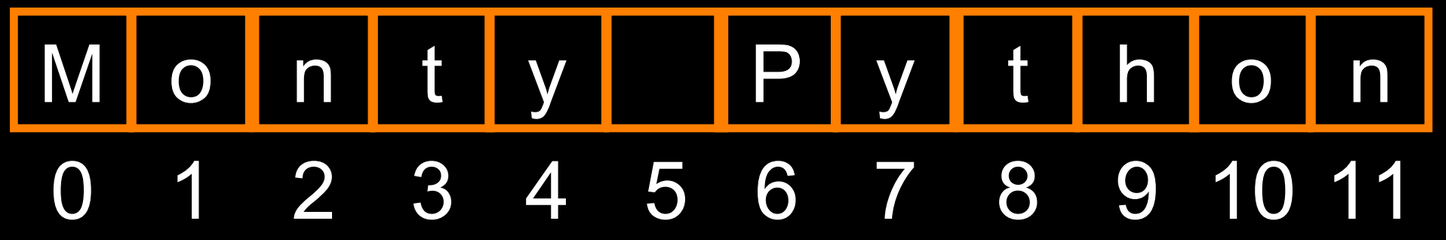
Slicing (particionando) Strings



- Podemos obter pedaços (slices ou substrings) de uma String usando o operador colon ':' (dois pontos)
- O segundo número é o índice do final da substring. ATENÇÃO: Observe que ele não fará parte da substring!
- Se ele for maior que o final da string, a substring termina no último caractere

```
>>> s = 'Monty Python'
>>> print s[0:4]
Mont
>>> print s[6:7]
P
>>> print s[6:20]
Python
```

Slicing (particionando) Strings



- Se não fornecermos o primeiro ou segundo número, serão assumidos como sendo, respectivamente, o início e o fim

```
>>> s = 'Monty Python'
>>> print s[:2]
Mo
>>> print s[8:]
thon
>>> print s[:]
Monty Python
```

Concatenação de Strings

- Quando o operador + é aplicado a strings, ele significa "**concatenação**"

```
>>> a = 'Bom'
>>> b = a + 'dia'
>>> print b
Bomdia
>>> c = a + ' ' + 'dia'
>>> print c
Bom dia
>>>
```

Usando in como um operador lógico

- A palavra chave in também pode ser usada para checar se uma string está dentro (in) de outra string
- O in é uma expressão lógica (retorna True ou False) e pode ser usada em um comando if

```
>>> fruta = 'banana'
>>> 'n' in fruta
True
>>> 'm' in fruta
False
>>> 'nan' in fruta
True
>>> if 'a' in fruta :
...     print 'Encontrado!'
...
Encontrado!
>>>
```

Comparação de Strings

```
if palavra == 'banana':
    print  'Ok, bananas.'

if palavra < 'banana':
    print 'A palavra,' + palavra + ', vem antes de banana.'
elif palavra > 'banana':
    print 'A palavra,' + palavra + ', vem depois de banana.'
else:
    print 'Ok, bananas.'
```

Biblioteca (Library) de String

- Python tem várias funções que manipulam strings na biblioteca string
- Estas funções pertencem à própria string. Para chamá-las, basta adicionar um ponto '.' e a chamada à função após o nome da variável string
- Estas funções não modificam a string original. Ao invés disso, retornam uma nova string que contém o valor alterado

```
>>> cump = 'Ola Bob'
>>> cump_min = cump.lower()
>>> print cump_min
ola bob
>>> print cump
Ola Bob
>>> print 'Bom Dia!'.lower()
bom dia!
>>>
```

```
>>> str = 'Ola Mundo'
>>> type(str)
<type 'str'>
>>> dir(str)
['capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

https://docs.python.org/2/library/stdtypes.html#string-methods

`str.` **`replace`**(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.` **`rfind`**(*sub*[, *start*[, *end*]])

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within s[start,end]. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

`str.` **`rindex`**(*sub*[, *start*[, *end*]])

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

`str.` **`rjust`**(*width*[, *fillchar*])

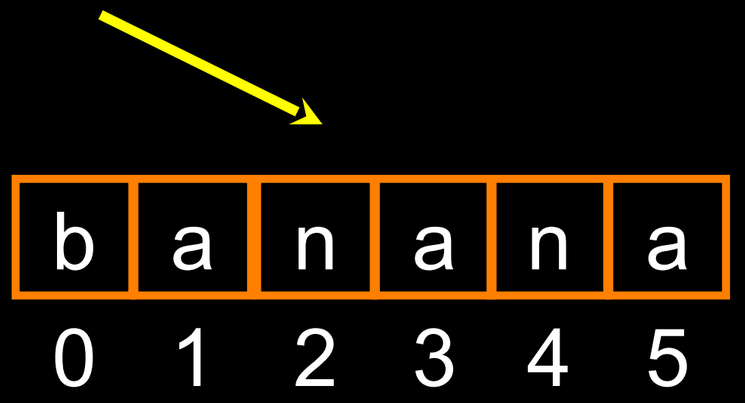
Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is a space). The original string is returned if *width* is less than `len(s)`.

Biblioteca String

str.capitalize()
str.center(width[, fillchar])
str.endswith(suffix[, start[, end]])
str.find(sub[, start[, end]])
str.lstrip([chars])
str.replace(old, new[, count])
str.lower()
str.rstrip([chars])
str.strip([chars])
str.upper()

Buscando uma String

- Usamos a função find() para buscar por uma substring dentro de outra string
- find() encontra a primeira ocorrência da substring, iniciando a busca pelo índice 0
- Se a substring não for encontrada, a função find() retorna -1
- Lembre-se que as posições em strings começam em 0




```
>>> fruta = 'banana'
>>> pos = fruta.find('na')
>>> print pos
2
>>> pos_z = fruta.find('z')
>>> print pos_z
-1
```

Transformando tudo para maiúsculas (UPPER CASE)

- Podemos criar uma cópia de uma string em lower case para upper case
- Com certa freqüência, ao fazer buscas em uma String com a função find(), precisamos converter para lower case para que a busca ocorra corretamente

```
>>> cump = 'Ola Bob'
>>> maiu = cump.upper()
>>> print maiu
OLA BOB
>>> minu = cump.lower()
>>> print minu
ola bob
>>>
```

Busca e Substituição

- A função replace() trabalha como a operação de "localizar e substituir" de um editor de texto
- Ela substitui todas as ocorrência da string de busca pela string substituta

```
>>> cump = 'Ola Bob'
>>> nstr = cump.replace('Bob','Jane')
>>> print nstr
Ola Jane
>>> nstr = cump.replace('o','X')
>>> print nstr
Ola BXb
>>>
```

Extraindo Espaços em Branco

- É comum precisarmos remover espaços do início e/ou final de uma String
- lstrip() e rstrip() removem espaços em branco à esquerda e à direita, respectivamente
- strip() remove os brancos tanto no início como no final

```
>>> cumprimento = ' Ola Bob  '
>>> cumprimento.lstrip()
'Ola Bob  '
>>> cumprimento.rstrip()
'   Ola Bob'
>>> cumprimento.strip()
'Ola Bob'
>>>
```

Prefixos

```
>>> linha = 'Por favor tenha um bom dia'
>>> line.startswith('Por favor')
True
>>> line.startswith('p')
False
```

Analisando e Extraindo

21 31

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
>>> dado = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> pos_arroba = dado.find('@')
>>> print pos_arroba
21
>>> pos_espaco = dado.find(' ',pos_arroba)
>>> print pos_espaco
31
>>> dominio = dado[pos_arroba+1 : pos_espaco]
>>> print dominio
uct.ac.za
```

Acknowledgements / Contributions
Agradecimentos / Contribuições

...

Initial Development: Charles Severance, University of Michigan School of Information

These slides were translated and adapted by Alberto Costa Neto (albertocn.sytes.net) of the Federal University of Sergipe