Processamento da Informação

Fabrício Olivetti de França

02 de Fevereiro de 2019

UFABC

CMCC
Centro de Matemática, Computação e Cognição
Center for Mathematics, Computation and Cognition

Topics

1. Strings

Strings

Manipulação de Textos

Em Python o tipo básico para representar textos é a `string`.

Uma `string` é caracterizada por uma sequência de caracteres, imprimíveis ou não, provenientes da tabela ASCII ou da UTF-8.

Manipulação de Textos

```
s1 = "Olá mundo!"
s2 = "Tabulação é \\t\tpular linha com \\n\nEntenderam?"
s3 = "A prova se aproxima!"

print(s1)
print(s2)
print(s3)
```

Visualizar

Operações com strings

A linguagem Python fornece algumas funções para manipularmos `strings`:

função	descrição
len(s)	retorna o tamanho da string
<,>,==,!=,<=,>=	compara uma string lexicograficamente
s1 + s2	concatena duas strings
s1 * n	replica a string *n* vezes

Operações com strings

```
s = "Olá mundo"
len(s) == 9
"abacate" < "bola"
"bolacha" > "bola"
s == "Olá " + "mundo"
"Olá "*2 + "mundo" == "Olá Olá mundo"
```

Indexação

Para acessar um caractere de uma `string` utilizamos a notação:

`s[pos]`

com `pos` sendo a posição que queremos acessar.

Indexação

A posição começa a contar do 0:

```
s = "ola"
s[0] == 'o'
s[1] == 'l'
s[2] == 'a'
s[3] == ERRO!
```

Indexação

O Python também permite posições negativas, começando a contar do final para o começo:

```
s = "ola"
s[-1] == 'a'
s[-2] == 'l'
s[-3] == 'o'
```

Indexação

o	l	a
0	1	2
-3	-2	-1

Imutabilidade

A `string` em Python é dita imutável, pois não podemos alterar um elemento dela:

```
s = "ola"
s[0] = 'a' # ERRO!
```

Imutabilidade

Quando fazemos

```
s = "ola"
s = s + "!"
```

estamos na verdade criando uma nova `string` e descartando a antiga.

Slices

Outra forma de indexar é através dos `slices` em que determinamos uma faixa: `[inicio:fim:passo]`

```
s = "abcdefgh"
s[1:4] == "bcd"
s[2:5] == "cde"
s[:3]  == "abc"
```

Slices

Outra forma de indexar é através dos `slices` em que determinamos uma faixa: `[inicio:fim:passo]`

```
s = "abcdefgh"
s[6:] == ??
s[1:7:2] == ??
s[:]  == ??
```

Slices

Outra forma de indexar é através dos `slices` em que determinamos uma faixa: `[inicio:fim:passo]`

```
s = "abcdefgh"
s[6:] == "gh"
s[1:7:2] == "bdf"
s[:]  == "abcdefgh"
```

Slices

Um caso especial quando passamos apenas o último parâmetro com valor igual a −1:

```
s = "abcdefgh"
s[::-1] = "hgfedcba"
```

Exercício

Utilizando o comando `for` e a função `len`, faça uma função que imprime cada caracter em uma nova linha.

Exercício

```
def charPorLinha(s):
    for i in range(len(s)):
        print(s[i])
```

Iterando os caracteres

O comando `for`, na verdade, pode iterar qualquer conjunto de *coisas* automaticamente. O exercício anterior poderia ser escrito da seguinte forma:

```
def charPorLinha(s):
    for letra in s:
        print(letra)
```

Exercício

Crie uma função `tamanho` que faz a mesma coisa que `len` porém usando o código do slide anterior como base.

Exercício

Defina uma função `iguais` que determina se duas strings são iguais.

Exercício

Faça uma função para contar as vogais de uma string. Para isso utilize a instrução `in`:

```
s = "12345"
"5" in s == True
"6" in s == False
```

Exercício

Defina uma função `removeNaoAlfa` que recebe uma string e retorna outra string idêntica, porém sem caracteres não-alfabéticos.

Exercício

Faça uma função que recebe uma `string` representando um número binário e retorne o decimal correspondente.