

## Python - Seqüências

Uma estrutura de dados é uma coleção de dados organizados de alguma maneira. Estes dados podem ser números, caracteres ou outras estruturas de dados. A estrutura de dados mais básica em Python é a seqüência. Cada elemento de uma seqüência recebe um número que identifica sua posição dentro da seqüência. Chamamos este número de índice e ele inicia com zero. Assim, o primeiro elemento de uma seqüência em Python tem o índice zero, o segundo tem o índice 1, o terceiro tem o índice 2, e assim por diante.

Você pode até questionar porque os índices das seqüências em Python iniciam com zero e não com 1, como é normal. Uma das razões é porque utilizando este método você também pode indexar os elementos da seqüência a partir do final. O último elemento da seqüência é -1, o antepenúltimo é -2, e assim por diante. Assim, você pode contar, organizar ou manipular os elementos da seqüência a partir do primeiro ou do último elemento.

Existem seis tipos de seqüências em Python, porém as mais usadas são strings, listas e tuplas.

**strings** são conjuntos de caracteres que tem seus elementos todos juntos como por exemplo: "2Ae4F", "23 Jan 1960", "Seqüências em Python". Em Python as strings deve estar envolvidas em aspas simples, aspas duplas ou aspas triplas:

```
>>> "Isto e uma string"
'Isto e uma string'
>>> 'Isto tambem e uma string'
'Isto tambem e uma string'
>>> ''' E esta tambem e uma string'''
'E esta tambem e uma string'
>>>
```

As listas e tuplas são seqüências que tem seus elementos separados por vírgula. A diferença entre elas é que as **listas podem ser modificadas e as tuplas não**.

Seqüências podem ser úteis para manipular dados relacionados. Por exemplo, você pode ter uma seqüência representando os dados de uma pessoa com o primeiro elemento sendo o nome, o segundo a idade e o terceiro a altura. Usando uma **lista** a sintaxe seria:

```
>>> antonio = ["Jose Antonio",35,1.70]
>>> antonio
['Jose Antonio', 35, 1.7]
>>>
```

Observe que a **lista** é definida com seus elementos entre colchetes e separados por vírgula. Caso você quisesse usar uma **tupla** bastaria envolver os elementos em parênteses:

```
>>> antonio = ("Jose Antonio",35,1.70)
>>> antonio
('Jose Antonio', 35, 1.7)
>>>
```

Seqüências podem conter, inclusive, outras seqüências:

```
>>> antonio = ["Jose Antonio", 35, 1.70]
>>> maria = ["Maria Raimunda", 25, 1.60]
>>> manuel = ["Joaquim Manuel", 30, 1.75]
>>> povo = [antonio, maria, manuel]
>>> povo
[['Jose Antonio', 35, 1.70], ['Maria Raimunda', 25, 1.60, ['Joaquim Manuel',
30, 1.75]]
>>>
```

Observe que os elementos da lista *povo* são outras listas.

## Operações normalmente executadas com as seqüências

### Indexação

Como já foi citado anteriormente, os elementos de uma seqüência são indexados. O primeiro elemento tem índice zero, o segundo tem índice 1, o terceiro tem índice 2, e assim por diante. A indexação pode começar a partir do último elemento da seqüência, com este tendo índice -1, o antepenúltimo com índice -2 e assim por diante.

```
>>> manuel = ["Joaquim Manuel", 30, 1.75]
>>> manuel[0]
'Joaquim Manuel'
>>> manuel[1]
30
>>> manuel[2]
1.75
>>> manuel[-1]
1.75
>>> manuel[-2]
30
>>> manuel[-3]
'Joaquim Manuel'
>>> manuel[-0]
'Joaquim Manuel'
>>>
```

Observe que o índice -0 é igual ao índice 0.

### Corte

Você pode selecionar intervalos da seqüência usando operações de corte. A sintaxe para cortar uma seqüência é colocar os índices do início e do final do corte separados por dois pontos, entre colchetes.

```
>>> l = ["arroz","feijao","farinha","carne","tempero","cafe"]
>>> print l
['arroz', 'feijao', 'farinha', 'carne', 'tempero', 'cafe']
>>> l[1:4]
['feijao', 'farinha', 'carne']
>>>
```

Observe que o primeiro número indica o índice do primeiro elemento que você deseja inserir no corte e o segundo número indica o índice do elemento imediatamente após o corte. Por exemplo, digamos que você tenha a lista:

```
['arroz', 'feijao', 'farinha', 'carne', 'tempero', 'cafe']
```

e quer selecionar um corte entre o terceiro e o quinto elemento. O terceiro elemento (*farinha*) tem índice 2 e o quinto elemento (*tempero*) tem índice 4. Assim, nosso comando de corte ficaria assim:

```
l[2:5]
```

Observe:

```
>>> l = ["arroz","feijao","farinha","carne","tempero","cafe"]
>>> l[2:5]
['farinha', 'carne', 'tempero']
>>>
```

Existem ainda algumas observações interessantes quanto ao uso de cortes em seqüências. Elas são mostradas na figura abaixo. Observe com atenção. As linhas iniciadas com o caractere # são apenas comentários. De acordo com os índices passados, observe o subconjunto da lista que é retornado:

```
>>> l = ["arroz","feijao","farinha","carne","tempero","cafe"]
>>> l
['arroz', 'feijao', 'farinha', 'carne', 'tempero', 'cafe']
>>> # um corte normal, dentro do intervalo de índices
... l[2:5]
['farinha', 'carne', 'tempero']
>>> # índice do fim do corte além do final da seqüência
... l[2:10]
['farinha', 'carne', 'tempero', 'cafe']
>>> # índice inicial do corte antes do início da seqüência
... l[-2:5]
['tempero']
>>> l[-2:0]
[]
>>> l[-2:-10]
[]
>>> # deixando de especificar o segundo índice do corte
... l[2:]
['farinha', 'carne', 'tempero', 'cafe']
>>> # deixando de especificar o primeiro índice do corte
... l[:5]
['arroz', 'feijao', 'farinha', 'carne', 'tempero']
>>> l[:4]
```

```
['arroz', 'feijao', 'farinha', 'carne']
>>> # deixando de especificar os dois índices do corte
... l[: ]
['arroz', 'feijao', 'farinha', 'carne', 'tempero', 'cafe']
>>>
```

## Intervalo entre elementos num corte

Existe um terceiro parâmetro que pode ser usado nos cortes de seqüências e define o intervalo entre os elementos a serem retornados. Observe:

```
>>> l = ["I","II","III","IV","V","VI","VII","VIII","IX","X"]
>>> l
['I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX', 'X']
>>> l[2:9]
['III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX']
>>> # utilizando o terceiro parâmetro no corte
... l[2:9:2]
['III', 'V', 'VII', 'IX']
>>> l[2:9:1]
['III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX']
>>> # observe que o terceiro parâmetro com valor 1 já é implícito no Python
...
>>> # o terceiro parâmetro pode ter valor negativo
... l[9::-3]
['X', 'VII', 'IV', 'I']
>>> l[1::-2]
['II']
>>> l[8::-2]
['IX', 'VII', 'V', 'III', 'I']
>>> # observe que usando o terceiro parâmetro negativo, a seqüência é analisada
do fim para o começo
...
>>>
```

## Adicionando seqüências

Seqüências do mesmo tipo podem ser adicionadas usando-se o operador +.

```
>>> [1,2,3,4] + [5,6,7,8]
[1, 2, 3, 4, 5, 6, 7, 8]
>>> ["Trabalhando","com"] + ["sequencias","em","Python"]
['Trabalhando', 'com', 'sequencias', 'em', 'Python']
>>> [1,2] + ["Python","P"]
[1, 2, 'Python', 'P']
>>> [1.23,5,7] + ["Python","Linux"]
[1.23, 5, 7, 'Python', 'Linux']
>>> t = (1,2,3)
>>> t1 = ("a","b")
>>> t + t1
(1, 2, 3, 'a', 'b')
>>> "Juntando " + "strings"
'Juntando strings'
```

Porém, tentar adicionar seqüências de tipos diferentes gerará um erro:

```
>>> [1,2] + "Python"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "str") to list
>>> l = [1,2]
>>> t = (3,4)
>>> l + t
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "tuple") to list
```

## Multiplicando seqüências

Multiplicar uma seqüência por um número  $x$  cria uma nova seqüência na qual a seqüência original é repetida  $x$  vezes.

```
>>> "python" * 3
'pythonpythonpython'
>>> l = [1,2,3]
>>> l * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> [67] * 5
[67, 67, 67, 67, 67]
>>> (5,6,3) * 2
(5, 6, 3, 5, 6, 3)
>>>
```

## Lista vazia

Caso queira uma lista vazia, você pode usar, simplesmente, dois colchetes sem nada entre eles ( `[]` ). Porém, as vezes, você pode querer uma lista com determinado número de elementos e, inicialmente, querer que estes elementos não tenham nenhum valor. Neste caso você pode até fazer algo assim:

```
>>> ficha = [0] * 5
>>> ficha
[0, 0, 0, 0, 0]
>>>
```

Mas o Python tem um valor que pode lhe ajudar nestes casos, é *None*. *None* significa “nada”, então você poderia fazer algo assim:

```
>>> ficha = [None] * 5
>>> ficha
[None, None, None, None, None]
>>>
```

Fica mais limpo e profissional.

## Verificando a existência de um valor na seqüência

Para verificar a existência de um valor numa seqüência use o operador **in**. Se o valor existir na seqüência, ele retorna *True* (verdadeiro). Se o valor não existir na seqüência, ele retorna *False* (falso). Este tipo de operador, que retorna dois valores, um para verdadeiro e outro para falso, é chamado operador booleano. Abaixo você pode ver algumas utilizações do operador **in**.

```
>>> permissoes = 'rw'
>>> 'w' in permissoes
True
>>> 'x' in permissoes
False
>>> usuarios = ['joao','maria','jose']
>>> raw_input("Entre com o nome do usuario :") in usuarios
Entre com o nome do usuario :samuel
False
>>> raw_input("Entre com o nome do usuario :") in usuarios
Entre com o nome do usuario :maria
True
>>> url = 'www.python.com'
>>> 'Python' in url
False
>>> 'python' in url
True
>>>
```

Os primeiros dois exemplos poderiam ser usados para checar as permissões de um arquivo. O terceiro e quarto exemplos poderiam fazer parte de um programa que implemente uma certa segurança exigindo o nome de usuário para acesso. O mesmo pode ser feito com relação a senha. Os dois últimos exemplos poderiam fazer parte de uma rotina de proxy ou filtro de spam, por exemplo, onde os sites que possuem certa string devem ser identificados para o bloqueio do acesso.

## len, min, max

Estas três funções embutidas são muito úteis. **len** retorna o número de elementos da seqüência, **min** retorna o menor elemento e **max** retorna o maior elemento.

```
>>> numeros = [1967,1968,1969,1970]
>>> len(numeros)
4
>>> max(numeros)
1970
>>> min(numeros)
1967
>>> max(1,2,3,4,5)
5
>>> min(6,7,8,9)
6
>>>
```

## Exemplos

Abaixo você pode ver alguns exemplos que utilizam o que foi ensinado acima. Cada exemplo tem duas tabelas, a primeira mostra o código e a segunda mostra a execução do programa.

### 1. data.py

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

# data.py

# Recebe os números referentes a dia, mês, ano; e devolve
# uma data no formato "00 de aaa de 0000"

meses = ['Jan', 'Fev', 'Mar', 'Abr',
         'Mai', 'Jun', 'Jul', 'Ago',
         'Set', 'Out', 'Nov', 'Dez']

dia = raw_input("Dia      :")
mes = raw_input("Mês(1-12):")
ano = raw_input("Ano(0000):")

mes_nr = int(mes)

mes_nome = meses[mes_nr - 1]

print dia + " de " + mes_nome + " de " + ano
```

```
$ python data.py
Dia      :30
Mês(1-12):12
Ano(0000):1967
30 de Dez de 1967
$
```

### 2. url.py

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

# url.py

# Recebe uma URL no formato http://www.dominio.com e
# devolve o nome do dominio

# 0 1 2 3 4 5 6 7 8 9 10 11          -4 -3 -2 -1
# h t t p : / / w w w . d o m i n i o . c o m

url = raw_input("Entre com a URL :")

dominio = url[11:-4]
```



```
# Recebe um nome de usuario e a respectiva senha e verifica
# se este tem acesso ao programa

dados = [
    ['Joao', '1234'],
    ['Maria', '5678'],
    ['Jose', '90ab']
]

nome = raw_input("Usuario :")
senha = raw_input("Senha :")

if [nome, senha] in dados:
    print "Acesso permitido"
else:
    print "Acesso negado"
```

```
$ python login.py
Usuario :Samuel
Senha :25356
Acesso negado
$ python login.py
Usuario :Joao
Senha :1234
Acesso permitido
$
```

Bons estudos.

Samuel

[sdiasneto@yahoo.com.br](mailto:sdiasneto@yahoo.com.br)

<http://br.geocities.com/sdiasneto>