LISTAS

Neste vídeo, o expert **Allan Dieguez**, Head de Data Science da LuizaLabs, apresenta a manipulação de dados em sequências através de Listas.

09:05

ESTRUTURAS SEQUENCIAIS

Existem em **Python** algumas formas básicas de armazenar dados em grupo. Um dos tipos mais usados para construção de estruturas de dados são as chamadas **sequências**, grupo que compreende as **listas** *('list')* e as **tuplas** *('tuple')*, temas centrais desta aula.

As outras sequências básicas existentes na linguagem, que **não serão** abordadas nesta aula, são as *'strings'*, *'byte sequences'*, *'byte arrays'* e os *'range objects'*.

Para ser considerado uma sequência, um grupo de dados deve obedecer às seguintes regras:

dado, a **posição de um deles será maior** que a do outro. Isso permite que um elemento dentro de uma sequência seja **referenciado pela sua posição**.

02 Uma sequência pode ser composta de **elementos com valores repetidos**: cada elemento será tratado como um indivíduo e sua posição o descreve na sequência. Dessa forma, cada elemento dentro de uma sequência tem uma **posição** (número inteiro) e um **valor** (objeto python) associados.

LISTAS

Listas são a estrutura básica de sequência mais versátil. Essa estrutura **permite modificações do seu conteúdo** sem a necessidade da criação de outro objeto em memória, sendo portanto uma sequência **mutável** ou **dinâmica**.

O tipo interno *'type'* da lista em python é a *'list'*.

CRIAÇÃO DE LISTAS

A forma mais simples de criar uma lista é a declaração dos objetos que a compõem entre colchetes.

```
x = [1, 2, 3, 4, 5]
```

Uma outra forma de construir uma lista é **convertendo outra sequência** através do construtor list()

```
x = list((1,2,3,4,5))
```

elementos sejam do mesmo tipo, a lista é heterogênea por construção.

Veja o exemplo abaixo:

```
x = [None, 1, 'a', 'palavra completa', [None, 'x', 1999],
('nome', 'joao')]
```

ACESSO AOS DADOS DA LISTA

Acesso pela posição do item:

A forma mais simples de acessar um objeto da lista é pelo seu **índice**, ou seja, pela posição que ele ocupa dentro da sequência.

Vamos usar a lista abaixo para os próximos exemplos

```
x = ['primeiro', 'segundo', 'terceiro', 'quarto', 'quinto',
'penúltimo', 'último']
```

Os índices em Python começam do **zero**, então para acessar o primeiro elemento de uma lista devemos chamar lista[0], para acessar o segundo elemento: lista[1], e assim por diante.

Testando na lista que criamos anteriormente:

```
x[3]
‘quarto’
x[0]
‘primeiro’
```

Em python também é possível acessar os dados de uma lista através de um **índice negativo**. O índice -1 representa a última posição, o -2 a penúltima e assim por diante.

LISTAS E TUPLAS

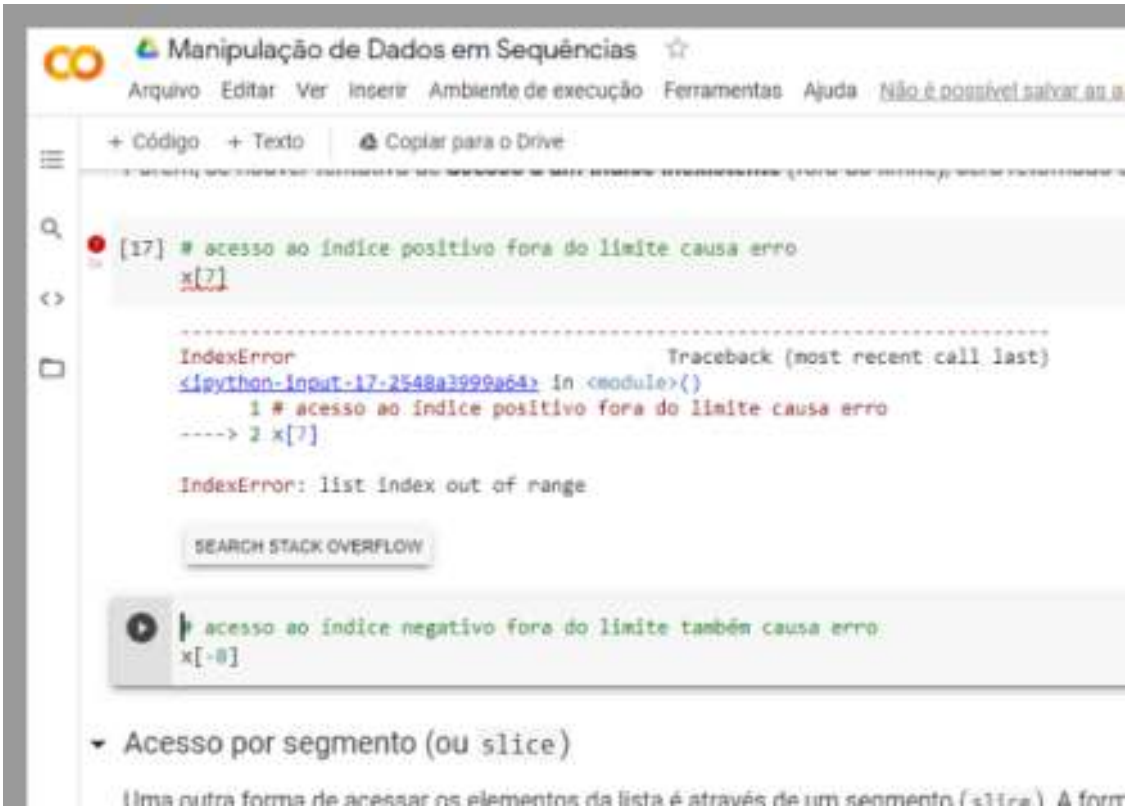
```
x[-2]
'penúltimo'
```

É possível acessar **todos os elementos** através de índices negativos.

```
x[-7]
'primeiro'
```

Se houver tentativa de acesso a um **índice inexistente** (fora do limite), será retornado um erro.

```
x[-7]
x[-8]
```



Acesso por segmento (ou *slice*):

através do operador **:** informando qual a **posição inicial** do segmento e também a **posição final**.

Esse seria o formato:

```
x[ <primeira posição> : <última posição>]
```

Vamos ver um exemplo:

```
x[2:5]
['terceiro', 'quarto', 'quinto']
```

Repare que o slice **incluiu** o elemento que estava na posição 2 mas **excluiu** o elemento que estava na posição 5. Este é comportamento padrão do Python. O código x[2:5] vai retornar os elementos x[2], x[3] e x[4]

Para acessar **o último elemento** da lista, é necessário **deixar vazio** o último número do *slice.*

```
x[5:]
['penúltimo', 'último']
```

Para o caso do primeiro índice, **tanto faz** usar o **valor vazio** ou **zero**.

```
x[0:3]
```

é o mesmo que

```
x[:3]
```

padrão, se o valor for omitido, se assume que é **igual a um**.

Veja o funcionamento do slice com este terceiro parâmetro.

```
x[1:5:2]
['segundo', 'quarto']
```

Resumo das formas que o Slice pode ter:

```
x[começo:fim:passo]
```

```
x[começo:fim]
```

```
x[começo:]
```

```
x[:fim]
```

```
x[:]
```

INFORMAÇÕES SOBRE A LISTA

Algumas informações gerais sobre a lista podem ser obtidas a partir das funções auxiliares mostradas a seguir.

Função len:

Essa função retorna a quantidade de elementos dentro de uma lista.

```
len(x)
7
```

As funções min e max já esperam que os itens dentro da lista sejam ordenáveis, pois é feita uma comparação entre todos para que seja retornado o mínimo ou máximo.
Exemplos:

```
min(['a', 'e', 'f', 'g', 'n', 'o', 'p', 'q'])
‘a’
max([0, 3, 1, 8, 10, 1, 20])
20
```

Função sum:

Caso seja uma lista de objetos do tipo numérico, é possível calcular a soma sobre a lista com a função sum().

```
sum([10, 20, 30])
60
```

INSERÇÃO E REMOÇÃO DE ELEMENTOS NA LISTA

A maior vantagem de uma lista é a possibilidade de adicionar ou remover elementos sem precisar criar outra lista.

Função append:

Essa função adiciona um elemento ao final da lista, aumentando sempre o tamanho em uma unidade. Vamos criar uma nova lista:

```
x = [1, 2, 3]
```

Para adicionar um elemento nesta lista podemos fazer:

```
x.append(4)
```

```
[1, 2, 3, 4]
```

Função insert:

Para inserir um elemento em outro lugar da lista que não seja só no final, é recomendado usar a função insert(). Para adicionar o número 0.42 no começo da lista, podemos fazer:

```
x.insert(0, 0.42)
```

Função remove:

A função remove() permite remover um elemento da lista usando como entrada **o elemento** que se quer remover. Se houver elementos repetidos, remove **a primeira ocorrência** do elemento, mantendo os outros na lista. Para remover mais, deve-se chamar novamente a função na lista.

Veja a seguinte lista:

```
x = [1,2,2,3,4]
```

Se chamarmos a função:

```
x.remove(2)
```

Teremos como resultado:

```
[1, 2, 3, 4]
```

LISTAS MULTIDIMENSIONAIS

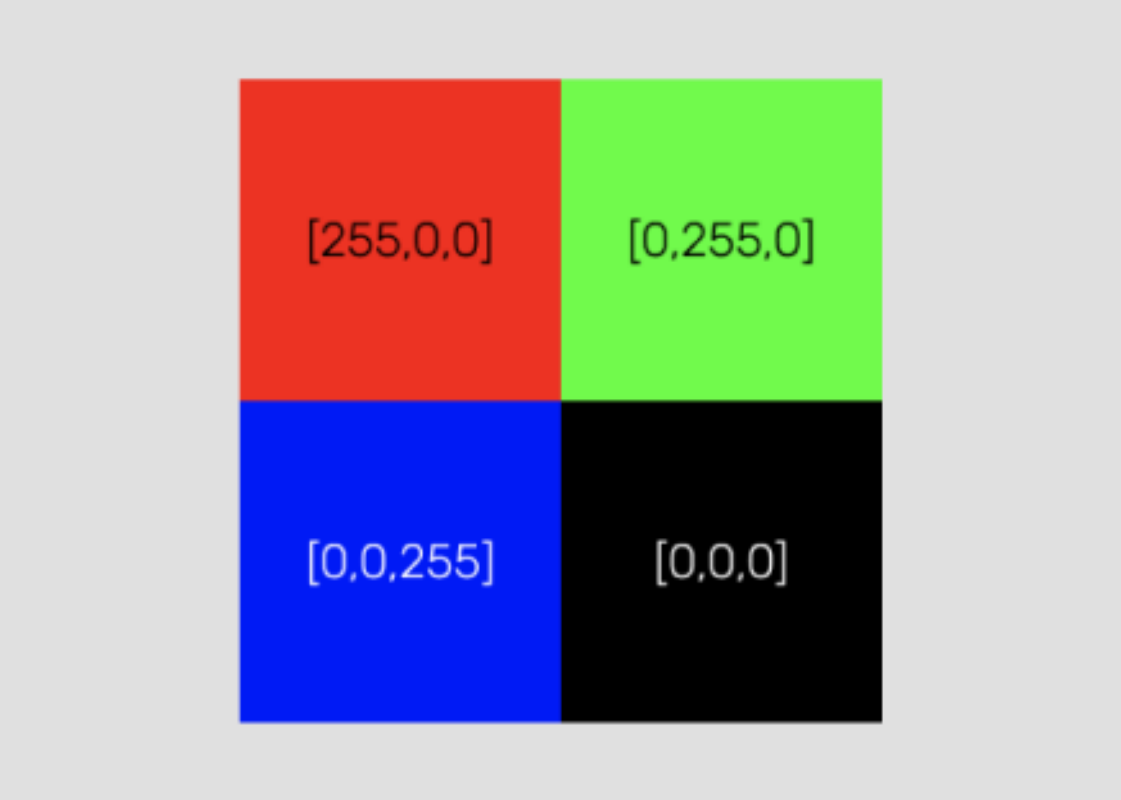
listas multidimensionais. Vamos ver sua utilidade neste caso prático:

Para representarmos uma cor numericamente podemos usar o sistema **RGB** - **R**ed (Vermelho), **G**reen (Verde), **B**lue (Azul). Associando um número de 0 a 255 para cada uma dessas cores podemos representar uma enorme quantidade de cores.

Para representar um pixel vermelho poderíamos utilizar a seguinte notação

```
[255,0,0]
```

E como faríamos para representar a imagem abaixo composta por esses 4 pixels?



Você pode pensar na melhor estratégia para fazer essa representação. Vamos ver um exemplo com duas dimensões:

```
imagem = [ [255,0,0], [0,255,0], [0,0,255], [0,0,0] ]
```

dimensão para cada linha:

Se contarmos a quantidade de elementos dessa variável através do len(imagem) o python retornará pra gente o número **4**.

Podemos contar também a quantidade de elementos dentro das listas que estão dentro da principal. Com o comando **len(imagem[0])** temos como retorno **3**.

AVANÇAR