Operações Avançadas com Listas

CIÊNCIA DA COMPUTAÇÃO

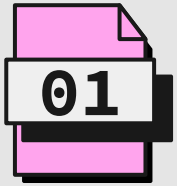
ATITUS EDUCAÇÃO

- Aula 09 -
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto

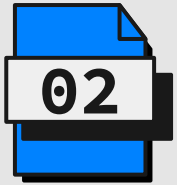
ATITUS EDUCAÇÃO

Na aula de hoje...



Operações Avançadas com Listas

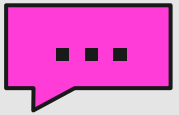
Concatenação
Cópia de Listas
Fatiamento



List Comprehensions

Sintaxe
Mapeamento
Filtragem

Aninhamento de Listas



Concatenação

* Concatenação é uma operação que permite unir duas listas (ou outras estruturas que veremos depois). A sintaxe é a seguinte:

```
new_list = list1 + list2
```

Cuidado! O operador de concatenação (+) é o mesmo de soma.

Por exemplo, considere duas listas `list1 = [1, 2, 3]` e `list2 = [4, 5, 6]`:

```
>>> list3 = list1 + list2
>>> list3
[1, 2, 3, 4, 5, 6]
>>> list3 + [7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
```

A operação de concatenação **não altera as listas** envolvidas, ela devolve uma **nova lista**.

```
>>> [-2, -1, 0] + list3
[-2, -1, 0, 1, 2, 3, 4, 5, 6]
>>> list3
[1, 2, 3, 4, 5, 6]
```

Note que após as operações de concatenação o conteúdo de `list3` permanece inalterado.

Cópia de Listas

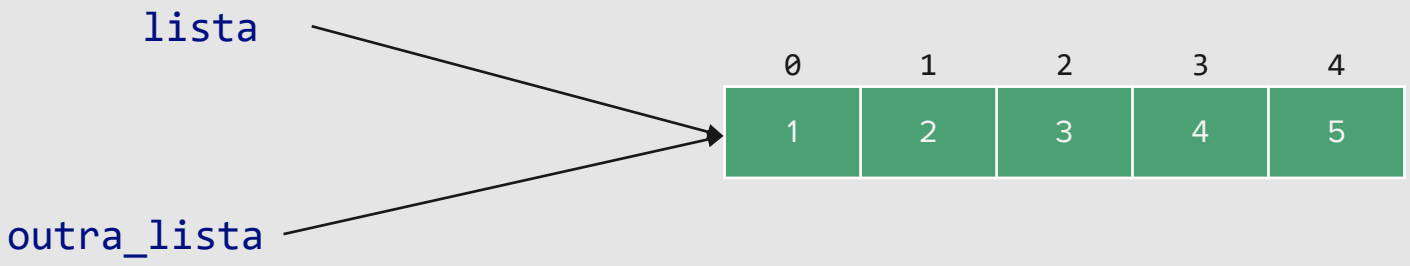
* O operador de atribuição funciona de uma forma diferente quando usamos listas.

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

outra_lista = lista
print("Antes da atribuição:")
print(f"lista = {lista} - outra_lista = {outra_lista}")

outra_lista[0] = "zero"

print("Depois da atribuição:")
print(f"lista = {lista} - outra_lista = {outra_lista}")
```

```
Antes da atribuição:
lista = [1, 2, 3, 4, 5] - outra_lista = [1, 2, 3, 4, 5]
Depois da atribuição:
lista = ['zero', 2, 3, 4, 5] - outra_lista = ['zero', 2, 3, 4, 5]
```

Cópia de Listas

Uma lista em Python é um objeto e, quando atribuímos um objeto a outro, estamos apenas copiando a mesma referência da lista, e não seus dados em si.



Podemos utilizar o fatiamento para realizar uma cópia de uma lista!

Fatiamento

* A notação de colch…em de uma lista através do seu índice.
* Ela também pode… sub-listas
* Por exemplo, consi…

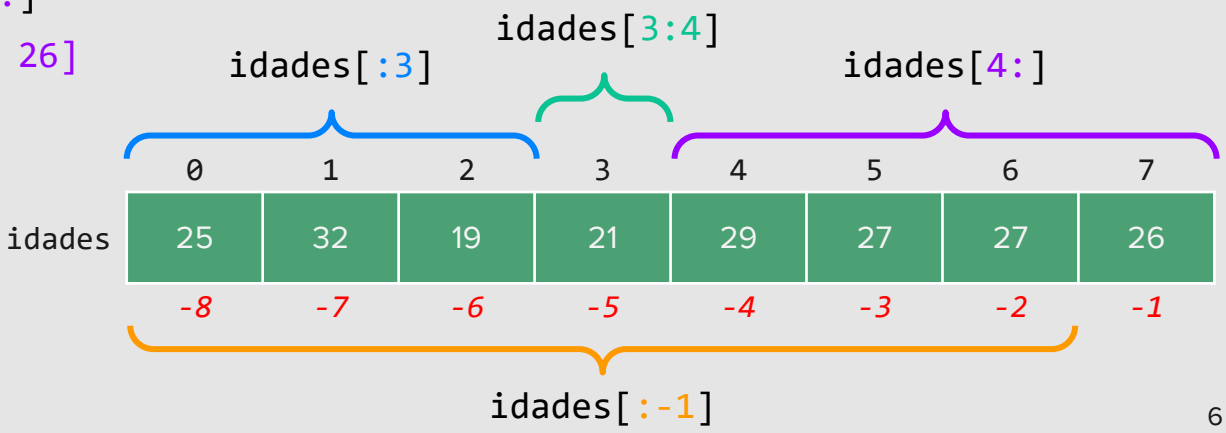
`<início>` e `<fim>` são **índices** da lista, ambos **opcionais** e o índice `<fim>`, quando informado, **não é incluído** na fatia.

```
>>> idades[3:4]
[21]
```

```
>>> idades[4:]
[29, 27, 27, 26]
```

```
>>> idades[:3]
[25, 32, 19]
```

```
>>> idades[:-1]
[25, 32, 19, 21, 29, 27, 27]
```



Fatiamento

```
primos = [2, 3, 5, 7, 11, 13, 17, 19]

primos[2:6]        [5, 7, 11, 13]

primos[:6]         [2, 3, 5, 7, 11, 13]

primos[6:]         [17, 19]

primos[:]          [2, 3, 5, 7, 11, 13, 17, 19]
```

Faz uma cópia da lista!

Fatiamento

* Podemos informar também um terceiro valor no fatiamento, que indica o passo em que os elementos serão selecionados.

```
primos = [2, 3, 5, 7, 11, 13, 17, 19]

primos[::2]          [2, 5, 11, 17]

primos[-1:-9:-1]   [19, 17, 13, 11, 7, 5, 3, 2]
```

Seleciona os índices da lista pulando de dois em dois

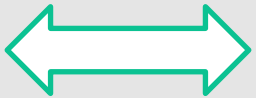
Fatiamento - Exercício

* Crie uma lista chamada `numeros` contendo os valores de 1 até 15 (use a função `range` para fazer isso), usando fatiamento faça as seguintes operações:
  a) Selecione somente os elementos 6, 7 e 8 da lista
  b) Os valores de 1 até 8 usando índices de início e fim
  c) Os valores de 1 até 8 usando somente o índice de fim
  d) Os três últimos valores da lista usando índices de início e fim
  e) Os três últimos valores da lista usando somente o índice de início
  f) Selecione somente os números ímpares da lista.
  g) Selecione os valores múltiplos de três presentes na lista
  h) A lista em ordem reversa

List Comprehensions

* Notação concisa (estilo funcional) para se criar novas listas.
* Consegue compactar em uma única linha um laço de repetição inteiro

```
lista = []
for i in range(1, 6):
    lista.append(i)

print(lista)
```

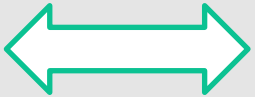


```
lista = [i for i in range(1, 6)]

print(lista)
```

List Comprehensions

* Notação concisa (estilo funcional) para se criar novas listas.
* Consegue compactar em uma única linha um laço de repetição inteiro

```
lista = []
for i in range(1, 6):
    lista.append(i)

print(lista)
```



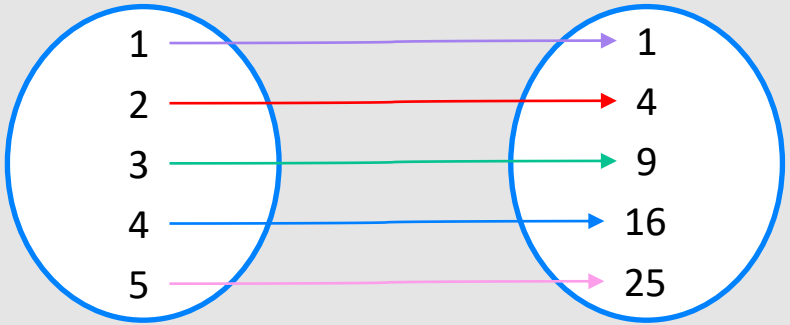
```
lista = [i for i in range(1, 6)]

print(lista)
```

Para cada valor `i`, gerado pelo laço for, a expressão à esquerda é avaliada e o valor resultante (nesse caso o próprio `i`) é inserido na lista.

List Comprehensions - Mapeamento

* A expressão em uma list comprehension pode realizar alguma tarefa
* Mapeamento consiste em aplicar alguma operação antes de atribuir o valor para a lista, o que produzirá uma lista do mesmo tamanho que a sequencia que está sendo mapeada, porém com valores diferente.

```
lista = [i ** 2 for i in range(1, 6)]
print(lista)
```



List Comprehensions - Filtragem

* Filtragem consiste em aplicar alguma condição nos valores gerados, de modo a produzir uma lista que contenha somente os valores que passarem por aquele filtro

```
lista = [i for i in range(1, 6) if i % 2 == 0]
print(lista)
```


```
lista = [2 4]
```

List Comprehensions

* Uma list comprehension pode processar outras listas:

```
cores = ["azul", "vermelho", "amarelo", "verde"]
novas_cores = [cor.upper() for cor in cores]

print(f"novas_cores = {novas_cores}")
print(f"cores = {cores}")
```

```
novas_cores = ['AZUL', 'VERMELHO', 'AMARELO', 'VERDE']
cores = ['azul', 'vermelho', 'amarelo', 'verde']
```

List Comprehensions

* Qual a lista produzida pelos seguintes comandos?

```
temperaturas = [88, 94, 97, 89, 101, 98, 102, 95, 100]

t1 = [t for t in temperaturas if t >= 100]
print(t1)

t2 = [(t - 32) * 5/9 for t in temperaturas]
print(t2)
```

```
[101, 102, 100]
[31.1, 34.4, 36.1, 31.6, 38.3, 36.6, 38.8, 35.0, 37.7]
```

List Comprehensions

* Escreva expressões de list comprehensions para:
  - Produzir uma lista com as consoantes que aparecem em uma variável string chamada `palavra`
  - Criar uma lista dos números entre 1 e 100 que são divisíveis por 3.
  - Criar uma lista de números chamada `valores_zero`, à partir de uma lista de números em ponto flutuante chamada `leituras`, selecionando somente os valores que estão a uma distância `epsilon` de zero.

Listas Aninhadas

* Podemos armazenar em listas valores de qualquer tipo, inclusive outras listas. Aninhar listas pode ser útil para uma série de aplicações. Por exemplo, podemos definir uma matriz 3x2 da seguinte forma:

```
matrix = [ [1,  3],
           [8, -1],
           [0, -3] ]
```

Essa notação em múltiplas linhas é apenas uma questão de **organização** do código, não tem efeito na funcionalidade.

Listas Aninhadas

* Todas as operações que vimos até agora se aplicam tanto à lista matrix quanto a qualquer uma das “sub-listas”, por exemplo:

```
>>> matrix.append([1, -1]) # insere uma nova lista ao final da matriz
>>> matrix[0].pop() # retira e retorna o último ítem da primeira sub-lista
3
```

O item foi retornado pelo método pop e agora a primeira sub-lista tem apenas 1 elemento.

```
# Imprimindo a lista no formato de matriz
for linha in matriz:
    for coluna in linha:
        print(f"{coluna:3}", end="")
    print()
```

```
1
8 -1
0 -3
1 -1
```

Listas Aninhadas

* O que será impresso por cada um dos seguintes comandos?

```
lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(lst[0])

print(lst[0][1])

print(lst[1])

print(lst[1][1])
```

```
[1, 2, 3]
2
[4, 5, 6]
5
```

Problema 1: Notas

Enunciado de um problema:

Ler e armazenar as 4 notas de cada um dos alunos de uma turma (10 alunos). Calcular e informar a média da primeira nota. Quantos alunos tem a primeira nota superior a esta média?

```
notas = [ [8.5, 7.4, 8.1, 9.2],  # Aluno 1
          [6.4, 7,1, 4.9, 5.3],  # Aluno 2
          ...
          [9.1, 3.9, 6.7, 8.3] ] # Aluno 10
```

```python
import random

NRO_ALUNOS = 10
NRO_NOTAS = 4

notas = []

for aluno in range(NRO_ALUNOS):
    notas_aluno = []
    for nota in range(NRO_NOTAS):
        #nt = float(input(f"Informe a nota {nota + 1} do aluno {aluno + 1}: "))
        nt = round(random.uniform(1, 10), 1)
        notas_aluno.append(nt)
    notas.append(notas_aluno)

# Cálculo da média da primeira nota
soma = 0
for aluno in range(NRO_ALUNOS):
    soma += notas[aluno][0]
media = soma / NRO_ALUNOS
print(f"Média da turma na primeira nota: {media:.2f}")
```

```python
# Cálculo da média da primeira nota
soma = 0
for aluno in range(NRO_ALUNOS):
    soma += notas[aluno][0]
media = soma / NRO_ALUNOS
print(f"Média da turma na primeira nota: {media:.2f}")

# Contando os alunos que estão acima da média
cont_alunos = 0
for aluno in range(NRO_ALUNOS):
    if notas[aluno][0] > media:
        cont_alunos += 1

print(f"{cont_alunos} alunos tiveram nota acima da média")
```

Problema 2: Operações

Enunciado de um problema:

Dada uma matriz inteira m com número de linhas igual a NL e número de colunas igual a NC, preenchê-la com números aleatórios de -10 a 10 e imprimir:

a) a média dos elementos de cada linha
b) o maior elemento de cada coluna da matriz
c) o produto de todos os elementos diferentes de zero
d) quantos elementos são negativos
e) posição ocupada (linha-coluna) por um elemento cujo valor será lido pelo programa (via teclado). Informar se houver mais de uma ocorrência, ou se o elemento não estiver presente na matriz