U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 73990
ADRIANO LIMA DA SILVA 223FE109 - 1
/ _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere FO=[2, 10, 12, 14, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FO[-6:5:3]
 b) FO[3:-1]
 c) FO[:4]
 d) FO[2:-2:2]
 e) FO[2:5:2]

2. Considere DN=[1, 9, 11, 15, 20, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DN[-6:6:2]
 b) DN[1:5]
 c) DN[:5:2]
 d) DN[-7:5:2]
 e) DN[2:-1:2]

3. Considere IK=[2, 6, 7, 9, 17, 22, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IK[2:7]
 b) IK[1:-3:2]
 c) IK[2:6:2]
 d) IK[-7:-3:3]
 e) IK[:6:-1]

4. Considere QB=[4, 8, 12, 14, 19, 20, 22, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QB[-7:6]
 b) QB[:8:2]
 c) QB[:7]
 d) QB[3:6]
 e) QB[:8:3]

5. Considere UB=[2, 9, 15, 18, 23, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UB[3:-1:2]
 b) UB[1:4:2]
 c) UB[:4:2]
 d) UB[2:5]
 e) UB[:4:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74005
ALANA KHRISSIE SILVA DE SOUZA 223FE109 - 2 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere XK=[1, 4, 15, 23, 24, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XK[-7:6:2]
 b) XK[3:-2:2]
 c) XK[2:5]
 d) XK[2:6:2]
 e) XK[3:7:2]

2. Considere JX=[4, 14, 19, 20, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JX[3:6:2]
 b) JX[3:-1]
 c) JX[-6:6:2]
 d) JX[:5:2]
 e) JX[1:-2:3]

3. Considere CV=[5, 13, 19, 22, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CV[1:5]
 b) CV[2:-2:2]
 c) CV[2:-1:3]
 d) CV[-7:-1:2]
 e) CV[3:6:3]

4. Considere TL=[3, 12, 19, 24, 25, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TL[2:5:3]
 b) TL[-6:-2]
 c) TL[1:6]
 d) TL[:7:3]
 e) TL[:5:2]

5. Considere OE=[1, 8, 11, 14, 16, 21, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OE[:5:2]
 b) OE[-7:-1]
 c) OE[:5:3]
 d) OE[1:5:3]
 e) OE[1:-1:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74012
ARTHUR ADAIR VIEIRA 223FE109 - 3 _____ / / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere XL=[5, 11, 13, 16, 20, 21, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XL[1:-1:2]
 b) XL[1:7]
 c) XL[1:8]
 d) XL[:8:2]
 e) XL[-7:6:3]

2. Considere EV=[2, 8, 10, 11, 20, 21, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EV[3:7]
 b) EV[1:-3]
 c) EV[:7:2]
 d) EV[:8]
 e) EV[2:6:2]

3. Considere VS=[2, 4, 17, 23, 25, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VS[-6:4]
 b) VS[3:5]
 c) VS[2:4:2]
 d) VS[2:-2:3]
 e) VS[:4]

4. Considere ZL=[3, 5, 6, 8, 16, 20, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZL[-6:7:2]
 b) ZL[:7]
 c) ZL[3:8]
 d) ZL[1:7:2]
 e) ZL[-8:8]

5. Considere LO=[4, 8, 11, 12, 20, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LO[-4:6:3]
 b) LO[-5:4]
 c) LO[2:5:2]
 d) LO[-4:4:3]
 e) LO[-6:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74029
ARTHUR HIDEIO NOGUTI 223FE109 - 4
/ _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere NR=[3, 8, 22, 23, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NR[1:6]
 b) NR[:4:2]
 c) NR[3:-2]
 d) NR[-5:6:3]
 e) NR[2:6:2]

2. Considere KC=[2, 7, 8, 11, 15, 16, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KC[-7:8:2]
 b) KC[:6:2]
 c) KC[-7:-2:3]
 d) KC[:8:2]
 e) KC[2:6]

3. Considere LA=[2, 5, 15, 17, 18, 23, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LA[-6:7]
 b) LA[2:6]
 c) LA[-6:5]
 d) LA[-7:-1:2]
 e) LA[1:6:2]

4. Considere HW=[6, 8, 13, 20, 22, 23, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HW[:5]
 b) HW[:5:2]
 c) HW[-6:5:3]
 d) HW[2:7:3]
 e) HW[:6:2]

5. Considere DF=[6, 8, 10, 17, 19, 23, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DF[2:-3:2]
 b) DF[:5:2]
 c) DF[:7:3]
 d) DF[2:6]
 e) DF[2:5:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74036
CAMILLA VIRGINIA RODRIGUES DA
223FE109 - 5 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere OB=[2, 5, 6, 10, 12, 14, 16]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OB[-6:5]
 b) OB[2:5:2]
 c) OB[-7:7]
 d) OB[1:6:3]
 e) OB[3:6:2]

2. Considere JN=[2, 5, 15, 24, 25, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JN[:6]
 b) JN[1:-3:3]
 c) JN[2:6:2]
 d) JN[2:6]
 e) JN[2:6:2]

3. Considere SQ=[3, 5, 9, 17, 19, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SQ[1:-3]
 b) SQ[-4:-3:2]
 c) SQ[-5:-3]
 d) SQ[:6]
 e) SQ[:4:-2]

4. Considere GV=[9, 15, 16, 17, 20, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GV[2:-3]
 b) GV[-4:6:3]
 c) GV[:6]
 d) GV[:6:2]
 e) GV[3:5]

5. Considere KP=[4, 8, 10, 11, 12, 20, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KP[-6:5:2]
 b) KP[-7:-2]
 c) KP[:5]
 d) KP[3:7:3]
 e) KP[:6]
```

Responda aqui:

1	2	3	4	5

109-74036 - /

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74043
CARLOS CRISTIAN XAVIER FARIAS
223FE109 - 6 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere ZK=[2, 10, 11, 12, 19, 21, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZK[1:-1:2]
 b) ZK[3:5:3]
 c) ZK[:7:2]
 d) ZK[2:6]
 e) ZK[2:7]

2. Considere WO=[5, 6, 7, 16, 20, 22, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) WO[:6:3]
 b) WO[:7:2]
 c) WO[3:5:2]
 d) WO[:6]
 e) WO[-5:7]

3. Considere VY=[1, 14, 18, 22, 24, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VY[3:-2]
 b) VY[:7]
 c) VY[1:5:2]
 d) VY[:6:3]
 e) VY[2:7:2]

4. Considere MK=[1, 2, 12, 23, 25, 26, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MK[1:-1]
 b) MK[:6:-2]
 c) MK[3:8]
 d) MK[-8:6:3]
 e) MK[:8]

5. Considere JD=[7, 10, 12, 13, 14, 23, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JD[:5:2]
 b) JD[3:-2]
 c) JD[3:5]
 d) JD[2:5:2]
 e) JD[3:-3:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74050
CASSIA BRUEL MORO DE OLIVEIRA
223FE109 - 7 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere DX=[13, 17, 21, 23, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DX[1:5:3]
 b) DX[-7:5]
 c) DX[-7:7:3]
 d) DX[3:-2:3]
 e) DX[3:-3]

2. Considere GV=[3, 8, 13, 14, 19, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GV[2:6:3]
 b) GV[1:5]
 c) GV[:5:2]
 d) GV[:4]
 e) GV[:4]

3. Considere RO=[2, 3, 7, 9, 13, 14, 19]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RO[1:-2:2]
 b) RO[:7]
 c) RO[1:6]
 d) RO[:6:2]
 e) RO[3:5:2]

4. Considere JK=[2, 6, 7, 10, 19, 20, 21, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JK[:6:-1]
 b) JK[3:-3:3]
 c) JK[2:-1]
 d) JK[2:6:2]
 e) JK[:6]

5. Considere EI=[2, 7, 11, 18, 23, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EI[3:4:2]
 b) EI[1:4:2]
 c) EI[:6:2]
 d) EI[-5:-1:3]
 e) EI[1:4:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74067
CRISTIAN PESSATTI DOS ANJOS 223FE109 - 8 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere AD=[4, 12, 14, 16, 22, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AD[1:4:2]
 b) AD[2:6:2]
 c) AD[:5:3]
 d) AD[-4:5:2]
 e) AD[3:6]

2. Considere VT=[5, 6, 9, 17, 19, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VT[2:7:2]
 b) VT[-7:7:2]
 c) VT[3:7:2]
 d) VT[:6]
 e) VT[:5:3]

3. Considere OQ=[3, 6, 10, 12, 16, 19, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OQ[1:7:3]
 b) OQ[-5:6]
 c) OQ[:5:-1]
 d) OQ[:7:2]
 e) OQ[-6:-3:2]

4. Considere BC=[3, 7, 11, 13, 14, 19, 20, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) BC[3:-2:2]
 b) BC[3:6:2]
 c) BC[3:6:2]
 d) BC[:6:2]
 e) BC[2:-1:2]

5. Considere UT=[3, 7, 8, 12, 16, 19, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UT[-7:-2]
 b) UT[2:-3:2]
 c) UT[:6]
 d) UT[:6]
 e) UT[:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74074
DIOGO BAYER CAVALLI 223FE109 - 9 _____ / / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere ZB=[1, 4, 5, 6, 16, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZB[:4]
 b) ZB[:5:2]
 c) ZB[3:4:2]
 d) ZB[-4:-2]
 e) ZB[:6:2]

2. Considere YE=[2, 10, 13, 14, 23, 25, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YE[-7:6:2]
 b) YE[:6]
 c) YE[:6:2]
 d) YE[-8:6:3]
 e) YE[1:7:3]

3. Considere UG=[3, 8, 13, 15, 21, 22, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UG[3:7:3]
 b) UG[:6]
 c) UG[-6:-3:3]
 d) UG[-6:-3]
 e) UG[:5:2]

4. Considere LA=[1, 7, 17, 18, 19, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LA[1:7:2]
 b) LA[3:-2:3]
 c) LA[2:6:2]
 d) LA[2:7]
 e) LA[:6:3]

5. Considere MO=[3, 4, 6, 8, 9, 19, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MO[1:-2:3]
 b) MO[:5:2]
 c) MO[3:7]
 d) MO[2:-3:2]
 e) MO[-6:7:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74081
EDILSON ADRIANO ROMANETCH 223FE109 - 10 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere CZ=[7, 9, 14, 18, 19, 21, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CZ[-6:7]
 b) CZ[3:7]
 c) CZ[2:6:2]
 d) CZ[:5:2]
 e) CZ[1:5:2]

2. Considere AL=[1, 5, 10, 14, 18, 20]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AL[1:6:2]
 b) AL[-6:-2:2]
 c) AL[:6:2]
 d) AL[3:-1:2]
 e) AL[:5:2]

3. Considere WS=[7, 10, 13, 14, 18, 26, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) WS[-8:-1]
 b) WS[1:7:3]
 c) WS[:6:2]
 d) WS[:7:2]
 e) WS[:6:-1]

4. Considere ZB=[7, 9, 10, 20, 24, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZB[:7:2]
 b) ZB[-6:6:2]
 c) ZB[:5]
 d) ZB[:6:2]
 e) ZB[-5:-2:2]

5. Considere SU=[2, 3, 6, 15, 19, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SU[1:6]
 b) SU[:4:2]
 c) SU[:6:2]
 d) SU[-4:-3]
 e) SU[3:6]
```

Responda aqui:

1	2	3	4	5

109-74081 - /

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74098
ERICK CINQUINI 223FE109 - 11 _____ / / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere AF=[9, 11, 17, 19, 22, 24, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AF[-5:6:2]
 b) AF[:7:2]
 c) AF[1:-1:2]
 d) AF[-6:7:2]
 e) AF[3:6:2]

2. Considere TN=[3, 10, 12, 16, 21, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TN[:5:2]
 b) TN[:4:2]
 c) TN[:4:-1]
 d) TN[-6:4]
 e) TN[3:-2]

3. Considere GI=[1, 3, 4, 10, 21, 23, 24, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GI[-7:-3]
 b) GI[3:7:2]
 c) GI[:6:2]
 d) GI[:7:3]
 e) GI[:7:3]

4. Considere QM=[6, 10, 17, 21, 23, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QM[3:7]
 b) QM[1:5]
 c) QM[:7:2]
 d) QM[-5:7:2]
 e) QM[3:5:2]

5. Considere XI=[4, 17, 24, 25, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XI[2:6:2]
 b) XI[-5:-1]
 c) XI[1:-3:2]
 d) XI[2:5:2]
 e) XI[-4:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74100
GABRIEL SOUZA BRAGA 223FE109 - 12
/ _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere BR=[1, 2, 21, 23, 24, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) BR[:4:2]
 b) BR[:6:2]
 c) BR[-6:-1:2]
 d) BR[:5]
 e) BR[3:5]

2. Considere CH=[4, 8, 11, 17, 20, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CH[1:5]
 b) CH[:4:2]
 c) CH[-4:5]
 d) CH[2:6:2]
 e) CH[:4:2]

3. Considere SW=[2, 8, 9, 10, 16, 20, 22, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SW[-8:-1:2]
 b) SW[3:-3:3]
 c) SW[:7:2]
 d) SW[-7:-3:2]
 e) SW[3:-3:2]

4. Considere OB=[10, 11, 12, 14, 15, 20, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OB[3:6]
 b) OB[:5:-2]
 c) OB[-6:-3:2]
 d) OB[-6:5]
 e) OB[1:-3]

5. Considere DP=[3, 12, 16, 20, 25, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DP[-5:5]
 b) DP[2:5:2]
 c) DP[-6:7:2]
 d) DP[2:5]
 e) DP[:5:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74117
GABRIEL VINICIUS RITT 223FE109 - 13
/ _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere KF=[2, 4, 6, 9, 16, 17, 21, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KF[1:6:3]
 b) KF[:6:2]
 c) KF[2:-3:2]
 d) KF[3:-3:2]
 e) KF[3:6:2]

2. Considere SQ=[1, 3, 10, 17, 19, 20, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SQ[1:-3:2]
 b) SQ[2:-2:2]
 c) SQ[:5:2]
 d) SQ[-5:5]
 e) SQ[2:7:3]

3. Considere DG=[3, 8, 11, 18, 20, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DG[2:7:2]
 b) DG[2:7:3]
 c) DG[:6:2]
 d) DG[-7:5]
 e) DG[2:5:2]

4. Considere JL=[2, 4, 5, 10, 13, 17, 21, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JL[-7:6:2]
 b) JL[:8]
 c) JL[1:-2]
 d) JL[:6:-2]
 e) JL[-6:6]

5. Considere KW=[2, 10, 13, 14, 15, 16, 21, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KW[:8:2]
 b) KW[:8:2]
 c) KW[2:-2:2]
 d) KW[-6:8:2]
 e) KW[3:7:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74124
GIULIA TEIXEIRA DUARTE 223FE109 - 14 / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere AR=[1, 2, 3, 5, 7, 11, 13, 20]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AR[3:8]
 b) AR[3:7:3]
 c) AR[-8:7]
 d) AR[-8:7:2]
 e) AR[-7:7]

2. Considere LK=[1, 10, 14, 15, 23, 24, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LK[-5:5]
 b) LK[3:7]
 c) LK[1:-3:2]
 d) LK[:7:2]
 e) LK[2:6:3]

3. Considere LA=[1, 2, 8, 13, 19, 21, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LA[1:-3]
 b) LA[:5:2]
 c) LA[-6:-3]
 d) LA[:7:2]
 e) LA[1:-1]

4. Considere HO=[5, 7, 11, 19, 24, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HO[3:-2:2]
 b) HO[:4:-1]
 c) HO[3:6]
 d) HO[-5:5]
 e) HO[1:5:2]

5. Considere VX=[1, 4, 15, 20, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VX[2:-3]
 b) VX[-7:7]
 c) VX[:6:2]
 d) VX[2:6:2]
 e) VX[-7:6]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74131
HENRIQUE PINTO MOREIRA MARCOND
223FE109 - 15 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere HX=[3, 5, 8, 14, 25, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HX[3:5:2]
 b) HX[3:7:2]
 c) HX[:6]
 d) HX[:6:2]
 e) HX[:6]

2. Considere IY=[1, 6, 8, 13, 16, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IY[1:5:2]
 b) IY[-7:5:2]
 c) IY[3:5]
 d) IY[:7]
 e) IY[3:-1]

3. Considere JL=[7, 9, 10, 11, 13, 18, 21, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JL[1:6]
 b) JL[3:7]
 c) JL[2:-2:2]
 d) JL[:8:3]
 e) JL[1:7:3]

4. Considere TE=[1, 3, 4, 7, 10, 12, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TE[2:-1:2]
 b) TE[:5:2]
 c) TE[-5:7:2]
 d) TE[:5:2]
 e) TE[1:5:3]

5. Considere XD=[9, 12, 14, 20, 21, 24, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XD[:6:2]
 b) XD[:7:2]
 c) XD[3:7]
 d) XD[:6:2]
 e) XD[:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74148
JANAINA KOVALSKI DOS SANTOS 223FE109 - 16 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere FW=[6, 16, 20, 21, 23, 24, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FW[2:8:2]
 b) FW[:6]
 c) FW[:7:2]
 d) FW[-8:-3]
 e) FW[:6:-1]

2. Considere KO=[2, 8, 9, 25, 26, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KO[3:-2:2]
 b) KO[-5:7:2]
 c) KO[1:-2:2]
 d) KO[2:5:2]
 e) KO[2:5:2]

3. Considere ZE=[5, 13, 14, 15, 16, 19, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZE[1:5:2]
 b) ZE[3:-3]
 c) ZE[-7:-3]
 d) ZE[:7:2]
 e) ZE[-7:-2:3]

4. Considere RQ=[3, 4, 12, 18, 20, 21, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RQ[:7:2]
 b) RQ[:7:3]
 c) RQ[:5:2]
 d) RQ[:7:2]
 e) RQ[1:6:2]

5. Considere QK=[6, 9, 23, 26, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QK[:4:3]
 b) QK[3:4:2]
 c) QK[2:6]
 d) QK[-6:4]
 e) QK[:5]
```

Responda aqui:

1	2	3	4	5

109-74148 - /

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74898
JHONATAN MARAFIGO M DA SILVA
223FE109 - 17 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere WQ=[1, 14, 15, 24, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) WQ[3:4:2]
 b) WQ[-5:4:2]
 c) WQ[-4:5]
 d) WQ[1:6:2]
 e) WQ[1:5]

2. Considere KW=[1, 4, 13, 22, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KW[:4:-2]
 b) KW[:4:-2]
 c) KW[:6]
 d) KW[:4:-3]
 e) KW[:5:3]

3. Considere XD=[6, 7, 10, 14, 16, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XD[-5:6:2]
 b) XD[3:5:2]
 c) XD[3:-3:2]
 d) XD[:6:2]
 e) XD[:6]

4. Considere PB=[8, 11, 13, 16, 17, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) PB[3:-2:3]
 b) PB[-5:4]
 c) PB[:4:2]
 d) PB[:6]
 e) PB[:4:-2]

5. Considere QU=[4, 6, 7, 18, 19, 21, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QU[-7:-2:2]
 b) QU[:6:2]
 c) QU[-6:6:2]
 d) QU[1:6:2]
 e) QU[1:-1]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74155
JONATAS ROBERTO KNAUBER 223FE109 - 18 / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere KX=[3, 10, 18, 21, 24, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KX[1:6:2]
 b) KX[-7:-1:2]
 c) KX[2:6:3]
 d) KX[-6:7:2]
 e) KX[1:-3:2]

2. Considere TO=[1, 3, 6, 7, 8, 11, 13, 18]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TO[:6]
 b) TO[3:-3]
 c) TO[1:7:3]
 d) TO[3:7]
 e) TO[2:-3]

3. Considere OB=[3, 4, 6, 11, 14, 15]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OB[2:-1]
 b) OB[:4:3]
 c) OB[:4]
 d) OB[1:4:2]
 e) OB[:4]

4. Considere ER=[6, 10, 14, 17, 18, 19, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ER[-7:8:2]
 b) ER[-8:8]
 c) ER[3:7]
 d) ER[-8:-1:2]
 e) ER[1:7:2]

5. Considere MU=[1, 5, 7, 10, 18, 22, 23, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MU[1:-1:2]
 b) MU[3:7:2]
 c) MU[:6:3]
 d) MU[3:-1]
 e) MU[1:8:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74162
JULIANA MOREIRA FELIX 223FE109 - 19
/ _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere NA=[2, 8, 13, 18, 20, 23, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NA[2:-1:2]
 b) NA[1:-1:2]
 c) NA[-8:8:2]
 d) NA[2:6:2]
 e) NA[2:8]

2. Considere EX=[4, 5, 8, 18, 21, 22, 23, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EX[:6:-1]
 b) EX[:6]
 c) EX[1:8:3]
 d) EX[:8:2]
 e) EX[1:-3]

3. Considere XE=[5, 6, 10, 12, 14, 18, 24, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XE[1:7]
 b) XE[-6:6]
 c) XE[-6:8:2]
 d) XE[-6:-2:2]
 e) XE[-6:6:3]

4. Considere SO=[7, 13, 15, 17, 19, 20, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SO[2:6]
 b) SO[-5:-3:3]
 c) SO[:7]
 d) SO[:7:2]
 e) SO[3:5]

5. Considere FQ=[8, 18, 22, 23, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FQ[:5:2]
 b) FQ[:6]
 c) FQ[3:5:3]
 d) FQ[1:-3]
 e) FQ[1:4:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74179
LEONARDO VIDAL BHERING 223FE109 - 20 / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere MQ=[2, 4, 9, 16, 17, 20, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MQ[3:7:2]
 b) MQ[:7:3]
 c) MQ[3:7:2]
 d) MQ[:5:3]
 e) MQ[-6:7:2]

2. Considere FJ=[2, 11, 14, 21, 22, 24, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FJ[2:-3:2]
 b) FJ[:5:2]
 c) FJ[-7:-2:2]
 d) FJ[-7:5]
 e) FJ[:6:2]

3. Considere JT=[2, 4, 5, 6, 9, 17, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JT[-5:6]
 b) JT[:7:2]
 c) JT[2:-3]
 d) JT[3:6:2]
 e) JT[3:-1]

4. Considere EA=[3, 4, 17, 19, 24, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EA[3:7]
 b) EA[3:5]
 c) EA[:7:2]
 d) EA[-5:5:2]
 e) EA[:6:2]

5. Considere NV=[2, 3, 9, 19, 23, 25, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NV[3:7:3]
 b) NV[1:7]
 c) NV[3:7:2]
 d) NV[2:7:2]
 e) NV[:5:-2]
```

Responda aqui:

1	2	3	4	5

109-74179 - /

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74186
LUCAS FERREIRA EPIFANIO MELO M
223FE109 - 21 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere KS=[5, 15, 16, 17, 24, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KS[1:-3:2]
 b) KS[-6:-1:2]
 c) KS[2:-3]
 d) KS[:7]
 e) KS[:6:2]

2. Considere HW=[2, 4, 5, 14, 16, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HW[:5]
 b) HW[:6:2]
 c) HW[-4:4:2]
 d) HW[:6:2]
 e) HW[3:6:2]

3. Considere UG=[4, 6, 12, 13, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UG[2:6]
 b) UG[-6:-2:2]
 c) UG[2:7]
 d) UG[2:5:3]
 e) UG[1:6]

4. Considere CW=[13, 16, 18, 19, 23, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CW[2:6:2]
 b) CW[-6:5:2]
 c) CW[2:6:2]
 d) CW[:4:-2]
 e) CW[:4:2]

5. Considere FO=[1, 6, 9, 10, 22, 23, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FO[:6]
 b) FO[3:5:2]
 c) FO[:5:-2]
 d) FO[:5:2]
 e) FO[3:7:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74193
MARCIANO CARLOS BARRETO 223FE109 - 22 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere GC=[3, 5, 12, 13, 17, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GC[-6:-2:2]
 b) GC[3:4:3]
 c) GC[3:-1:2]
 d) GC[:4:2]
 e) GC[1:4:2]

2. Considere BD=[2, 3, 14, 18, 19, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) BD[:5:2]
 b) BD[-5:6]
 c) BD[3:-1:2]
 d) BD[3:-1]
 e) BD[1:6:2]

3. Considere PD=[5, 10, 12, 16, 19, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) PD[:7:3]
 b) PD[-6:7]
 c) PD[1:6:2]
 d) PD[3:6]
 e) PD[2:5:2]

4. Considere HA=[8, 14, 16, 21, 23, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HA[1:6:2]
 b) HA[-6:-2:2]
 c) HA[-5:6:2]
 d) HA[-6:5]
 e) HA[:5:2]

5. Considere XB=[9, 17, 19, 21, 23, 24, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XB[-6:8:2]
 b) XB[:7]
 c) XB[1:-1]
 d) XB[2:6:2]
 e) XB[1:6:2]
```

Responda aqui:

1	2	3	4	5

109-74193 - /

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74205
MARIA EDUARDA MATIAS MORALES
223FE109 - 23 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere IO=[4, 8, 10, 11, 13, 15, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IO[:5:-2]
 b) IO[-7:6:2]
 c) IO[:5]
 d) IO[1:6:2]
 e) IO[:7:2]

2. Considere ME=[5, 6, 8, 9, 22, 24, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ME[3:5:2]
 b) ME[1:-1:2]
 c) ME[2:-2:2]
 d) ME[2:6:3]
 e) ME[1:5]

3. Considere YK=[8, 11, 13, 16, 19, 22, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YK[2:8:2]
 b) YK[-6:-1:2]
 c) YK[:8:2]
 d) YK[2:6:2]
 e) YK[-6:-1:2]

4. Considere IC=[4, 7, 14, 16, 24, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IC[2:5:2]
 b) IC[-7:6]
 c) IC[:7:2]
 d) IC[-5:-3:2]
 e) IC[-7:6]

5. Considere XA=[10, 13, 18, 22, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XA[:5]
 b) XA[2:-3:2]
 c) XA[3:-2:3]
 d) XA[-6:4]
 e) XA[3:4]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74900
MARIA EDUARDA V MACHADO 223FE109 - 24 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere AZ=[1, 6, 15, 22, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AZ[:6]
 b) AZ[1:-1]
 c) AZ[3:4:3]
 d) AZ[2:6:2]
 e) AZ[2:4]

2. Considere NT=[2, 3, 10, 12, 18, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NT[:4:-3]
 b) NT[3:-1:3]
 c) NT[-5:-1:2]
 d) NT[2:5:3]
 e) NT[3:4]

3. Considere WN=[3, 7, 8, 18, 20, 22, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) WN[:7]
 b) WN[:7:2]
 c) WN[3:6:2]
 d) WN[:6:2]
 e) WN[-8:7:2]

4. Considere IX=[2, 6, 8, 13, 18, 23, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IX[3:-3]
 b) IX[1:8:2]
 c) IX[:8]
 d) IX[1:6:2]
 e) IX[-6:7:3]

5. Considere HX=[5, 12, 15, 18, 22, 23, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HX[3:5:3]
 b) HX[:7:2]
 c) HX[1:7:2]
 d) HX[2:-1]
 e) HX[:5]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74212 NEY KAWAN DA SILVA 223FE109 - 25 _____ / / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere HC=[2, 4, 5, 7, 8, 22, 25, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HC[2:7:2]
 b) HC[-6:-1]
 c) HC[1:8:2]
 d) HC[3:6:3]
 e) HC[2:-1:3]

2. Considere CW=[7, 10, 14, 22, 23, 25, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CW[2:7:2]
 b) CW[:8:2]
 c) CW[2:-2]
 d) CW[:6:2]
 e) CW[1:6]

3. Considere OG=[5, 6, 9, 13, 15, 23, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OG[-7:-2:2]
 b) OG[3:-2]
 c) OG[1:7:2]
 d) OG[2:6]
 e) OG[-7:-1:2]

4. Considere DW=[4, 11, 19, 21, 24, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DW[2:6:2]
 b) DW[-5:-1]
 c) DW[3:7]
 d) DW[-6:6:2]
 e) DW[:6]

5. Considere JC=[18, 19, 20, 22, 23, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JC[3:6]
 b) JC[:5:3]
 c) JC[-6:-2]
 d) JC[:5:-2]
 e) JC[:7]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74229
NICOLAS BAGGIO MADEIRA 223FE109 - 26 / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere TM=[8, 17, 18, 19, 20, 22, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TM[:8:2]
 b) TM[-8:7:3]
 c) TM[3:8]
 d) TM[3:7:2]
 e) TM[1:-2]

2. Considere DR=[1, 5, 6, 15, 17, 19, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DR[1:6:3]
 b) DR[1:5:2]
 c) DR[2:7:2]
 d) DR[-7:6:2]
 e) DR[3:6:2]

3. Considere XE=[2, 11, 12, 17, 18, 20, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XE[:7:3]
 b) XE[:5:2]
 c) XE[3:5]
 d) XE[:7:2]
 e) XE[:6:2]

4. Considere MX=[2, 5, 10, 12, 16, 25, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MX[:6]
 b) MX[:8:3]
 c) MX[-8:-1:3]
 d) MX[1:6]
 e) MX[3:6:2]

5. Considere ZX=[1, 6, 17, 18, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZX[-5:5:2]
 b) ZX[3:6]
 c) ZX[2:6]
 d) ZX[-4:4:2]
 e) ZX[:4:-1]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74236
PAULO CEZAR PINZAN DE MATOS 223FE109 - 27 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere ZG=[3, 8, 9, 10, 11, 14]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZG[3:4]
 b) ZG[1:4:2]
 c) ZG[3:5]
 d) ZG[1:-1:2]
 e) ZG[:4]

2. Considere LE=[9, 13, 17, 18, 23, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LE[-5:6:3]
 b) LE[:5]
 c) LE[:4:3]
 d) LE[-4:6]
 e) LE[:6]

3. Considere EM=[3, 10, 12, 13, 16, 23, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EM[:5]
 b) EM[3:6:2]
 c) EM[3:7:2]
 d) EM[1:-3:2]
 e) EM[-5:-3]

4. Considere YR=[1, 5, 14, 19, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YR[-4:4]
 b) YR[-4:-1]
 c) YR[:5]
 d) YR[-5:-3:2]
 e) YR[:4:3]

5. Considere RC=[2, 3, 6, 7, 8, 11, 12, 20]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RC[3:6:2]
 b) RC[:8]
 c) RC[:6]
 d) RC[1:-1]
 e) RC[2:-1:3]
```

Responda aqui:

1	2	3	4	5

109-74236 - /

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74243
PEDRO HENRIQUE DANDREA 223FE109 - 28 / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere ZA=[2, 3, 4, 6, 21, 27, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZA[3:8]
 b) ZA[:8]
 c) ZA[2:6:2]
 d) ZA[-7:-2]
 e) ZA[3:7:2]

2. Considere YM=[2, 3, 12, 13, 15, 17, 20, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YM[2:8:2]
 b) YM[:6:2]
 c) YM[2:6:3]
 d) YM[3:8]
 e) YM[:6:2]

3. Considere XC=[1, 3, 13, 14, 15, 16, 21, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XC[-8:7:2]
 b) XC[-8:7]
 c) XC[:8:2]
 d) XC[1:7]
 e) XC[:7]

4. Considere NK=[1, 2, 6, 11, 13, 19, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NK[:6:2]
 b) NK[3:8:2]
 c) NK[:7:2]
 d) NK[:8:2]
 e) NK[-7:7:2]

5. Considere SM=[6, 8, 9, 11, 18, 23, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SM[-8:7]
 b) SM[:8]
 c) SM[3:7:2]
 d) SM[2:-3:2]
 e) SM[2:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74250
RAFAELLE RIBEIRO DIAS 223FE109 - 29
/ _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere OS=[3, 6, 7, 16, 23, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OS[2:7:2]
 b) OS[-7:5:2]
 c) OS[-7:5:2]
 d) OS[3:6:2]
 e) OS[-7:-2:2]

2. Considere TE=[5, 8, 15, 18, 19, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TE[:7:2]
 b) TE[2:7]
 c) TE[:6:2]
 d) TE[-7:-1:2]
 e) TE[3:7:3]

3. Considere ZC=[4, 9, 20, 22, 25, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZC[1:5:2]
 b) ZC[-5:6]
 c) ZC[:5:-2]
 d) ZC[:5:2]
 e) ZC[2:5]

4. Considere VI=[1, 2, 8, 9, 10, 14, 19, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VI[:6:3]
 b) VI[2:7:2]
 c) VI[:7:3]
 d) VI[:6:2]
 e) VI[-8:6:2]

5. Considere AZ=[1, 3, 16, 17, 19, 25, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AZ[:6:3]
 b) AZ[-7:6]
 c) AZ[2:-2]
 d) AZ[1:-3]
 e) AZ[2:-2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74267
STEFANY PINHEIRO 223FE109 - 30 _____ / / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere CB=[3, 5, 15, 20, 24, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CB[:5]
 b) CB[3:6]
 c) CB[:6]
 d) CB[1:6]
 e) CB[1:7]

2. Considere JS=[1, 7, 9, 10, 17, 20, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JS[-6:-2:3]
 b) JS[:8]
 c) JS[2:-1:3]
 d) JS[-7:-2]
 e) JS[2:-3]

3. Considere UK=[7, 10, 11, 13, 15, 19, 23, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UK[3:7:2]
 b) UK[1:6:2]
 c) UK[2:7:2]
 d) UK[3:6]
 e) UK[3:-2:2]

4. Considere CK=[3, 8, 9, 14, 17, 23, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CK[1:-3:2]
 b) CK[-7:6]
 c) CK[-6:7]
 d) CK[1:-2:2]
 e) CK[1:-1:2]

5. Considere AQ=[2, 5, 8, 13, 17, 18, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AQ[:5:2]
 b) AQ[:6]
 c) AQ[:5:-1]
 d) AQ[-7:6:2]
 e) AQ[2:6:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74274
VICTOR PINHEIRO COSTA LOSS 223FE109 - 31 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere MA=[2, 8, 9, 17, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MA[:4:2]
 b) MA[2:5]
 c) MA[3:4:2]
 d) MA[2:5]
 e) MA[1:5:2]

2. Considere XZ=[1, 4, 7, 11, 14, 15]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XZ[-6:-1]
 b) XZ[:6:3]
 c) XZ[:5]
 d) XZ[-4:6]
 e) XZ[:4:-1]

3. Considere OY=[2, 8, 14, 20, 24, 25, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OY[3:6]
 b) OY[:7:2]
 c) OY[2:-1:3]
 d) OY[:6:2]
 e) OY[3:6:2]

4. Considere LT=[7, 14, 15, 16, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LT[3:6:2]
 b) LT[2:-2]
 c) LT[3:4:2]
 d) LT[1:-1]
 e) LT[:6]

5. Considere HJ=[6, 13, 14, 16, 22, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HJ[3:-1]
 b) HJ[-4:5:2]
 c) HJ[:4:2]
 d) HJ[:4:3]
 e) HJ[2:-1:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74281
VITOR ALBERTI LIMA 223FE109 - 32 _____ / / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere UG=[1, 5, 9, 11, 17, 21, 22, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UG[2:6:2]
 b) UG[:8:2]
 c) UG[3:7]
 d) UG[2:-2:2]
 e) UG[:8:3]

2. Considere FL=[3, 10, 21, 22, 26, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FL[:4:-2]
 b) FL[:4:2]
 c) FL[:4:2]
 d) FL[:6]
 e) FL[1:-2]

3. Considere TW=[4, 12, 13, 15, 19, 20, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TW[3:5:2]
 b) TW[3:7]
 c) TW[:7:2]
 d) TW[-5:5:2]
 e) TW[2:-2]

4. Considere OX=[5, 7, 10, 14, 18, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OX[-5:6:3]
 b) OX[3:-2:2]
 c) OX[:4]
 d) OX[:4]
 e) OX[2:5:2]

5. Considere HN=[14, 15, 19, 20, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HN[:4:2]
 b) HN[-4:-2:3]
 c) HN[3:-2]
 d) HN[-5:-3]
 e) HN[3:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74298
VITOR PRADO ZAMBALDI 223FE109 - 33 / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere DX=[4, 7, 14, 15, 18, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DX[:7:3]
 b) DX[1:7:2]
 c) DX[:5:-2]
 d) DX[1:6]
 e) DX[-7:5]

2. Considere MP=[5, 7, 15, 17, 18, 24, 26, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MP[-8:7:2]
 b) MP[:6:-1]
 c) MP[:6:-1]
 d) MP[3:8]
 e) MP[1:-3:2]

3. Considere AN=[3, 17, 18, 19, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[2:4:3]
 b) AN[-6:-2]
 c) AN[2:4]
 d) AN[:6]
 e) AN[2:-3:2]

4. Considere TP=[9, 14, 15, 16, 23, 24, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TP[3:-3:3]
 b) TP[:7]
 c) TP[:7:2]
 d) TP[:8:3]
 e) TP[:6]

5. Considere TW=[1, 3, 6, 7, 13, 21, 23, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TW[-7:-2:2]
 b) TW[:6:2]
 c) TW[:8]
 d) TW[-6:-2:2]
 e) TW[2:8:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74300
WELLITON DA SILVA OLIVERIO 223FE109 - 34 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
    LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere UE=[3, 14, 16, 20, 21, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UE[2:6:2]
 b) UE[:4:-2]
 c) UE[-4:6]
 d) UE[:4:-1]
 e) UE[-4:5]

2. Considere RD=[5, 9, 13, 16, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RD[2:6:2]
 b) RD[3:4]
 c) RD[3:6:2]
 d) RD[:5:2]
 e) RD[2:4]

3. Considere JN=[1, 4, 9, 11, 14, 19, 22, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JN[-6:-1:2]
 b) JN[2:-1]
 c) JN[3:6:3]
 d) JN[1:8]
 e) JN[2:7:2]

4. Considere ZJ=[6, 7, 8, 14, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZJ[-5:4]
 b) ZJ[3:6]
 c) ZJ[1:-2:3]
 d) ZJ[-4:4:2]
 e) ZJ[-5:6]

5. Considere QE=[1, 12, 19, 22, 23, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QE[-6:-1:2]
 b) QE[:7]
 c) QE[2:6:2]
 d) QE[3:-2]
 e) QE[2:5]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 05/12/2022 - 09:05:28.6
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 74317
WESLLEY ALEXANDRE DA SILVA 223FE109 - 35 _____ / _____ / _____

Listas em Python

Uma lista em Python é uma coleção de coisas. Os elementos individuais podem ser acessados pelo seu índice dentro da lista. Em outras linguagens (C, Java, C++, APL, PHP, etc) este conceito é conhecido como vetor, e nelas todos os elementos têm que ter o mesmo tipo. Essa restrição não existe em Python, mostra de sua modernidade. Em C, uma lista de inteiros só pode ter inteiros. Em Python uma lista pode ter inteiros, nomes, caracteres, todos misturados.

Uma lista em Python é caracterizada pela presença dos colchetes delimitando os elementos que são separados por uma vírgula. Por exemplo `A=[1, 5, 7, 90]`. Aqui, a lista de nome `A` tem 4 elementos, a saber `1`, `5`, `7` e `90`.

A lista pode conter zero elementos, o que se conhece como lista vazia, e ela é criada assim a partir da definição ou seja `A=[]`. Novos elementos podem ser adicionados ao final de uma lista já existente, usando-se a operação `append`. Acompanhe o exemplo:

```
>>> A = []
>>> A.append(5)
>>> A
[5]
>>> A.append('viva')
>>> A
[5, 'viva']
```

Indexação

Os elementos individuais de uma lista podem ser acessados (tanto para leitura quanto para gravação) usando-se um índice. Indices positivos vão da esquerda para a direita e negativos da direita para a esquerda. O primeiro elemento da lista à esquerda é o elemento zero. O segundo é o 1, o terceiro o 2 e assim por diante até o $n-1$-ésimo que o último elemento de uma lista de n elementos.

Já no sentido oposto, o último elemento é o -1, o penúltimo é o -2 e assim por diante até o $-n$ que é o primeiro elemento de uma lista de n elementos.

Acompanhe o que se disse no exemplo:

```
     LA  = [  5, 33, 21, -4, 126.7, 'oba', 18]
índice >= 0:  0   1   2   3     4      5    6
índice <0:   -7  -6  -5  -4    -3     -2   -1
```

A lista pode ser consultada, como em `>>>LA[3]` que é igual a `-4` como pode ser alterada, fazendo-se `LA[1]=100`, o que vai substituir o valor 33 pelo valor 100 dentro da lista.

Fatiamento

É uma operação muito comum em Python envolvendo partes (fatias) de listas e que é fortemente baseada na indexação de listas. O formato de um fatiamento é

```
inicio:final:incremento
```

Os 3 valores podem ser omitidos, e quando o incremento é omitido, omite-se também o segundo `:`. O `início` indica qual o elemento inicial da fatia, e o `final` indica o seguinte ao último elemento da fatia. Quando `início` é omitido, o Python assume o primeiro elemento da lista, quando o `final` é omitido, assume-se o último e quando o `incremento` é omitido assume-se o valor 1. Acompanhe e procure entender cada um dos exemplos a seguir.

```
>>> LB=[5, 17, 33, 80, 91]
>>> LB[:-1]
[5, 17, 33, 80]
>>> LB[1:-1]
[17, 33, 80]
>>> LB[::2]
[5, 33, 91]
>>> LB[::-1]
[91, 80, 33, 17, 5]
>>> LB[::-2]
[91, 33, 5]
>>> LB[1:3]
[17, 33]
>>> LB[-3:-1]
[33, 80]
>>> LB[:3]
[5, 17, 33]
>>> LB[3:]
[80, 91]
>>> LB[:]
[5, 17, 33, 80, 91]
```

Para operar este conceito é interessante desenhar um esquema identificando e numerando os limites de cada elemento. Faça isso marcando os colchetes e as vírgulas da lista, positivos da esquerda para a direita começando em zero e negativos da direita para a esquerda. Agora imagine o número fornecido como uma faca cortando uma torta no local indicado. Veja:

```
LB = [   5,  17,  33,  80, 91 ]
     |    |    |    |    |    |
     0    1    2    3    4    5
    -6   -5   -4   -3   -2   -1
```

Outras operações

lista.insert(índice, elemento): Insere o elemento dado na lista citada, após o índice especificado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.insert(3, 123)
>>> LC
[5, 33, 9.5, 123, 'oba', 'além']
```

lista.remove(elemento): Remove o primeiro elemento citado. Esta operação não retorna nada.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.remove(9.5)
>>> LC
[5, 33, 'oba', 'além']
```

lista.pop([índice]): Remove e DEVOLVE o elemento citado. Como o índice é opcional, se omitido, remove e devolve o último elemento.

```
>>> LC=[5, 33, 9.5, 'oba', 'além']
>>> LC.pop()
'além'
>>> LC
[5, 33, 9.5, 'oba']
```

lista.sort(): Ordena a lista no local.
lista.reverse(): Reverte a lista no local
len(lista): Retorna o tamanho da lista
min(lista): Retorna o valor mínimo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
max(lista): Retorna o valor máximo da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.
sum(lista): Retorna a soma da lista. Só funciona em listas homogêneas contendo números. Senão dá erro.

```
>>> LD=[1, 4, 6.8, 9]
>>> len(LD)
4
>>> min(LD)
1
>>> max(LD)
9
>>> sum(LD)
20.8
```

Estas são ALGUMAS das operações envolvendo listas. Para ver tudo, consulte a documentação do Python.

Exemplo

A seguir uma instância feita e correta para os exercícios.

```
1. Considere GO=[3, 8, 11, 12, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GO[-6:5:2]
 b) GO[1:-2:2]
 c) GO[2:-3:2]
 d) GO[1:6:3]
 e) GO[3:4:2]

2. Considere OH=[10, 15, 21, 22, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[-6:4]
 b) OH[1:4:2]
 c) OH[-4:6:3]
 d) OH[3:5]
 e) OH[2:-3:2]

3. Considere AN=[1, 5, 9, 14, 18, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AN[-6:8:3]
 b) AN[:6:2]
 c) AN[1:-2:2]
 d) AN[3:-3:2]
 e) AN[-8:6:2]

4. Considere TX=[1, 2, 4, 14, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TX[:8:2]
 b) TX[:7:2]
 c) TX[2:-2:2]
 d) TX[1:7]
 e) TX[-6:-1:2]

5. Considere
   CF=[3, 6, 8, 10, 15, 19, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CF[:9]
 b) CF[:7]
 c) CF[1:8:2]
 d) CF[2:-2:2]
 e) CF[3:-2]

Respostas deste exemplo: 117 220 146 236 393
```

☞ Para você fazer

```
1. Considere UM=[10, 14, 15, 23, 25, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UM[-5:-1:3]
 b) UM[:5:3]
 c) UM[:6]
 d) UM[:5:3]
 e) UM[3:5:2]

2. Considere KW=[10, 11, 16, 18, 19, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KW[-5:-3:2]
 b) KW[2:-2:2]
 c) KW[:4:3]
 d) KW[:4]
 e) KW[-6:5:2]

3. Considere MR=[2, 9, 19, 20, 21, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MR[2:-3:3]
 b) MR[-4:4:2]
 c) MR[1:4:2]
 d) MR[2:4:2]
 e) MR[:5]

4. Considere KP=[4, 5, 12, 14, 18, 19, 20, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KP[:7:2]
 b) KP[-8:7]
 c) KP[:7:2]
 d) KP[1:8:2]
 e) KP[:6]

5. Considere ID=[1, 3, 19, 20, 24, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ID[3:-1]
 b) ID[2:5:3]
 c) ID[2:-2:2]
 d) ID[-5:7:2]
 e) ID[2:6:2]
```

Responda aqui:

1	2	3	4	5

109-74317 - /