U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69215
ALLAN MORINAGA 19FFU108 - 1 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 ZE=[2, 7, 9, 12, 14, 15, 22, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZE[1:-2]
 b) ZE[:8:3]
 c) ZE[2:-2]
 d) ZE[:6:2]
 e) ZE[:6:2]

2. Considere LX=[10, 13, 14, 21, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LX[:6]
 b) LX[3:-1:2]
 c) LX[:5]
 d) LX[:5:2]
 e) LX[1:5:2]

3. Considere GX=[5, 9, 11, 14, 20, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GX[-4:-3:2]
 b) GX[-5:4:3]
 c) GX[1:4:3]
 d) GX[:4:2]
 e) GX[-4:-1:2]

4. Considere GS=[1, 8, 14, 20, 21, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GS[-5:-1]
 b) GS[-4:5:2]
 c) GS[1:-1:2]
 d) GS[:4:2]
 e) GS[1:-1:3]

5. Considere QI=[4, 7, 19, 20, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QI[:5:3]
 b) QI[:6]
 c) QI[2:6:3]
 d) QI[:5:2]
 e) QI[:4]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69822
ANDREA A ALVES 19FFU108 - 2 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 MG=[3, 7, 9, 17, 19, 22, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MG[1:-2:2]
 b) MG[-8:-1:2]
 c) MG[2:8:2]
 d) MG[:7:2]
 e) MG[-8:-3:2]

2. Considere PK=[7, 9, 11, 18, 23, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) PK[1:7]
 b) PK[2:6:2]
 c) PK[3:5]
 d) PK[2:7]
 e) PK[:5:-2]

3. Considere DB=[3, 9, 10, 19, 21, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DB[:6:3]
 b) DB[1:5:2]
 c) DB[-6:-2]
 d) DB[2:4:2]
 e) DB[1:6:2]

4. Considere QF=[6, 13, 19, 24, 25, 26, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QF[2:-1:3]
 b) QF[-7:-2:2]
 c) QF[-7:8:3]
 d) QF[3:8:3]
 e) QF[-6:-3:3]

5. Considere VW=[4, 7, 15, 16, 18, 20, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VW[:6:2]
 b) VW[2:-1]
 c) VW[:5:-2]
 d) VW[2:7:2]
 e) VW[1:7:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69084
EDSON RODRIGUES VELOSO 19FFU108 - 3
apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 ZP=[2, 5, 6, 11, 14, 21, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZP[-6:5:3]
 b) ZP[2:-1:2]
 c) ZP[-5:-3:2]
 d) ZP[-6:6:2]
 e) ZP[3:5]

2. Considere OY=[3, 6, 10, 11, 17, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OY[-8:8:2]
 b) OY[-8:-3:2]
 c) OY[:6:3]
 d) OY[:7:3]
 e) OY[1:8:3]

3. Considere AH=[2, 3, 5, 8, 23, 24, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AH[:7:3]
 b) AH[-8:8:2]
 c) AH[:7]
 d) AH[1:7:3]
 e) AH[:6:2]

4. Considere XI=[1, 4, 10, 11, 14, 15, 16]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XI[:5:-2]
 b) XI[-7:-3]
 c) XI[:6:2]
 d) XI[-6:7:2]
 e) XI[3:7]

5. Considere TA=[4, 7, 9, 17, 24, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TA[:7:2]
 b) TA[-6:6:2]
 c) TA[3:-2:2]
 d) TA[:5:3]
 e) TA[3:7:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69222
ELAINE GREBOGY (OUV) 19FFU108 - 4 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 YM=[3, 6, 8, 17, 21, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YM[:6:2]
 b) YM[1:-2]
 c) YM[:6]
 d) YM[2:5:2]
 e) YM[-4:-1:2]

2. Considere KR=[1, 3, 5, 13, 19, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KR[1:-3]
 b) KR[3:5:3]
 c) KR[2:4]
 d) KR[3:6:2]
 e) KR[1:4:2]

3. Considere VS=[3, 5, 16, 17, 20, 21, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VS[:6:-1]
 b) VS[3:-2:2]
 c) VS[1:8:2]
 d) VS[:7]
 e) VS[-7:-1:2]

4. Considere RU=[3, 4, 6, 7, 19, 20, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RU[-6:6:2]
 b) RU[3:-2:2]
 c) RU[1:6:2]
 d) RU[-5:6:2]
 e) RU[:7:2]

5. Considere UF=[1, 4, 12, 18, 24, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UF[:7:2]
 b) UF[:5]
 c) UF[:6:3]
 d) UF[3:-1]
 e) UF[-5:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69091
FAGNER FERREIRA 19FFU108 - 5 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 NL=[3, 8, 12, 14, 21, 24, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NL[2:-3:3]
 b) NL[2:-3:2]
 c) NL[:7]
 d) NL[-6:8]
 e) NL[:6]

2. Considere CM=[1, 4, 5, 11, 14, 15, 23, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CM[3:7:2]
 b) CM[-8:-3]
 c) CM[1:8:2]
 d) CM[:6:2]
 e) CM[-8:-1:2]

3. Considere FI=[1, 4, 10, 11, 13, 21, 22, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FI[3:-2]
 b) FI[3:6]
 c) FI[1:8:2]
 d) FI[:7]
 e) FI[:7]

4. Considere JU=[3, 6, 19, 22, 23, 24, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JU[1:6:2]
 b) JU[1:8:2]
 c) JU[:7:2]
 d) JU[:8:2]
 e) JU[2:7]

5. Considere QL=[2, 9, 10, 21, 22, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QL[2:5:2]
 b) QL[-5:-3:2]
 c) QL[-6:5]
 d) QL[3:7:2]
 e) QL[3:6]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69103
FELLIPE RICHTER NEVES 19FFU108 - 6 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 XD=[1, 11, 12, 17, 18, 21, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XD[-7:7:3]
 b) XD[1:6:2]
 c) XD[:6]
 d) XD[3:-3:2]
 e) XD[:6:3]

2. Considere WF=[6, 9, 14, 17, 19, 21, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) WF[-7:-1]
 b) WF[:6:3]
 c) WF[:7:2]
 d) WF[:5:-1]
 e) WF[-7:7:2]

3. Considere ZH=[2, 7, 11, 15, 17, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZH[-5:6:2]
 b) ZH[-7:6:2]
 c) ZH[2:6:2]
 d) ZH[:7:2]
 e) ZH[:6:3]

4. Considere BA=[3, 5, 11, 17, 20, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) BA[-4:4:3]
 b) BA[-5:-3:2]
 c) BA[:6]
 d) BA[:4:-3]
 e) BA[:4:-1]

5. Considere SB=[2, 9, 12, 15, 16, 18, 20, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SB[:8:3]
 b) SB[-6:8]
 c) SB[:8:2]
 d) SB[1:-1]
 e) SB[2:6:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69110
GABRIEL MELEK DOS SANTOS 19FFU108 - 7 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 RL=[2, 3, 4, 11, 13, 17, 18, 19]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RL[3:6:2]
 b) RL[-7:7:2]
 c) RL[-6:-3:2]
 d) RL[3:6:3]
 e) RL[1:-3:2]

2. Considere NW=[3, 14, 17, 19, 20, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NW[2:4]
 b) NW[-6:-3]
 c) NW[-6:5:2]
 d) NW[3:4]
 e) NW[:4:2]

3. Considere IY=[3, 4, 7, 10, 12, 13, 16, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IY[1:8:2]
 b) IY[:7:2]
 c) IY[:7:2]
 d) IY[:7]
 e) IY[-6:-1:2]

4. Considere ZT=[2, 15, 21, 23, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZT[1:4]
 b) ZT[:6:3]
 c) ZT[:6:3]
 d) ZT[:5]
 e) ZT[:4]

5. Considere GP=[2, 3, 18, 23, 24, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GP[-7:7:3]
 b) GP[:7:2]
 c) GP[3:7:2]
 d) GP[2:6:2]
 e) GP[-5:-2:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69127
GABRIEL ROBERTO DA ROSA 19FFU108 - 8
apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 HL=[1, 7, 11, 12, 17, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HL[3:6:2]
 b) HL[3:4:2]
 c) HL[:5]
 d) HL[1:4:2]
 e) HL[-5:-2:2]

2. Considere RT=[3, 6, 7, 10, 14, 19, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RT[-7:-1]
 b) RT[1:-2:3]
 c) RT[1:7:2]
 d) RT[-6:7:2]
 e) RT[:7]

3. Considere IR=[1, 6, 8, 10, 18, 22, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IR[:7]
 b) IR[1:6:2]
 c) IR[-7:6:2]
 d) IR[:6:3]
 e) IR[-5:7]

4. Considere MI=[3, 6, 7, 12, 13, 16, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MI[:5:2]
 b) MI[:7:2]
 c) MI[:7]
 d) MI[-5:5]
 e) MI[2:6:2]

5. Considere RZ=[4, 7, 12, 21, 22, 24, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RZ[-6:5]
 b) RZ[1:-1]
 c) RZ[2:6:3]
 d) RZ[3:-3]
 e) RZ[2:-3:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69134
GABRIEL SOUZA BRAGA 19FFU108 - 9 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 YM=[6, 7, 8, 14, 17, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YM[3:5]
 b) YM[:4:3]
 c) YM[-5:-3]
 d) YM[1:6:2]
 e) YM[-6:4]

2. Considere NU=[1, 5, 8, 14, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NU[-6:6]
 b) NU[:6:3]
 c) NU[1:6:2]
 d) NU[1:-3]
 e) NU[-4:6:2]

3. Considere IK=[2, 3, 4, 11, 23, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IK[:5:2]
 b) IK[2:6]
 c) IK[1:5:2]
 d) IK[3:4:2]
 e) IK[2:-1:2]

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

5. Considere HY=[3, 4, 6, 8, 9, 14, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HY[:7]
 b) HY[2:7:3]
 c) HY[1:7:2]
 d) HY[-5:-2:2]
 e) HY[:5:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69141
GUILHERME SAITO BANDEIRA 19FFU108 - 10 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 PQ=[1, 9, 11, 16, 18, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) PQ[-5:5:2]
 b) PQ[3:7]
 c) PQ[:6]
 d) PQ[-7:-3:2]
 e) PQ[3:7:2]

2. Considere CB=[1, 3, 4, 6, 12, 24, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CB[1:7:2]
 b) CB[-7:8:2]
 c) CB[:6:-2]
 d) CB[2:7]
 e) CB[1:6]

3. Considere OH=[4, 6, 11, 17, 19, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[:4:2]
 b) OH[3:6:2]
 c) OH[-6:-1]
 d) OH[-5:-3:3]
 e) OH[:5]

4. Considere UQ=[1, 4, 7, 18, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UQ[2:5]
 b) UQ[:4:-3]
 c) UQ[-6:5:3]
 d) UQ[2:5]
 e) UQ[-6:4:3]

5. Considere TO=[2, 9, 10, 13, 15, 19, 20, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TO[3:-3:2]
 b) TO[:8:3]
 c) TO[3:7:3]
 d) TO[:6:2]
 e) TO[:6:-1]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69158
HARON AURELIANO TRAVASSOS 19FFU108 - 11 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 RG=[4, 7, 10, 13, 17, 18, 22, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RG[3:7:2]
 b) RG[:7]
 c) RG[-7:7:2]
 d) RG[2:6:2]
 e) RG[:6]

2. Considere SC=[3, 5, 11, 19, 20, 25, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SC[:8]
 b) SC[2:-3]
 c) SC[:7:2]
 d) SC[2:7:2]
 e) SC[1:8]

3. Considere DI=[1, 3, 5, 11, 19, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DI[1:6:2]
 b) DI[-4:4:2]
 c) DI[2:-3:2]
 d) DI[2:6:3]
 e) DI[-5:-3:3]

4. Considere UE=[7, 10, 11, 12, 19, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UE[:5:2]
 b) UE[2:-2:2]
 c) UE[-4:5:3]
 d) UE[3:5]
 e) UE[1:5:3]

5. Considere CY=[3, 7, 13, 14, 18, 19, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CY[3:7:2]
 b) CY[-7:5]
 c) CY[:5:2]
 d) CY[:7:3]
 e) CY[:5:-3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69239
ICLEIA SANTOS (OUV) 19FFU108 - 12 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 CI=[2, 6, 13, 17, 26, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CI[:7:2]
 b) CI[2:6:2]
 c) CI[1:5:2]
 d) CI[-7:7:2]
 e) CI[3:-1]

2. Considere CZ=[11, 13, 18, 21, 22, 23, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CZ[2:-2]
 b) CZ[-6:8]
 c) CZ[-7:6]
 d) CZ[:6:-2]
 e) CZ[:6:2]

3. Considere SC=[1, 2, 6, 7, 14, 15, 16, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SC[:7:2]
 b) SC[3:-2]
 c) SC[2:-2]
 d) SC[3:7]
 e) SC[3:-2:3]

4. Considere QN=[5, 6, 23, 26, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QN[1:-2]
 b) QN[:5]
 c) QN[2:5:2]
 d) QN[1:-3:3]
 e) QN[2:-3]

5. Considere LR=[7, 10, 14, 16, 20, 23, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LR[:8:2]
 b) LR[1:7]
 c) LR[3:-2]
 d) LR[3:-1:2]
 e) LR[3:-3:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69246
MAURICIO KOBAY DO AMARAL 19FFU108 - 13 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 NE=[11, 18, 19, 21, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NE[1:5]
 b) NE[-6:5:2]
 c) NE[:7]
 d) NE[2:7:2]
 e) NE[-6:7]

2. Considere SG=[2, 8, 10, 11, 12, 20, 24, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SG[-6:7:2]
 b) SG[:8:3]
 c) SG[-8:7:2]
 d) SG[3:8:2]
 e) SG[-7:8]

3. Considere PZ=[3, 4, 18, 21, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) PZ[-4:4:2]
 b) PZ[:4:-2]
 c) PZ[2:6:2]
 d) PZ[-6:6]
 e) PZ[:5]

4. Considere AV=[1, 6, 15, 16, 26, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AV[3:4:3]
 b) AV[1:5:2]
 c) AV[-5:-3:2]
 d) AV[1:5:3]
 e) AV[3:4]

5. Considere LX=[4, 5, 17, 18, 19, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LX[1:5:3]
 b) LX[3:5]
 c) LX[1:6:2]
 d) LX[2:5:2]
 e) LX[:4:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69253
PEDRO HENNIG 19FFU108 - 14 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 IX=[4, 13, 16, 17, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IX[3:4:3]
 b) IX[:6:2]
 c) IX[1:5]
 d) IX[:4:3]
 e) IX[2:4]

2. Considere HD=[3, 10, 11, 12, 20, 22, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HD[1:-1:2]
 b) HD[-6:5:2]
 c) HD[:5:2]
 d) HD[3:-2:2]
 e) HD[2:6:2]

3. Considere MJ=[8, 16, 19, 22, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MJ[:5]
 b) MJ[3:4]
 c) MJ[-4:6:3]
 d) MJ[2:4:2]
 e) MJ[:5:2]

4. Considere GV=[1, 3, 8, 13, 22, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GV[:5:3]
 b) GV[3:-3:2]
 c) GV[1:7:2]
 d) GV[3:6:2]
 e) GV[-6:6:2]

5. Considere UC=[7, 12, 15, 16, 23, 26, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UC[-7:6:2]
 b) UC[3:5:2]
 c) UC[:5:-2]
 d) UC[-5:-1]
 e) UC[-5:5:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69165
RAFAEL DO NASCIMENTO BARBOSA
19FFU108 - 15 apos 09/04, 50% _____ / / _____

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 operaçao 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
    -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 WE=[5, 7, 12, 14, 16, 19, 20]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) WE[2:7]
 b) WE[3:7:3]
 c) WE[:7:2]
 d) WE[2:6:2]
 e) WE[:7:2]

2. Considere UJ=[10, 14, 15, 20, 25, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UJ[1:-2:2]
 b) UJ[-5:5:3]
 c) UJ[1:7]
 d) UJ[2:6]
 e) UJ[1:7]

3. Considere HI=[3, 7, 9, 11, 15, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HI[-6:-3:2]
 b) HI[-6:6]
 c) HI[1:6]
 d) HI[:5:3]
 e) HI[-7:7:3]

4. Considere IU=[1, 3, 6, 9, 12, 15, 19, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IU[:8:2]
 b) IU[1:-3:3]
 c) IU[3:8:3]
 d) IU[2:8]
 e) IU[-6:8:2]

5. Considere ML=[6, 8, 9, 11, 19, 24, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ML[-6:6:2]
 b) ML[:6:2]
 c) ML[:6:-2]
 d) ML[:8:2]
 e) ML[:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69172
RODRIGO PORTANTIOLO WAGNER
19FFU108 - 16 apos 09/04, 50% _____ / / _____

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 operaçao 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
    -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 HF=[3, 5, 8, 16, 18, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HF[1:7:2]
 b) HF[3:5:2]
 c) HF[3:5]
 d) HF[:5]
 e) HF[:7:2]

2. Considere LV=[2, 3, 9, 12, 16, 17, 18, 20]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LV[:6:-1]
 b) LV[-6:-2:2]
 c) LV[:7:3]
 d) LV[1:6:2]
 e) LV[:7:3]

3. Considere YV=[2, 4, 7, 8, 12, 17, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YV[2:6:2]
 b) YV[1:7]
 c) YV[1:8:3]
 d) YV[1:7:2]
 e) YV[:6]

4. Considere HU=[4, 7, 9, 13, 16, 21, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HU[3:6]
 b) HU[2:7]
 c) HU[3:7]
 d) HU[3:7:2]
 e) HU[:7:2]

5. Considere ZM=[1, 3, 11, 18, 22, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZM[:5:-1]
 b) ZM[3:6:2]
 c) ZM[2:6:2]
 d) ZM[3:-2]
 e) ZM[-7:-1:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 16/03/2019 - 11:20:38.2
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVO930a, V: 1.05 69189
WILSON DE MELO 19FFU108 - 17 apos 09/04, 50% _____ / _____ / _____

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 operaçao 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
    -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 UI=[3, 5, 9, 10, 11, 16, 20, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) UI[:6:2]
 b) UI[:7:2]
 c) UI[1:8:2]
 d) UI[-6:6:2]
 e) UI[3:6:2]

2. Considere XP=[18, 19, 24, 25, 26, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) XP[2:5:2]
 b) XP[3:-2]
 c) XP[2:-2]
 d) XP[2:7]
 e) XP[-5:-3:3]

3. Considere FO=[1, 11, 12, 14, 16, 18, 19, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FO[-7:7]
 b) FO[1:8:2]
 c) FO[:7]
 d) FO[2:6:2]
 e) FO[3:6:3]

4. Considere AV=[1, 2, 5, 7, 8, 9, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AV[1:7:2]
 b) AV[2:7:2]
 c) AV[3:-3:3]
 d) AV[3:6]
 e) AV[2:7:2]

5. Considere RB=[2, 3, 10, 16, 25, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RB[2:5]
 b) RB[:5]
 c) RB[-7:-2]
 d) RB[-5:-2]
 e) RB[-7:-1:3]
```

Responda aqui:

1	2	3	4	5