U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75789
ARIEL GUSTAVO RODRIGUES DA S
24PRG308 - 1 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere WP=[2, 3, 4, 19, 23, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) WP[:5]
 b) WP[:7]
 c) WP[2:7:2]
 d) WP[2:7:2]
 e) WP[1:5:2]

2. Considere HR=[4, 8, 11, 14, 15, 19, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HR[3:-1:2]
 b) HR[3:-1:2]
 c) HR[2:7:2]
 d) HR[2:-3:2]
 e) HR[3:7:3]

3. Considere DV=[4, 5, 7, 17, 18, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DV[-6:6:3]
 b) DV[:5:3]
 c) DV[-5:6]
 d) DV[3:4:2]
 e) DV[:4:2]

4. Considere DN=[2, 8, 14, 15, 22, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DN[3:6:2]
 b) DN[3:-1:2]
 c) DN[1:5]
 d) DN[3:-1:3]
 e) DN[1:-1:2]

5. Considere IN=[2, 17, 18, 20, 22, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IN[-4:-1]
 b) IN[:4:-1]
 c) IN[3:5:2]
 d) IN[3:6]
 e) IN[:6:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75796
CASSIANO MAGNO CHAGAS E SA 24PRG308 - 2 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere JY=[14, 16, 17, 21, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JY[1:4]
 b) JY[1:4:2]
 c) JY[2:6:2]
 d) JY[3:5:2]
 e) JY[:6:3]

2. Considere IP=[1, 5, 9, 13, 19, 20, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IP[2:7:2]
 b) IP[:7]
 c) IP[2:-3]
 d) IP[3:-2]
 e) IP[:5:3]

3. Considere SL=[7, 9, 10, 12, 13, 24, 26, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SL[3:-3]
 b) SL[3:7:2]
 c) SL[-6:6:2]
 d) SL[:8]
 e) SL[1:7:2]

4. Considere OQ=[5, 9, 15, 23, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OQ[3:6:2]
 b) OQ[1:-3:2]
 c) OQ[3:5]
 d) OQ[1:6]
 e) OQ[-6:5:2]

5. Considere MS=[1, 4, 6, 17, 19, 20]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) MS[-5:4]
 b) MS[-6:-3]
 c) MS[-5:6]
 d) MS[3:6:2]
 e) MS[-4:-3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75808
CHRISTOPHER BUCH FILIPAK 24PRG308 - 3 / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere NC=[2, 15, 17, 18, 20, 23, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NC[:6:2]
 b) NC[-7:-1:2]
 c) NC[:6]
 d) NC[:6]
 e) NC[2:8:3]

2. Considere YC=[3, 7, 9, 12, 14, 15, 21, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YC[3:-2]
 b) YC[:6]
 c) YC[:6:2]
 d) YC[3:8]
 e) YC[2:6:2]

3. Considere FO=[1, 2, 3, 5, 6, 9, 24, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FO[2:-3:3]
 b) FO[1:6:3]
 c) FO[:6:2]
 d) FO[2:7]
 e) FO[1:8]

4. Considere NW=[1, 4, 12, 14, 15, 17, 23, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NW[:7:2]
 b) NW[1:6:2]
 c) NW[2:8]
 d) NW[1:6]
 e) NW[3:-3]

5. Considere AE=[4, 18, 21, 22, 23, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AE[:5]
 b) AE[2:-1]
 c) AE[2:6:2]
 d) AE[:6]
 e) AE[:5:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75815
ERAN MARTINEZ RAMOS 24PRG308 - 4
/ _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere KT=[8, 9, 20, 21, 22, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KT[2:5:3]
 b) KT[3:4]
 c) KT[3:5:3]
 d) KT[:6:3]
 e) KT[:4:-3]

2. Considere OH=[2, 8, 13, 19, 20, 23, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OH[:6:3]
 b) OH[2:-2:2]
 c) OH[:5:2]
 d) OH[1:-3:3]
 e) OH[-6:-1:2]

3. Considere RO=[1, 5, 14, 15, 19, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RO[-5:5:2]
 b) RO[2:-3:2]
 c) RO[2:-1:2]
 d) RO[2:5:3]
 e) RO[3:-1:2]

4. Considere EC=[2, 9, 14, 15, 16, 21, 22, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EC[2:7]
 b) EC[1:-1:2]
 c) EC[2:-3:2]
 d) EC[1:-2]
 e) EC[:6:-3]

5. Considere PH=[2, 3, 9, 13, 14, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) PH[:4:-2]
 b) PH[2:-1]
 c) PH[3:-2]
 d) PH[-6:6]
 e) PH[1:4:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75822
HENRIQUE IVANKIO GUSSE 24PRG308 - 5 / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere SV=[4, 5, 6, 7, 23, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SV[1:4]
 b) SV[3:4:2]
 c) SV[1:6:2]
 d) SV[3:5]
 e) SV[:6:2]

2. Considere LZ=[2, 6, 14, 17, 20, 23, 24, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LZ[:7:2]
 b) LZ[:6:-1]
 c) LZ[:8]
 d) LZ[-7:-1]
 e) LZ[-8:-1]

3. Considere ZQ=[2, 3, 9, 13, 17, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZQ[2:6:2]
 b) ZQ[-5:4:3]
 c) ZQ[3:4]
 d) ZQ[:4:-2]
 e) ZQ[3:-1]

4. Considere SE=[4, 5, 10, 11, 12, 16, 17, 19]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SE[2:6:2]
 b) SE[2:8:2]
 c) SE[3:6]
 d) SE[1:-3]
 e) SE[-7:6]

5. Considere EP=[1, 3, 7, 11, 15, 17, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EP[3:5:2]
 b) EP[2:-2:2]
 c) EP[1:7:2]
 d) EP[:6:2]
 e) EP[1:-2:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75839 IAN DOLABELLA OHATA 24PRG308 - 6 _____ / / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere SF=[1, 2, 5, 6, 8, 19, 24, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SF[3:7:3]
 b) SF[2:8:3]
 c) SF[1:-2:2]
 d) SF[-7:6:2]
 e) SF[1:8:2]

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

3. Considere AU=[5, 9, 15, 21, 25, 27, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AU[1:8:2]
 b) AU[1:-3]
 c) AU[:6:-2]
 d) AU[3:8]
 e) AU[:6:-2]

4. Considere LG=[1, 2, 3, 15, 17, 21, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LG[-7:-1:2]
 b) LG[-6:7:2]
 c) LG[3:8]
 d) LG[1:6:2]
 e) LG[1:7:3]

5. Considere ZO=[3, 7, 10, 19, 22, 26, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZO[1:-3]
 b) ZO[1:6:3]
 c) ZO[3:-1:3]
 d) ZO[1:5:2]
 e) ZO[-7:-2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75846
JOAO BONFIN LINO 24PRG308 - 7 _____ / / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere OR=[1, 5, 6, 11, 18, 24, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OR[-5:7]
 b) OR[-6:5:2]
 c) OR[-6:7:2]
 d) OR[-5:5:2]
 e) OR[3:-3]

2. Considere NK=[4, 9, 17, 18, 19, 21, 22, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NK[3:6:2]
 b) NK[-7:-1]
 c) NK[-6:-2:3]
 d) NK[-6:-1:2]
 e) NK[:7]

3. Considere JE=[3, 9, 12, 16, 19, 20, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JE[-7:7:3]
 b) JE[-6:5:2]
 c) JE[1:5:2]
 d) JE[3:6:2]
 e) JE[-7:7:2]

4. Considere AO=[1, 8, 13, 20, 23, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) AO[-7:6:3]
 b) AO[3:7:3]
 c) AO[:6:2]
 d) AO[1:6]
 e) AO[:6:2]

5. Considere QW=[4, 9, 20, 22, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QW[:4]
 b) QW[1:6:2]
 c) QW[1:5:2]
 d) QW[2:-2]
 e) QW[3:6]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75853
JOAO LUCA FERNANDES DE OLIVE
24PRG308 - 8 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere JZ=[3, 9, 10, 11, 18, 21, 22, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JZ[-6:-1:3]
 b) JZ[-7:6:2]
 c) JZ[-6:-1:2]
 d) JZ[-8:7:2]
 e) JZ[3:-2:2]

2. Considere KL=[3, 4, 11, 13, 15, 16, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KL[:6:3]
 b) KL[3:5:3]
 c) KL[:5:2]
 d) KL[-7:7:2]
 e) KL[3:6:2]

3. Considere OS=[3, 5, 8, 9, 17, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OS[-7:6:3]
 b) OS[:5]
 c) OS[1:5]
 d) OS[3:-3:2]
 e) OS[:6:2]

4. Considere RL=[12, 15, 19, 21, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RL[-6:4]
 b) RL[:5:3]
 c) RL[3:-2:2]
 d) RL[-5:4:2]
 e) RL[-6:-2]

5. Considere ZD=[2, 8, 12, 18, 21, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZD[-4:-3:3]
 b) ZD[3:-1]
 c) ZD[2:-3:2]
 d) ZD[-4:6:2]
 e) ZD[:6]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75860
JOAO PEDRO DE BRITO DUARTE 24PRG308 - 9 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere ZJ=[3, 10, 15, 17, 21, 22, 25, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) ZJ[:8:2]
 b) ZJ[:7:3]
 c) ZJ[:8:2]
 d) ZJ[-8:8:2]
 e) ZJ[:6]

2. Considere BG=[3, 14, 18, 22, 26, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) BG[1:5:2]
 b) BG[2:6:2]
 c) BG[:4:3]
 d) BG[3:5:2]
 e) BG[:5:2]

3. Considere RJ=[1, 2, 11, 12, 20, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RJ[1:6:2]
 b) RJ[-5:4:3]
 c) RJ[:6]
 d) RJ[:4]
 e) RJ[-4:6:3]

4. Considere EJ=[5, 6, 10, 11, 13, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EJ[-7:8]
 b) EJ[1:7]
 c) EJ[:7]
 d) EJ[:6:2]
 e) EJ[2:6:2]

5. Considere LP=[12, 13, 14, 18, 20, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LP[3:-2:2]
 b) LP[-6:4:2]
 c) LP[-4:4:2]
 d) LP[-6:-3]
 e) LP[:5:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75877
JOAO PEDRO MARIANO SUEKE 24PRG308 - 10 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere LJ=[3, 4, 14, 16, 18, 19, 20]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LJ[3:5]
 b) LJ[3:6:2]
 c) LJ[2:6]
 d) LJ[1:7]
 e) LJ[-5:-2:2]

2. Considere SY=[3, 5, 16, 21, 25, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SY[1:5]
 b) SY[1:5]
 c) SY[2:-1:2]
 d) SY[-7:7:2]
 e) SY[:5]

3. Considere OT=[1, 6, 16, 17, 18, 21, 26, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OT[:7]
 b) OT[:7:2]
 c) OT[:7]
 d) OT[-6:-2]
 e) OT[:7]

4. Considere QB=[1, 7, 9, 11, 14, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QB[3:5:3]
 b) QB[:6:3]
 c) QB[3:5:2]
 d) QB[:6:2]
 e) QB[:4:3]

5. Considere HZ=[3, 6, 7, 8, 20, 23, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HZ[-6:8:2]
 b) HZ[-7:8:2]
 c) HZ[-6:7]
 d) HZ[:7:2]
 e) HZ[-6:-1:3]
```

Responda aqui:

1	2	3	4	5

308-75877 - /

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75884
JOAO VICTOR GOMES VILELA GER
24PRG308 - 11 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere QZ=[1, 9, 12, 21, 22, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QZ[-4:-3]
 b) QZ[:4:-3]
 c) QZ[:5:2]
 d) QZ[:5:2]
 e) QZ[-4:-1]

2. Considere KR=[3, 8, 10, 18, 19, 22, 25, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KR[2:8]
 b) KR[-6:6:3]
 c) KR[:8:2]
 d) KR[-6:7]
 e) KR[:6:2]

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

4. Considere KW=[1, 15, 17, 19, 20, 23, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KW[2:7:3]
 b) KW[1:7]
 c) KW[2:-1:2]
 d) KW[-6:6:2]
 e) KW[2:7]

5. Considere CI=[5, 8, 11, 15, 20, 25, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CI[:6:2]
 b) CI[:7:2]
 c) CI[:7:3]
 d) CI[:8:2]
 e) CI[:7]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75989
LUCAS ANTOSZCZYSZEN 24PRG308 - 12 / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere FL=[2, 4, 11, 14, 18, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FL[-6:6]
 b) FL[-5:6]
 c) FL[-7:7:2]
 d) FL[2:-1:2]
 e) FL[:5:-2]

2. Considere KL=[2, 13, 16, 18, 20, 24, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KL[2:-2]
 b) KL[:6:-2]
 c) KL[3:8:2]
 d) KL[:6:-2]
 e) KL[-8:8]

3. Considere LW=[8, 9, 12, 13, 14, 23, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LW[-7:8:2]
 b) LW[-6:-2:2]
 c) LW[-6:8:2]
 d) LW[3:6:2]
 e) LW[:7:2]

4. Considere EY=[3, 4, 8, 12, 17, 18, 25, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) EY[3:-1:3]
 b) EY[1:7:2]
 c) EY[1:-1:3]
 d) EY[-6:-1:2]
 e) EY[:6:2]

5. Considere QM=[3, 8, 9, 21, 23, 26, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QM[1:6:2]
 b) QM[-7:6:2]
 c) QM[:7]
 d) QM[:6:-2]
 e) QM[-6:-3:3]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75891 LUIZ FELIPE GUEDES BUCCHERI 24PRG308 - 13 / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere VP=[4, 6, 8, 16, 23, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VP[-6:5:2]
 b) VP[3:5:2]
 c) VP[:4]
 d) VP[:4:2]
 e) VP[:4:3]

2. Considere CT=[1, 8, 12, 14, 16, 18, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CT[1:5:3]
 b) CT[:5:-1]
 c) CT[-5:5]
 d) CT[2:5:2]
 e) CT[:7]

3. Considere VS=[3, 8, 9, 18, 20, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VS[2:-2]
 b) VS[2:-3:2]
 c) VS[:4]
 d) VS[3:-2:3]
 e) VS[:5:3]

4. Considere OU=[1, 2, 5, 8, 11, 15, 19, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OU[3:-3:2]
 b) OU[1:-2]
 c) OU[3:7]
 d) OU[-8:8]
 e) OU[3:-1:2]

5. Considere GS=[5, 9, 10, 11, 12, 15, 16, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GS[3:6]
 b) GS[3:8]
 c) GS[-6:-1:2]
 d) GS[2:8:2]
 e) GS[:7]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75903
MANUELLA LEAL DE MEIRELLES F
24PRG308 - 14 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere JA=[1, 3, 7, 8, 11, 20, 22, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JA[:8]
 b) JA[1:-2:3]
 c) JA[2:6]
 d) JA[-6:6:3]
 e) JA[:6:3]

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

3. Considere FR=[2, 3, 6, 14, 24, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FR[1:7]
 b) FR[2:7]
 c) FR[-5:-1:2]
 d) FR[:5:2]
 e) FR[1:6:2]

4. Considere YT=[8, 9, 11, 17, 19, 20, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) YT[2:-1:3]
 b) YT[-7:-2:3]
 c) YT[-5:-3]
 d) YT[-7:-2:3]
 e) YT[:5:2]

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

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75910
MARIA EDUARDA DA SILVA 24PRG308 - 15 / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere SD=[1, 4, 5, 15, 20, 21, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SD[:6:2]
 b) SD[1:8]
 c) SD[:8]
 d) SD[2:-2:2]
 e) SD[:6:-2]

2. Considere OR=[7, 12, 14, 16, 17, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OR[1:6:2]
 b) OR[:4:-3]
 c) OR[:4:2]
 d) OR[3:4]
 e) OR[2:6:3]

3. Considere NQ=[6, 7, 12, 13, 16, 18, 22, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NQ[:8:2]
 b) NQ[1:-3]
 c) NQ[1:7:2]
 d) NQ[1:-2:2]
 e) NQ[1:-2:3]

4. Considere DW=[9, 14, 17, 18, 27, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DW[-4:6:3]
 b) DW[:4]
 c) DW[2:6:2]
 d) DW[-4:4:2]
 e) DW[1:5:2]

5. Considere CP=[4, 5, 7, 9, 11, 21, 22, 23]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) CP[3:7:3]
 b) CP[:6:3]
 c) CP[-8:6]
 d) CP[:7:2]
 e) CP[3:7:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75927
MATHEUS AITA FABRICIO DE CAR
24PRG308 - 16 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere NT=[6, 10, 14, 18, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NT[3:-1:2]
 b) NT[3:6:3]
 c) NT[1:4:2]
 d) NT[-4:-1:2]
 e) NT[:5:2]

2. Considere SW=[2, 6, 12, 13, 20, 23, 24, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) SW[2:8:3]
 b) SW[:7]
 c) SW[-8:8:2]
 d) SW[:8:2]
 e) SW[3:-3:2]

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

4. Considere VS=[8, 12, 13, 14, 16, 18, 21, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) VS[1:8:2]
 b) VS[-8:7:2]
 c) VS[-6:8:2]
 d) VS[1:7:2]
 e) VS[:8]

5. Considere IF=[2, 12, 13, 14, 17, 19, 20]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) IF[2:7:2]
 b) IF[2:-1:2]
 c) IF[-5:5:2]
 d) IF[:5:3]
 e) IF[1:7]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75934
MATHEUS LEANDRO DE BITENCOUR
24PRG308 - 17 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere GW=[6, 8, 10, 13, 15, 18, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) GW[-5:6:2]
 b) GW[1:6:2]
 c) GW[2:7:3]
 d) GW[:7:2]
 e) GW[-7:7:2]

2. Considere QR=[1, 2, 3, 14, 18, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QR[-4:-1]
 b) QR[2:6]
 c) QR[1:5:2]
 d) QR[:6]
 e) QR[-5:-3:2]

3. Considere FR=[1, 2, 12, 17, 18, 19, 22]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FR[-6:7:3]
 b) FR[-6:6:2]
 c) FR[3:5]
 d) FR[1:-1:2]
 e) FR[1:6:2]

4. Considere JF=[3, 10, 12, 16, 19, 23, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JF[2:7:2]
 b) JF[:6]
 c) JF[3:6:2]
 d) JF[-8:-1]
 e) JF[1:7:2]

5. Considere PY=[10, 12, 20, 21, 27, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) PY[:5:2]
 b) PY[1:-3:3]
 c) PY[3:4:2]
 d) PY[:6]
 e) PY[:6]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75941
MAYCON TAVARES WOLPE 24PRG308 - 18 / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere FL=[4, 5, 7, 10, 22, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FL[:4:2]
 b) FL[1:4:2]
 c) FL[:6]
 d) FL[-5:5:2]
 e) FL[2:-1:2]

2. Considere DG=[4, 7, 8, 10, 11, 15, 18]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DG[3:7:3]
 b) DG[1:-1]
 c) DG[2:6]
 d) DG[1:6]
 e) DG[1:5:3]

3. Considere TM=[1, 2, 10, 16, 17, 20, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) TM[-7:7]
 b) TM[-8:-1:2]
 c) TM[2:6:2]
 d) TM[1:-2:2]
 e) TM[3:7:2]

4. Considere FY=[1, 10, 13, 19, 21, 23, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) FY[2:7]
 b) FY[3:6]
 c) FY[:6:3]
 d) FY[1:6]
 e) FY[1:-2]

5. Considere HS=[4, 5, 6, 8, 15, 18, 21]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HS[-7:6:2]
 b) HS[2:-1:2]
 c) HS[:7]
 d) HS[3:-3]
 e) HS[-5:-2:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75958
PEDRO TOLEDO LEAL 24PRG308 - 19
/ _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere KV=[13, 14, 23, 25, 26, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) KV[:5:2]
 b) KV[-4:6:2]
 c) KV[:6:2]
 d) KV[-4:-3:3]
 e) KV[:4:2]

2. Considere HP=[3, 5, 7, 9, 11, 21, 22, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) HP[3:7]
 b) HP[3:-3:2]
 c) HP[:6:2]
 d) HP[1:-3:2]
 e) HP[2:6:2]

3. Considere BX=[11, 14, 17, 20, 21, 25, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) BX[:5:3]
 b) BX[:5:-1]
 c) BX[2:7]
 d) BX[:7:2]
 e) BX[1:-1]

4. Considere JK=[5, 12, 13, 14, 20, 23, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) JK[:7]
 b) JK[1:7:2]
 c) JK[2:7]
 d) JK[:5:2]
 e) JK[1:5:2]

5. Considere RV=[2, 5, 15, 16, 19, 20, 21, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) RV[-7:-2:2]
 b) RV[2:-1:3]
 c) RV[1:7:2]
 d) RV[-6:8:2]
 e) RV[2:8:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75965
PHELIPE GABRIEL LIMA DA SILV 24PRG308 - 20 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere LO=[4, 7, 9, 11, 20, 22, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LO[:7:2]
 b) LO[1:6]
 c) LO[1:6:2]
 d) LO[:7:2]
 e) LO[2:-2]

2. Considere OV=[4, 6, 10, 14, 22, 23, 24]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) OV[-7:5:2]
 b) OV[2:7]
 c) OV[-7:6]
 d) OV[1:7:2]
 e) OV[-5:5:2]

3. Considere LJ=[1, 2, 9, 15, 18, 22, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) LJ[1:7:3]
 b) LJ[2:5:2]
 c) LJ[3:5:2]
 d) LJ[-6:7:2]
 e) LJ[1:5:2]

4. Considere DJ=[3, 4, 10, 11, 12, 16, 20, 28]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DJ[-7:6]
 b) DJ[-7:8]
 c) DJ[2:-2]
 d) DJ[:6]
 e) DJ[1:-3:3]

5. Considere BR=[7, 9, 13, 15, 16, 18, 24, 26]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) BR[2:8:2]
 b) BR[1:6]
 c) BR[3:7]
 d) BR[-7:-3:3]
 e) BR[-8:8:2]
```

Responda aqui:

1	2	3	4	5

U Positivo - UTFPR - PUCPr – 29/07/2024 - 09:03:28.7
Prof Dr P Kantek (pkantek@gmail.com)
Listas em Python VIVX930a, V: 1.05 75972
REBECA CABRAL DOS SANTOS 24PRG308 - 21 _____ / _____ / _____

Listas em Python

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

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

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

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

Indexação

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

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

Acompanhe o que se disse no exemplo:

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

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

Fatiamento

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

```
inicio:final:incremento
```

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

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

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

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

Outras operações

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

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

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

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

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

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

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

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

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

Exemplo

A seguir uma instância feita e correta para os exercícios.

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

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

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

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

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

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

☞ Para você fazer

```
1. Considere BF=[3, 8, 10, 18, 19, 24, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) BF[-5:-3]
 b) BF[1:6:2]
 c) BF[-6:5:3]
 d) BF[:6]
 e) BF[1:7:2]

2. Considere NT=[1, 2, 3, 6, 11, 23, 24, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) NT[-6:7]
 b) NT[-6:6]
 c) NT[-7:8:2]
 d) NT[:8:3]
 e) NT[2:7:3]

3. Considere PD=[8, 10, 15, 19, 20, 24, 27]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) PD[3:7]
 b) PD[3:6:3]
 c) PD[:5]
 d) PD[:5:2]
 e) PD[2:5]

4. Considere DZ=[12, 13, 16, 17, 19, 28, 29]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) DZ[-6:-1:2]
 b) DZ[3:5:2]
 c) DZ[-5:7]
 d) DZ[:5:-3]
 e) DZ[1:5:2]

5. Considere QK=[4, 6, 11, 16, 18, 19, 25]
Efetue todos os fatiamentos.
Ao final, some todos os elementos.
 a) QK[2:5:3]
 b) QK[:6:2]
 c) QK[3:5:2]
 d) QK[1:7:2]
 e) QK[3:-1:2]
```

Responda aqui:

1	2	3	4	5

308-75972 - /