UTFPR - Universidade Tecnológica Federal do Paraná
DACOM - Departamento de Computação
BCC1001 :: Introdução à Programação

Lista de Exercícios no. 6 :: Listas

Instruções Gerais

- Os exercícios são de resolução individual.
- Sempre tente resolver os exercícios por conta própria, mesmo aqueles que já tenham sido feitos pelo professor em aula.
- Crie um arquivo (ex: lista.py) e faça cada exercício em uma ou mais funções.
- Utilize a extensão .py e o editor VS Code (ou outro de sua preferência).
- **Não é permitido o uso de recursos da linguagem ou bibliotecas que não foram abordados na disciplina até o momento da publicação desta lista.**
    - A relação de operadores e funções de listas que podem ser utilizadas está em anexo. Note que os exercícios podem adicionar restrições sobre o uso de recursos dessa relação.

1. Escreva uma função que recebe uma lista **vet** a imprime em ordem reversa. Não é permitido o uso de list.reverse().

   ```
   def print_reverse(vet: list)
   ```

2. Escreva uma função que recebe uma lista **vet** e imprime apenas os valores pares.

   ```
   def print_even(vet: list)
   ```

3. Escreva uma função que recebe uma lista **vet** contendo números inteiros. A função deve modificar a lista, invertendo o sinal dos números negativos, passando-os para positivo.

   ```
   def set_positive(vet: list)
   ```

   Ex: Entrada:[1, -5, 67, -45, -1, -1, 0, 48] → Saída:[1, 5, 67, 45, 1, 1, 0, 48]

4. Escreva uma função que recebe uma lista **vet** e devolve a média aritmética simples dos elementos.

```
def list_sum(vet: list) -> float
```

   Ex: Entrada: [1, 23, 4, 8, 41, 7, 3] → Saída: 12

5. Escreva uma função que recebe uma lista **vet**, bem como, um elemento **elem** a ser procurado. A função deve retornar a posição (índice) do elemento ou **None** caso ele não esteja no vetor.

   ```
   def find(vet: list, elem: int)
   ```

6. Escreva uma função que recebe uma lista **vet** ordenado crescentemente, bem como, um elemento **elem** a ser procurado. A função deve retornar a posição (índice) do elemento ou **None** caso ele não esteja na lista. **OBS: Tente usar o fato da lista estar ordenada para criar uma solução melhor que a anterior**

   ```
   def find(vet: list, elem: int)
   ```

7. Escreva uma função que recebe uma lista **vet** e um número **value**. A função deve retornar uma outra lista, contendo os múltiplos de **value** que estão em **vet**.

   def get_multiples(vet: list, value: int) -> list

8. Escreva uma função que recebe uma lista **vet** e retorna uma outra lista, com os primos em vet.

   def get_primes(vet: list) -> list

9. Escreva uma função que verifica se os elementos de uma lista estão em ordem crescente. A função deve retornar True, caso os elementos estejam dispostos em ordem crescente, ou False, em caso contrário. Obs: Não é permitido o uso de list.sort().

   def is_sorted(vet: list) -> bool

   ```
   Exemplo de uso da função: v = [1,4,7,9,15,22,48,512]
                                       print(is_sorted(v)) # True
   ```

10. Escreva uma função que recebe uma lista **vet** e devolve o segundo maior elemento. Dica: list.sort()

    def find_max2(vet: list) -> int

11. Escreva uma função que recebe uma lista **vet** e devolve o maior e o menor elementos. Obs: Não é permitido o uso de mint() e max()

    def find_min_max(vet: list) -> tuple[int, int]

12. Escreva uma função que recebe uma lista **vet** e devolve o maior e o menor elementos. Obs: Não é permitido o uso de mint(), max() e list.sort()

    def find_min_max(vet: list) -> tuple[int, int]

13. Escreva uma função que recebe uma lista **vet** e devolve seus 3 maiores elementos. Obs: Não é permitido o uso de max() e list.sort()

    def get_max3(vet: list) -> tuple[int, int, int]

14. Escreva uma função que recebe uma lista **vet** e um elemento **elem**. A função deve remover todas as ocorrências de **elem** de **vet**. Dica: list.remove().

    def remove_all(vet: list, elem: int)

15. Escreva uma função que recebe uma lista **vet** e um elemento **elem**. A função deve remover todas as ocorrências de **elem** de **vet**. Obs: Não é permitido o uso de list.index() e list.remove()

    def remove_all(vet: list, elem: int)

16. Escreva uma função que recebe uma lista **vet** e inverte os seus elementos na própria lista. Obs: Não é permitido o uso de list.reverse().

    def reverse_list(vet: list)

17. Escreva uma função que recebe uma lista **vet** preenchido com inteiros positivos. A função deve imprimir as ocorrências (contagem) de cada número na lista. Dica: utilize uma lista **count** para armazenar a contagem de cada elemento na lista **vet**, relacionando as posições de **count** aos valores em **vet**. A lista count precisa ter tamanho max(vet)+1.

    def count_elements(vet: list)

18. Escreva uma função que recebe um valor inteiro **value** e uma lista **notes** com as possíveis cédulas. A função deve imprimir a quantidade mínima de cédulas equivalente ao valor. Dica: use uma lista auxiliar **count** para armazenar a contagem de cada cédula.

    def min_bills(value: int, notes: list)

    ```
    Exemplo: min_bills(209, [1,5,10,50,100]) # lista de tipos de cédulas #
                Saída:
                # 2 x $100
                # 1 x $5
                # 4 x $1
    ```

19. Escreva uma função que recebe uma lista **days** de tamanho 7 que exibe um histograma da variação da temperatura durante a semana. A lista **days** armazena as temperaturas para cada dia da semana, iniciando por **domingo**. Por exemplo, se as temperaturas em **days** forem [19, 21, 25, 22, 20, 17 e 15] graus celsius, a função deverá exibir:

    ```
    D: >>>>>>>>>>>>>>>>>>>
    S: >>>>>>>>>>>>>>>>>>>>>
    T: >>>>>>>>>>>>>>>>>>>>>>>>
    Q: >>>>>>>>>>>>>>>>>>>>>
    Q: >>>>>>>>>>>>>>>>>>>
    S: >>>>>>>>>>>>>>>>>
    S: >>>>>>>>>>>>>>>>
    ```

    def histogram(days: list)

20. Chamando a função desenvolvida no exercício anterior, escreva uma função que recebe a lista **days** e exibe:
    a) As temperaturas média, mínima, máxima;
    b) O histograma.

    def temperature_report(days: list)