Vetores

* Um vetor é uma variável com múltiplas posições dispostas **sequencialmente** na memória.
* O fato delas serem dispostas sequencialmente é importante, porque possibilita o acesso **direto** a qualquer uma das posições.
* Dessa forma, tem-se um conjunto de informações acessíveis através de um **único nome de variável.**



* Vetores são úteis quando é necessário guardar várias informações e não se deseja criar uma variável para cada informação.
  + Até porque isso impossibilitaria a manipulação das informações com o uso de laços.
* Vetores em Python são normalmente implementados usando uma estrutura chamada **lista**.
* Sintaxe:

*vetor = []*

*vetor2 = [0]*

*vetor3 = [0] \* 10*

* Uma lista pode conter informações de diversos tipos misturados.

*vetor = [5, ‘Ricardo’, 3.14]*

* + O tipo de dados lista permite isso, embora vetores tenham como característica armazenar informações do mesmo tipo.
* O acesso a qualquer uma das posições é feito através dos **índices** das posições.
  + O índice de uma posição corresponde a quantas posições ela está distante da posição inicial do vetor.
  + Por isso, a posição inicial do vetor possui índice zero.
  + Não se pode acessar uma posição não reservada do vetor. Isso dispara um erro.
  + Sintaxe:

*vetor[posição]*

* Exemplo:
  + Solicite a idade de um grupo de 10 pessoas. Calcule a média de idade do grupo e informe quantas pessoas têm idade acima e abaixo da média.

Manipulação de listas

* Listas podem ser manipuladas com o uso de um conjunto de funções e operadores predefinidos.

1. len(nome\_da\_lista)
   * mostra quantos elementos há na lista / tamanho da lista.
2. +
   * concatena duas listas, formando uma única lista que contém os elementos das duas listas operadas, mantendo-se a ordem.
   * ex: resultado = lista1 + lista2
3. in
   * retorna um booleano (True / False) que indica se o elemento verificado está ou não presente na lista.
   * ex: 10 in lista1
4. min(nome\_da\_lista) e max(nome\_da\_lista)
   * retorna o menor ou o maior elemento presente na lista.
   * só funciona se a lista for completamente preenchida por valores numéricos.
5. sum(nome\_da\_lista)
   * retorna a soma de todos os valores contidos na lista.
   * só funciona se a lista for completamente preenchida por valores numéricos.

* Existe também uma série de métodos para manipulação de listas.
  + O que são métodos?
    - Outro tipo de função de manipulação.
    - Isso fará sentido quando estudarem orientação a objetos (Programação II).
  + Sintaxe:

*nome\_da\_lista.nome\_do\_metodo(parametros)*

1. append(elemento)
   * acrescenta um elemento no final da lista.
   * o elemento deve ser passado como parâmetro.
   * ex: lista.append(22)
2. insert(posição, elemento)
   * insere um novo elemento em uma posição específica da lista.
   * os demais elementos a partir da posição onde se inseriu o novo elemento são deslocados uma posição pra frente.
3. pop()
   * remove o último elemento da lista.
   * o conteúdo removido é retornado (pode ser atribuído a uma variável, entre outras coisas).
4. pop(indice)
   * remove na posição especificada, ao invés da última posição.
5. remove(item)
   * remove a primeira ocorrência do valor fornecido.
   * só funciona se o valor cuja remoção foi solicitada esteja presente na lista.
6. sort()
   * ordena os elementos da lista em ordem crescente.
7. reverse()
   * inverte a ordem dos elementos da lista.
8. count(elemento)
   * conta a quantidade de ocorrências do elemento especificado.

Exemplo:

Faça um programa que leia números inteiros para uma lista. O usuário irá informar esses números até que informe zero (que não deve ser inserido na lista).

Imprima a lista e exclua todas as ocorrências de um valor informado pelo usuário.