Listas e Arrays

- O conceito de lista (ou seqüência) se refere a um conjunto de valores, normalmente de um *mesmo tipo*, sobre o qual existe a noção de *ordem*.
    - Conceitualmente as listas podem sofrer modificações, inclusive a inserção e remoção de elementos.
        - Ou seja, o tamanho da lista também pode mudar.

Arrays

- São variáveis **compostas homogêneas**:
    - Variáveis de um **mesmo tipo e com um nome único**.
    - Para se referir a cada um dos componentes usa-se um ou mais **índices**.

Tipos de arrays:

- Unidimensionais – vetores.
- Bidimensionais – matrizes.
- Multidimensionais.

- Em Python, ao contrário das outras linguagens:
    - Listas são implementadas nativamente – tipo list.
        - Inclusive com tamanho variável.
    - Não é fornecido o tipo array.

Tipo *list*

- Os elementos podem ser heterogêneos, ou seja, o tipo dos elementos não precisa ser único. Porém, isto não tem muita utilidade na prática (no mundo real)...

> 💡 OBS: Minha recomendação é usá-lo com elementos de um mesmo tipo.

Sintaxe

- Listas são **delimitadas por colchetes ([e])** e os elementos são **separados por vírgulas**.
- O tipo *list* é um tipo como outro qualquer e pode ser usado sem definição prévia.
- Listas também são implementadas como objetos.

```
lista1 = [10,20,30,40]
lista2 = [‘a’, ‘b’, ‘c’]
lista3 = [ ] # Lista vazia (sem elementos).
lista4 = [None, None] # Lista com elementos nulos.
```

Acesso usando índices

- Como nos strings, cada um dos elementos de uma lista pode ser acessado por um índice posicional, com as mesmas regras:
  - Os índices começam a ser contados em **zero** e são escritos entre **colchetes**.
  - **Índices negativos** a partir de -1 podem ser usados para acessá-los de **traz para a frente**.
  - Usar um índice fora do intervalo válido de acordo com o tamanho da lista causa um erro fatal! – IndexError.
- Porque, diferentemente dos strings, as listas podem ser modificadas, não há restrições à alteração do valor de um elemento da lista usando o comando de atribuição e um índice.

```
lista = [10, 20, 30, 40]
prim = lista [0] # Resultado é 10.
ult = lista[-1] # Resultado é 40.
penult = lista [-2] # Resultado é 30.
lista [1] = 50 # Resultado é [10, 50, 30, 40].
```

Operadores e funções

- len: tamanho (quantidade de elementos) da lista.
- +: concatenação de listas (para juntar/emendar...).
- *: repetição de elementos – concatenação repetida...
    - Também é aceito para repetição de listas mas não é recomendado porque resultado não é o que parece...
- in: indica se um elemento está contido em uma lista.
- del: deleta elemento pelo índice, diminuindo a lista.
- min: menor elemento da lista.
- max: maior elemento da lista.
- sum: soma dos elementos da lista.

```
lista = [10, 20, 30, 40]
tam = len (lista) # Resultado é 4.
lis2 = lista + [50] # Resultado é [10, 20, 30, 40, 50].
lis3 = [0] * 50 # Resultado é [0, 0, 0, ..., 0].
res = 30 in lista # res ficará com True.
res = [30] in lista # res ficará com False.
del lis2 [3] # Resultado é [10, 20, 30, 50].
men = min (lista) # Resultado é 10.
mai = max (lista) # Resultado é 40.
total = sum (lista) # Resultado é 100.
```

Comando for

- O comando *for* pode ser usado para acessar os elementos de uma lista “um a um”
  sem precisar usar índices
    - Mas também é possível acessar e/ou alterar os elementos de uma lista usando o comando for mais tradicional, percorrendo pelos índices!

```
# Sintaxe do comando for para listas (sem usar índices).
for elem in qualquerLista :
```

```
    comandoUsandoElem

lista = [10, 20, 30, 40]
for e in lista : # e recebe cada elemento de lista.
    print e # e assume 10, 20, 30 e finalmente 40.

lista = [10, 20, 30, 40]
qtd = len (lista) # Se o valor não estiver disponível.
for i in range(qtd) : # i recebe somente os índices...
    lista[i] = lista [i] * 2 # e o acesso é assim...
print lista # lista ficará com [20, 40, 60, 80]
```

Slices

- Em Python, os slices também são válidos para ter acesso ou recuperar um pedaço de uma lista, com as mesmas regras usadas com strings:
    - Usa-se dois índices separados por : (dois pontos) para significar um intervalo de índices (fechado à esquerda e aberto à direita).
        - Cada um dos limites pode ser omitido e o significado será “a partir do início” e “até o fim”, respectivamente.
        - Se o intervalo for vazio, ou seja, o segundo parâmetro for menor ou igual ao primeiro, o resultado é uma lista vazia – [ ].

```
lista = [10, 20, 30, 40, 50, 60]
ex1 = lista [1:3] # Resultado é [20, 30].
ex2 = lista [0:2] # Resultado é [10, 20].
ex2 = lista [:2] # Mesma coisa...
ex3 = lista [3:6] # Resultado é [40, 50, 60].
ex3 = lista [3:] # Mesma coisa...
nada = lista [1:1] # Resultado é a lista vazia – [ ].
copia = lista [:] # Cria outra lista igual.
mesma = lista # Novo nome para mesmo objeto...
```

Métodos

- append: insere um elemento no final da lista.
- extend: semelhante, mas concatena uma outra lista.
- insert: insere um elemento num dado índice – desloca os elementos para a direita para abrir espaço.
- remove: remove primeira ocorrência de elemento. Se não existir, ocorre um erro fatal!
- index: retorna índice da prim. ocorr. de elemento. Se não existir, ocorre um erro fatal!
- reverse: inverte a ordem dos elementos da lista.
- count: retorna número de ocorr. de elemento na lista.
- sort: ordena uma lista de várias formas... Pode usar uma função de comparação definida pelo usuário... Pode ordenar pelos resultados de uma função qualquer aplicada aos elementos da lista.
- pop: remove um elemento da lista baseado no valor de um índice e o retorna como resultado. Se omitir o índice, remove o último elemento da lista.

```
lista = [20] # Cria uma lista com um só elemento.
lista.append (40) # Resultado é [20, 40].
lista.extend([50, 60]) # Resultado é [20, 40, 50, 60].
lista.remove(50) # Resultado é [20, 40, 60].
lista.remove(70) # Causa erro fatal!
lista.insert(1, 30) # Resultado é [20, 30, 40, 60].
pos = lista.index(40) # Resultado é 2.
pos = lista.index(80) # Causa erro fatal!
lista.reverse() # Resultado é [60, 40, 30, 20].
num = lista.count(20) # Resultado é 1

lista = [30, 20, 50, 10, 40]
lista.sort () # Resultado é [10, 20, 30, 40, 50].
lista.sort(reverse=True) # Res. é [50, 40, 30, 20, 10].
lista.sort (None, None, True) # Mesma coisa...
web = [‘www’, ‘ufpe’, ‘br’]
```

```
web.sort (key=len) # Res. é [‘br’, ‘www’, ’ufpe’]
web.sort (None, len) # Mesma coisa...
r1 = lista.pop ( ) # r1 = 10 e lista = [50, 40, 30, 20].
r2 = lista.pop (1) # r2 = 40 e lista = [50, 30, 20].
r3 = lista.pop (5) # Causa erro fatal!
```

Algumas observações...

- Os elementos de uma lista podem ser atribuídos a variáveis separadas, usando a atribuição múltipla.
    - O número de variáveis utilizado deve ser exatamente igual ao tamanho da lista.
    - Por exemplo, se a variável lista estiver com [10, 20, 30], é possível fazer a atribuição: **a, b, c = lista.**
- Em muitos problemas, o tamanho necessário para uma lista é fixo e conhecido no início.
    - Nestes casos, é interessante/recomendado criar a lista já com o tamanho correto, porque é bem mais eficiente. Por exemplo: lista1 = [None]*qtd ou lista2 = [0]*100.

Listas e strings: funções e métodos

- list: função que transforma um string em uma lista de caracteres.
- split: método que divide um string em uma lista de strings menores. Um segundo string, recebido como parâmetro, serve para definir os pontos de separação e não será parte do conteúdo da lista resultante.
- join: método que insere um string como separador no resultado da transformação de uma lista de strings em um único string.
    - Se for aplicado ao string nulo, o resultado é a simples concatenação dos elementos da lista.

```
fruta = ‘Banana’
fr1 = list (fruta) # Resultado é [‘B’, ‘a’, ‘n’, ‘a’, ‘n’, ‘a’]
```

```
web = ‘www.ufpe.br’
web1 = web.split (‘.’) # Resultado é [‘www’,’ufpe’, ‘br’].
web2 = web.split (‘:’) # Resultado é [‘www.ufpe.br’].
fr2 = “-”.join (fr1) # Resultado é “B-a-n-a-n-a”.
fr3 = “”.join (fr1) # Resultado é “Banana”.
web2 = ‘$’.join (web1) # Resultado é ‘www$ufpe$br’.
num = [10, 20, 30, 40]
erro = ‘’.join (num) # Causa erro fatal!
```

Arrays multidimensionais?

- Já vimos que Python não tem o tipo array.
- Mas os elementos de uma lista podem ser outras listas. Isto permite “simular” matrizes e outros tipos de array. Porém, para mais de 2 dimensões é ruim/complicado!
- *OBS: Não use a notação de repetição para criar arrays multidimensionais pois ela não funcionará da maneira correta... As dimensões mais externas ficarão apenas com cópias dos endereços, ao invés de objetos independentes... Será preciso usar um ou mais comandos for para criar adequadamente a estrutura desejada.*

```
m = [ ] # Cria uma lista vazia
for i in range (3) : # Cria matriz 3x4.
    m.append ([0]*4)
# Agora para manipular os elementos...
for i in range (3) : # O for externo será mais lento...
    for j in range (4) : # Para cada i passa por todos os j.
     m[i][j] = 10*(i+1) + j + 1
for i in range (3) : # Para imprimir como matriz...
    print m [i]
```

Exercícios

1. Ler 2 vetores de tamanho N, com N informado pelo usuário antes, somar os 2 vetores, imprimir os 2 vetores e depois o vetor resultado.
2. Ler uma lista de N números (N é informado pelo usuário antes), e depois criar duas outras listas com os números pares e ímpares separados. No final imprimir as 3 listas.
3. Ler as notas de N alunos (N é informado pelo usuário antes), calcular e imprimir a média das notas e depois imprimir as notas que sejam maiores do que a média calculada.
4. Fazer um programa Python para:
   – Ler uma sequência de números inteiros positivos (ou zero).
   – A leitura deve parar com um número negativo.
   – O programa deve imprimir os números lidos cujos valores têm dois dígitos, mas na ordem inversa em que forem lidos – o último número lido deve ser o primeiro a ser impresso.
5. Altere o programa anterior para garantir que o usuário digitará no máximo 1000 números.

```
print("1. Ler 2 vetores de tamanho N, com N informado pelo usuá
      "imprimir os 2 vetores e depois o vetor resultado.")
tamanho1 = int(input("Digite o tamanho das listas:"))
lista1 = []
lista11 = []
for i1 in range(tamanho1):
    lista1.append(int(input("Digite um elemento da primeira list
for i1 in range(tamanho1):
    lista11.append(int(input("Digite um elemento da segunda list
# Concatenando:
listaf1 = lista1 + lista11
print(f"Duas listas concatenadas: {listaf1}")
# Somando os valores:
listaf11 = lista1
for i1 in range(tamanho1):
    listaf11[i1] = lista1[i1] + lista11[i1]
print(f"Valores somados: {listaf11}\n")
```

```
print("2. Ler uma lista de N números (N é informado pelo usuário
      "com os números pares e ímpares separados. No final imprim
tamanho2 = int(input("Digite o tamanho da lista:"))
lista2 = []
lista2par = []
lista2impar = []
for i2 in range(tamanho2):
    lista2.append(int(input("Digite um elemento da lista:")))
for elemento2 in lista2:
    if elemento2 % 2 == 0:
        lista2par.append(elemento2)
    if elemento2 % 2 != 0:
        lista2impar.append(elemento2)
print(f"Lista original: {lista2}")
print(f"Pares: {lista2par}")
print(f"Ímpares: {lista2impar}\n")


print("3. Ler as notas de N alunos (N é informado pelo usuário a
      "notas e depois imprimir as notas que sejam maiores do que
tamanho3 = int(input("Digite a quantidade de alunos:"))
tamanho33 = int(input("Digite a quantidade de notas:"))
alunos3 = []
soma3 = 0
for i3 in range(tamanho3):
    alunos3.append([0]*tamanho33)
for i3 in range(tamanho3):
    for j3 in range(tamanho33):
        alunos3[i3][j3] = int(input(f"Digite uma nota do aluno {
print("\n")
for i3 in range(tamanho3):
    for j3 in range(tamanho33):
        soma3 += alunos3[i3][j3]
    media3 = soma3/tamanho33
```

```
    print(f"Aluno número {i3}")
    print(f"Notas: {alunos3[i3]}")
    print(f"Média: {media3}")
    print(f"Notas acima da média:", end=' ')
    for j3 in range(tamanho33):
        if alunos3[i3][j3] > media3:
            print(alunos3[i3][j3], end=' ')
    soma3 = 0
    print("\n")


print("4. Fazer um programa Python para: "
      "– Ler uma sequência de números inteiros positivos (ou ze
      "– A leitura deve parar com um número negativo. "
      "– O programa deve imprimir os números lidos cujos valore
      "ordem inversa em que forem lidos – o último número lido
lista4 = []
listafinal4 = []
elemento4 = int(input("Digite um elemento da lista:"))
lista4.append(elemento4)
while elemento4 < 0:
    print("O primeiro número não pode ser negativo.")
    lista4.pop()
    elemento4 = int(input("Digite um elemento da lista:"))
    lista4.append(elemento4)
while elemento4 >= 0:
    elemento4 = int(input("Digite um elemento da lista:"))
    lista4.append(elemento4)
lista4.pop(-1)
for elemento4 in lista4:
    if elemento4 > 9 and elemento4 < 100:
        listafinal4.append(elemento4)
print(f"Lista original: {lista4}")
print(f"Lista final: {listafinal4}")
print(f"Lista final reversa: {list(reversed(listafinal4))}")
```

```
print("5. Altere o programa anterior para garantir que o usuário
lista5 = []
listafinal5 = []
elemento5 = int(input("Digite um elemento da lista:"))
lista5.append(elemento5)
cont = 1
while elemento5 < 0:
    print("O primeiro número não pode ser negativo.")
    lista5.pop()
    elemento5 = int(input("Digite um elemento da lista:"))
    lista5.append(elemento5)
while (cont < 1000) and (elemento5 >= 0):
    elemento5 = int(input("Digite um elemento da lista:"))
    lista5.append(elemento5)
    cont += 1
if elemento5 < 0:
    lista5.pop(-1)
for elemento5 in lista5:
    if (elemento5 > 9) and (elemento5 < 100):
        listafinal5.append(elemento5)
print(f"Lista original: {lista5}")
print(f"Lista final: {listafinal5}")
print(f"Lista final reversa: {list(reversed(listafinal5))}")
```