Lógica de Programação e Linguagem de Programação

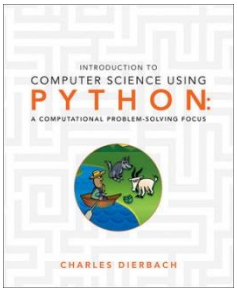
07: LISTAS EM PYTHON

Faculdade IMPACTA

Nossos **objetivos** nesta aula são:

- Entender o que é uma lista na programação de computadores.
- Conhecer quais as operações comumente efetuadas nas listas
- Criar e usar listas em Python
- Entender a diferença entre listas e tuplas em Python
- Entender o que são e como utilizar outras sequências em Python.
- Aprender a percorrer por todos os elementos de uma sequência.

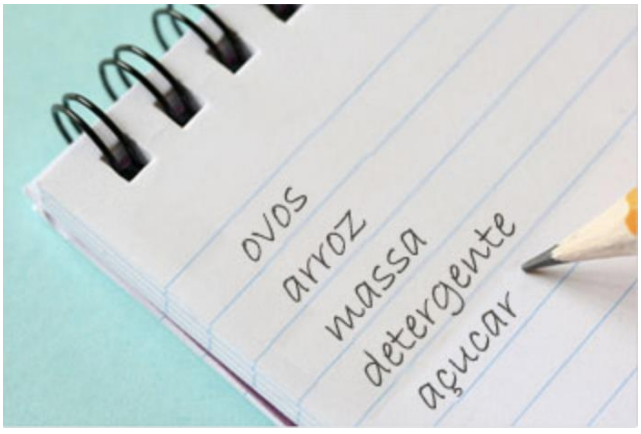
A referência para esta aula é o **Capítulo 4 (Listas)** do livro:



DIERBACH, C. *Introduction to Computer Science Using Python: A Computational Problem Solving Focus.* 1st Edition, New York: Wiley, 2012.

Introdução

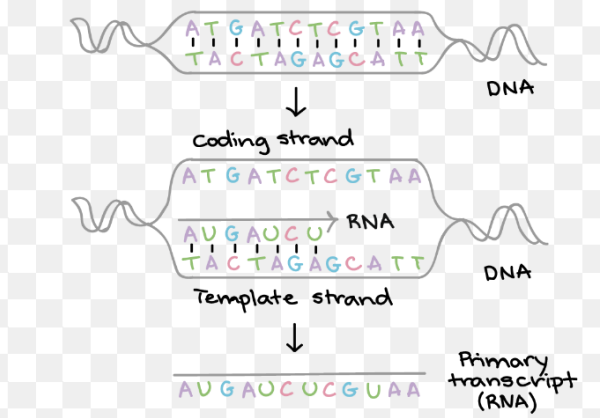
- A forma como os dados são organizados tem um impacto significativo sobre a eficácia com que podem ser usados. Uma das maneiras mais óbvias e úteis de organizar os dados é como uma lista.
- Nós usamos listas no nosso dia-a-dia, fazemos listas de compras, listas de tarefas, listas de convidados.



- Listas também ocorrem na natureza. Nosso DNA é essencialmente uma longa lista de moléculas na forma de uma dupla hélice, encontrada no núcleo de todas as células humanas e todos os organismos vivos. Sua finalidade também é armazenar informações - especificamente, as instruções usadas para construir todas as outras células do corpo - que chamamos de genes. Dados os 2,85 bilhões de nucleotídeos que compõem o genoma humano, determinar seu

sequenciamento (e, assim, entender nossa composição genética) é fundamentalmente um problema computacional.

- Veremos o uso de listas na programação. O conceito de uma lista é semelhante à nossa noção cotidiana de uma lista. Nós lemos (acesso) itens em nossa lista de tarefas, adicionamos itens, cruzamos (deletamos) itens e assim por diante.



O que é uma Lista?

- Uma lista é uma **estrutura de dados linear** que mantém a posição dos seus elementos em uma ordem linear. Ou seja, tem o primeiro elemento, o segundo elemento e assim por diante.
- Abaixo nós temos um exemplo de uma lista de compras. Note que o primeiro elemento da lista é o Cereal, o segundo é o Leite, o terceiro é a Banana etc.

Lista de compras
0 - Cereal
1 - Leite
2 - Banana
3 - Maçã
4 - Iogurte

- A ordem em que os elementos aparecem na lista pode ser a ordem em que o autor planeja pegar os itens no mercado, ou pode ser simplesmente a ordem em que ele anotou os itens à medida que foi se lembrando.

Operações em uma Lista

- As operações que normalmente são realizadas em uma lista são: recuperar, substituir, inserir, remover e acrescentar.

1. Recuperar (*retrieve*)

- É comum querermos recuperar o item que está em uma determinada posição na lista. Na nossa lista de compras, o item que está na posição de índice 2 é a Banana.

Lista de compras
0 - Cereal
1 - Leite
2 - Banana
3 - Maçã
4 - Iogurte

2. Substituir (replace)

- Se trocarmos o item na posição 3 da lista por Mamão estaremos realizando uma substituição.

Antes	*Depois*
0 - Cereal	0 - Cereal
1 - Leite	1 - Leite
2 - Banana	2 - Banana
3 - Maçã	**3 - Mamão**
4 - Iogurte	4 - Iogurte

3. Inserir (insert)

- Suponha que o autor da lista se lembrou de que ele deve pegar mais uma fruta (Melão) antes de ir para a seção de iogurtes. Neste caso, ele irá inserir na posição 4 da lista o novo item.

Antes	*Depois*
0 - Cereal	0 - Cereal
1 - Leite	1 - Leite
2 - Banana	2 - Banana
3 - Maçã	3 - Maçã
4 - Iogurte	**4 - Melão**
	5 - Iogurte

- Note que o item que antes estava na posição número 4 (Iogurte) passou agora para a posição de número 5, e o número total de itens aumentou de 5 para 6.

4. Remover (remove)

- Se o autor da lista não precisar mais de Leite, ele pode remover o item da posição número 1.

Antes	*Depois*
0 - Cereal	0 - Cereal
1 - Leite	1 - Banana
2 - Banana	2 - Maçã
3 - Maçã	3 - Melão
4 - Melão	4 - Iogurte
5 - Iogurte	

- Note que todos os itens que estavam abaixo do Leite na lista "subiram" uma posição.

5. Acrescentar (append)

- Caso o autor da lista queira apenas adicionar mais um item à lista (Mel) sem que isto tenha que ser feito em uma posição específica, o mais fácil é acrescentá-lo no final da lista.

Antes	*Depois*
0 - Cereal	0 - Cereal
1 - Banana	1 - Banana
2 - Maçã	2 - Maçã
3 - Melão	3 - Melão
4 - Iogurte	4 - Iogurte
	5 - Mel

Listas em Python

- Uma lista em Python é uma estrutura de dados linear, mutável (ou seja, pode ser modificada), de comprimento variável, que permite tipos diferentes de elementos. Ela é representada por uma lista de elementos separados por vírgula dentro de colchetes.
- Exemplos de listas:

```
[1, 10, 3]
['cereal', 'banana', 'maçã', 'melão', 'iogurte']
[5, 'maçã', True, 10]
[]
```

- Na última linha destes exemplos temos somente um par de colchetes ([]), o que representa uma **lista vazia**.

- Podemos inicializar uma variável com uma lista fazendo uso do **operador atribuição (=)**.

```
lst = [1, 10, 3]
```

- Após a atribuição, o conteúdo da lista pode ser acessado utilizando a variável.

- Se quisermos **imprimir todos os elementos da lista** que é acessada através desta variável, basta utilizar a função **print**:

```
lista_compras = ['cereal', 'leite', 'banana', 'maçã', 'iogurte']
print(lista_compras)
```

- Para saber o **número de itens** em uma lista, usamos a função **len**:

```
lista_compras = ['cereal', 'leite', 'banana', 'maçã', 'iogurte']
n = len(lista_compras)
print('Número de itens: ', n)
```

Operações nas listas em Python

- Para **recuperar (retrieve)** elementos nós acessamos a lista através do seu índice, que é colocado entre colchetes logo após o nome da variável.

```
lista_compras = ['cereal', 'leite', 'banana', 'maçã', 'iogurte']
print('Elemento na posição de índice 1:', lista_compras[1])
```

- A contagem dos índices em uma lista em Python inicia-se em 0 (zero), e não em 1. Assim, se uma lista possui 5 itens, o primeiro elemento é acessado utilizando o índice 0, o último elemento é acessado utilizando o índice 4, e não há elemento com índice 5.

```
lista_compras = ['cereal', 'leite', 'banana', 'maçã', 'iogurte']
print('Primeiro item:', lista_compras[0])
print('Último item:', lista_compras[4])
```

- Podemos acessar vários elementos da lista em uma única linha de código e fazer operações com eles. Por exemplo:

```
lista_notas = [8.0, 7.5, 7.0, 9.9]
soma = lista_notas[0] + lista_notas[1] + lista_notas[2] + lista_notas[3]
print('Soma:', soma)
```

- Para **substituir (replace),** deve-se fazer a atribuição de um novo valor para uma posição específica da lista.

```
lista_compras = ['cereal', 'leite', 'banana', 'maçã', 'iogurte']
print('Antes:', lista_compras)
lista_compras[3] = 'mamão'
print('Depois:', lista_compras)
```

- A operação de **inserir (insert)** um elemento é realizada utilizando o método insert da lista. Para executá-lo deve-se fazer a chamada utilizando a notação do ponto. Para isso, escreve-se:
    - o nome da variável que acessa a lista seguido por um ponto (.);
    - o nome do método (insert);
    - entre parênteses, o número do índice da posição onde deve ser realizada a inserção e o valor a ser inserido.

```
lista_compras = ['cereal', 'leite', 'banana', 'mamão', 'iogurte']
print('Antes:', lista_compras)
lista_compras.insert(4, 'melão')
print('Depois:', lista_compras)
```

- Para **remover (remove)** um elemento, deve-se utilizar a sentença **del** seguida do nome da variável e o índice do elemento que será removido.

```
lista_compras = ['cereal', 'leite', 'banana', 'mamão', 'melão', 'iogurte']
print('Antes:', lista_compras)
del lista_compras[1]
print('Depois:', lista_compras)
```

- A operação de **acrescentar (append)** um elemento no final da lista é feita utilizando o método append, informando qual o elemento que deve ser adicionado.

```
lista_compras = ['cereal', 'banana', 'mamão', 'melão', 'iogurte']
print('Antes:', lista_compras)
lista_compras.append('mel')
print('Depois:', lista_compras)
```

- Além disso, em Python temos à disposição o método **sort** (para alterar a posição dos elementos na lista de forma que fiquem em ordem crescente):

```
lista_compras = ['cereal', 'banana', 'mamão', 'melão', 'iogurte', 'mel']
print('Antes:', lista_compras)
lista_compras.sort()
print('Depois:', lista_compras)
```

- No caso deste exemplo, como os elementos da lista são strings, o método sort colocou os elementos em ordem alfabética.

- Há também o método **reverse**, que inverte a posição de todos os elementos.

```
lista_compras = ['cereal', 'banana', 'mamão', 'melão', 'iogurte', 'mel']
print('Antes:', lista_compras)
lista_compras.reverse()
print('Depois:', lista_compras)
```

Exercício 1

a) Quais serão os resultados apresentados se tentarmos executar as seguintes linhas em um interpretador Python?

```
>>> lst = [10, 20, 30]
>>> lst
???
>>> lst[0]
???
>>> lst[0] = 5
>>> lst
???
>>> del lst[2]
>>> lst
???
>>> lst.insert(1,15)
>>> lst
???
>>> lst.append(40)
>>> lst
???
```

b) Qual será a faixa dos valores dos índices para uma lista de 10 elementos?

1. 0-9
2. 0-10
3. 1-10
4. 1-9

c) Qual dos seguintes itens NÃO é uma operação comum em listas?

1. recuperação
2. substituição
3. entrelaçamento
4. inserção
5. remoção

d) Qual será o resultado se inserirmos na lista abaixo o valor 50 na posição de índice 2?

índice	*elemento*
0	35
1	15
2	45
3	28

O que é uma Sequência

- Uma sequência em Python é um conjuntos de elementos que podem ser acessados por um índice numérico e que estão ordenados de forma linear.
- Desta maneira, a **lista** que acabamos de ver é uma **sequência**. No entanto, há outras sequências: tuplas e strings.
- Uma string é uma sequência cujos elementos são os próprios caracteres que formam a string.
- As tuplas estão sendo trazidas ao nosso conhecimento nesta aula.

O que é uma Tupla

- Uma **tupla** é uma estrutura de dados linear **imutável**. Ou seja, uma tupla não pode ser alterada após a sua definição. Por isso nós, que aprendemos as listas na última seção, podemos entender a tupla como uma lista que não pode ser alterada.
- Em Python, uma tupla é representada em um conjunto de elementos dentro de **parênteses** e separados por vírgula.

```
(10, 20, 30)
('Marcos', 1.68, 20)
```

- Podemos atribuir uma tupla a uma variável, e a partir disto realizar operações na tupla por meio desta variável. O exemplo abaixo mostra a atribuição de uma tupla a uma variável, a impressão do conteúdo desta variável e a recuperação dos elementos através dos índices numéricos.

```
aluno = ('Marcos', 1.68, 20)
print(aluno)
nome = aluno[0]
altura = aluno[1]
idade = aluno[2]
print('Nome:', nome)
print('Altura:', altura)
print('Idade:', idade)
```

- Se formos definir uma **tupla com um único elemento**, devemos tomar o cuidado de adicionar uma vírgula logo após o elemento.

```
notas = (10.0,)
print(notas)
```

- Se não incluírmos a vírgula, o interpretador não entenderá isto como uma tupla, mas somente como o próprio elemento.

```
notas = (10.0)
print(notas)
```

- Uma tupla vazia é representada por um par de parênteses.

```
vazia = ()
print(vazia)
```

- Mais à frente no nosso curso vamos aprender possíveis usos para tuplas de um único elemento e tuplas vazias.
- Das 5 operações que aprendemos a realizar em uma lista (recuperar, substituir, inserir, remover e acrescentar) a única que pode ser realizada em tuplas é a operação de recuperar.

```
gastos = (150.01, 200.23, 22.55)
print('Segundo elemento:', gastos[1])
```

Exercício 2

Quais serão os resultados apresentados se tentarmos executar as seguintes linhas em um interpretador Python?

```
>>> t = (10, 20, 30)
>>> print(t[0])
???

>>> del t[1]
???

>>> t.insert(1,15)
???

>>> t.append(40)
???
```

Operações em Sequências

- Há algumas operações que podem ser executadas em todas as sequências, não importando se são listas, tuplas ou strings.

1. Length

- Sintaxe (para uma sequência acessada pela variável s):

```
len(s)
```

- Descrição: Obtém o número de elementos dentro de uma sequência.
- Exemplos:

```
string = 'Tudo bem?'
tupla = (40, 33, 22, 53, 98, 10)
lista = [25, 44, 99]
print(len(string), len(tupla), len(lista))
```

2. Select

- Sintaxe (para uma sequência acessada pela variável s, na posição indice):

```
s[indice]
```

- Descrição: Recupera o elemento na posição com o índice numérico informado.
- Exemplos:

```
string = 'Tudo bem?'
tupla = (40, 33, 22, 53, 98, 10)
lista = [25, 44, 99]
print(string[2], tupla[2], lista[2])
```

3. Slice

- Sintaxe (para uma sequência acessada pela variável s, posições indice1 até indice2):

```
s[indice1:indice2]
```

- Descrição: Recupera uma "subsequência" formada pelos elementos que estão entre as posições de índices informados, incluindo o elemento da posição *indice1* mas sem incluir o elemento que está na posição *indice2*. Se o *indice2* não for informado, isto quer dizer que deverá retornar os elementos até o final da sequência original.
- Exemplos:

```
string = 'Tudo bem?'
tupla = (40, 33, 22, 53, 98, 10)
lista = [25, 44, 99]
print(string[1:3])
print(string[1:])
print(tupla[1:3])
print(tupla[1:])
print(lista[1:3])
print(lista[1:])
```

4. Count

- Sintaxe (para uma sequência acessada pela variável s, deseja-se contar as ocorrências de elemento):

```
s.count(elemento)
```

- Descrição: Obtém o número de ocorrências de um elemento dentro da sequência.
- Exemplos:

```
string = 'Eu estou bem'
tupla = (40, 33, 22, 53, 98, 10)
lista = [25, 44, 99]
print('ocorrências da letra e na string:', string.count('e'))
print('ocorrências do número 22 na tupla:', tupla.count(22))
print('ocorrências do número 100 na lista:', lista.count(100))
```

5. Index

- Sintaxe (para um determinado elemento em uma sequência acessada pela variável s):

```
s.index(elemento)
```

- Descrição: Obtém o índice da primeira ocorrência do elemento dentro da sequência.
- Exemplos:

```
string = 'Eu estou bem'
tupla = (40, 33, 22, 53, 98, 10)
lista = [25, 44, 99]
print("índice da primeira ocorrência da letra 'e' na string:",
string.index('e'))
print('índice da primeira ocorrência do número 22 na tupla:', tupla.index(22))
print('índice da primeira ocorrência do número 44 na lista:', lista.index(44))
```

- Se procurarmos por um elemento que não está na lista, isto irá gerar um erro (ValueError) que interromperá a execução.

```
lista = [25, 44, 99]
print('índice da primeira ocorrência do número 100 na lista:',
lista.index(100))
print('fim')
```

- Note que este erro causa a interrupção da execução (a string 'fim' não chega a ser apresentada). Aprenderemos a tratar este tipo de erro no curso mais à frente.

6. Membership

- Sintaxe (para um determinado elemento em uma sequência acessada pela variável s):

```
elemento in s
```

- Descrição: Verifica se o elemento está ou não está presente na sequência.
- Exemplos:

```
string = 'Eu estou bem'
tupla = (40, 33, 22, 53, 98, 10)
lista = [25, 44, 99]
print("letra 'e' está na string:", 'e' in string)
print('número 22 está na tupla:', 22 in tupla)
print('número 100 está na lista:', 100 in lista)
```

7. Concatenação

- Sintaxe (para duas sequências, acessadas pelas variáveis s1 e s2):

```
s1 + s2
```

- Descrição: Cria uma nova sequência que é composta pelos elementos da sequência *s1* seguidos pelos elementos da sequência *s2*.
- Conhecemos o operador + como aquele que realiza a adição de 2 números. No entanto, quando ele é utilizado com sequências passa a realizar a concatenação das sequências. Por ter comportamentos diferentes de acordo com o tipo dos operandos, dizemos que o operador + é um operador **sobrecarregado**.
- Exemplos:

```
string1 = 'Eu estou bem'
string2 = ', e você?'
tupla1 = (40, 33, 22, 53, 98, 10)
tupla2 = (1,)
lista1 = [25, 44, 99]
lista2 = [55, 100]
print(string1 + string2)
print(tupla1 + tupla2)
print(lista1 + lista2)
```

8. Mínimo, máximo e soma

- Sintaxe (para uma sequência acessada pela variável s):

```
min(s)
max(s)
sum(s)
```

- Descrição: retornam, respectivamente, o elemento com o menor valor, o elemento com o maior valor e a soma dos elementos da sequência.
- Quando aplicados a uma string, as funções min e max determinam o menor e o maior elemento de acordo com a posição do elemento dentro da **ordem alfabética**.
- A função sum não se aplica a strings.
- Exemplos:

```
string = 'elemento'
tupla = (40, 33, 22, 53, 98, 10)
lista = [25, 44, 99]
print("menor elemento da string:", min(string))
print("maior elemento da string:", max(string))
print("menor elemento da tupla:", min(tupla))
print("maior elemento da tupla:", max(tupla))
print('soma dos elementos da tupla:', sum(tupla))
print("menor elemento da lista:", min(lista))
print("maior elemento da lista:", max(lista))
print('soma dos elementos da lista:', sum(lista))
```

9. Comparação

- Sintaxe (para duas sequências, acessadas pelas variáveis s1 e s2):

```
s1 == s2
```

- Descrição: retorna True se as duas sequências forem do mesmo tamanho e se os seus elementos correspondentes forem iguais.
- Exemplos:

```
string1 = 'Eu estou bem'
string2 = ', e você?'
tupla1 = (40, 33, 22, 53, 98, 10)
tupla2 = (40, 33, 22, 53, 98, 10)
lista1 = [25, 44, 99]
lista2 = [44, 25, 99]
print('strings são iguais:', string1 == string2)
print('tuplas são iguais:', tupla1 == tupla2)
print('listas são iguais:', lista1 == lista2)
```

Exercício 3

Quais serão os resultados apresentados se tentarmos executar as seguintes linhas em um interpretador Python?

```
>>> s = "amendoim"
>>> s[4:7]
???

>>> s.count('m')
???

>>> s.index('m')
???

>>> s + ' refrigerante'
???

>>> t = (10, 20, 30, 10)
>>> t[1:3]
???

>>> t.count(10)
???

>>> t.index(10)
???

>>> t + (40, 50)
???

>>> lst = [10, 20, 30, 10]
>>> lst[1:3]
???

>>> lst.count(10)
???

>>> lst.index(10)
???

>>> [40, 50] + lst
???

>>> lst + (40, 50)
???
```

Percorrendo uma Sequência

- Percorrer os elementos de uma sequência (ou realizar a iteração sobre uma sequência) é acessar os elementos um a um, começando pelo primeiro e terminando no último elemento da sequência.
- O percurso pelos elementos de uma sequência pode ser feita através de um laço de repetição.
- Exemplo de percurso utilizando um laço **while**:

```
lista = [10, 20, 30, 5]
indice = 0
tamanho = len(lista)
while indice < tamanho:
    print(lista[indice])
    indice = indice + 1
print('fim')
```

- Python oferece uma maneira conveniente de percorrer os elementos de uma sequência utilizando o laço **for**, com a seguinte sintaxe:

```
for k in sequencia:
    instruções
```

- A execução deste bloco irá fazer com que a variável **k** (que é a **variável do laço**) assuma, a cada execução das **instruções**, o valor de um elemento da sequência, começando pelo primeiro elemento e terminando pelo último.
- Desta forma, o laço que demos como primeiro exemplo nesta seção pode ser implementado da seguinte maneira utilizando o "**for iterativo**":

```
lista = [10, 20, 30, 5]
for x in lista:
    print(x)
print('fim')
```

A função *range*

- A função *range* (que já foi mencionada em aulas anteriores) gera uma sequência de inteiros que pode ser percorrida utilizando o **for iterativo**.

```
for i in range(5):
    print(i)

for i in range(2,15):
    print(i, end=' ')
```

- Desta maneira, também é possível percorrer os elementos de uma sequência utilizando a função range para gerar os índices e, dentro do laço, recuperar o elemento da sequência pelo índice.

```
lista = [10, 20, 30, 5]
for indice in range(len(lista)):
    print(lista[indice])
print('fim')
```

Qual a melhor forma de Percurso em uma Sequência?

Pelo que foi exposto, conhecemos então 3 maneiras de percorrer os elementos de uma sequência:

1) Usando o **for iterativo**

2) Usando o **for iterativo** combinado com a **função *range*** e acessando o elemento pelo índice

3) Usando um laço **while**.

- Quando precisamos, a cada iteração, **apenas consultar** o valor armazenados na sequência, sem alterá-los, o mais simples é utilizar o **for iterativo**. Dizemos que estamos iterando sobre os elementos da lista.

```
lista = [25, 10, 20, 30, 5]
for x in lista:
    print(x)
print('fim')
```

- Quando precisamos, a cada iteração, **alterar** o valor dos elementos, será necessário **acessar os elementos da sequência utilizando o índice.** Dizemos que estamos iterando sobre os valores dos índices da lista.
- Suponha, por exemplo, que desejamos incrementar de 1 unidade cada elemento armazenado em uma lista:

```
lista = [25, 10, 20, 30, 5]
for indice in range(len(lista)):
    lista[indice] += 1
print(lista)
print('fim')
```

- Quando precisamos percorrer uma sequência e este percurso **puder ser interrompido se uma determinada condição for satisfeita**, o uso do laço com while pode ser a opção mais adequada. Suponha que desejamos achar o índice do primeiro número par que aparece em uma tupla:

```
# teste este trecho de código para outros elementos nesta tupla
tupla = (25, 15, 20, 30, 5, 100)

indice = 0
tamanho = len(tupla)
par = False
indicePrimeiroPar = -1
while indice < tamanho and not par:
    if tupla[indice] % 2 == 0:
        par = True
        indicePrimeiroPar = indice
    indice = indice + 1


if par:
    print('O primeiro par aparece na posição de índice', indicePrimeiroPar)
else:
    print('Não encontrei um número par na sequência')
print('fim')
```

Exercício 4

```
k = 0
sum = 0
nums = range(100)
while k < len(nums) and sum < 100:
    sum = sum + nums[k]
    k = k + 1
print('Os primeiros', k, 'inteiros somam 100 ou mais')
```

LISTA DE EXERCÍCIOS

1) Marque verdadeiro ou falso para as seguintes afirmativas:
   a) Listas em Python são heterogêneas, ou seja, podem armazenar tipos diferentes de dados
   b) Uma lista em Python não pode aumentar e diminuir de tamanho.
   c) Listas em Python não são mutáveis.
   d) Uma lista deve conter pelo menos um item.
   e) Itens da lista podem ser removidos utilizando o operador del.

2) O método que adiciona um item no final de uma lista é:
   a) extend b) add c) plus d) append

3) Qual dos seguintes não é um método de listas em Python:
   a) index b) insert c) get d) pop

4) Qual característica não se aplica a lista em Python?
   a) É uma sequência
   b) É imutável
   c) É heterogênea
   d) É dinâmica

5) Escreva um programa em Python com uma função que recebe por parâmetro uma lista de números inteiros e retorna o índice em que se encontra o **maior elemento** da lista.

6) Escreva um programa em Python com uma função que recebe por parâmetro uma lista de números inteiros e retorna o índice em que se encontra o **menor elemento** da lista.

7) Escreva um programa em Python com uma função que recebe por parâmetro uma lista de números inteiros e um número inteiro qualquer X e retorna a **quantidade de vezes que X** encontra-se na lista.

8) Python provê métodos que auxiliam a realizar **operações em uma lista**, como por exemplo inverter os elementos de uma lista (lista.reverse()). Em outras linguagens, é necessário

escrever a própria função para realizar uma operação em lista. Escreva um programa em Python com as seguintes funções, que fazem as mesmas tarefas de alguns métodos de lista:

a) conta(lista, x): retornar o número de ocorrências de x na lista (assim como lista.count(x)).
b) existe(lista,x): verificar e retornar a existência de x na lista (assim como x in list)
c) indice(lista, x): retornar o índice da primeira ocorrência de x (assim como lista.index(x)) Caso não encontre, retorna -1.
d) inverte(lista): inverter e retornar os elementos de uma lista (assim como lista.reverse())
e) insere(lista, x, i): inserir x na posição i da lista (assim como lista.insert(x, i))
f) remove(lista, x): remover a primeira ocorrência do elemento x da lista (assim como lista.remove(x))

Escreva um programa em Python que receba os itens de uma lista de compras digitados pelo usuário. O programa deve continuar pedindo os itens até que o usuário aperte apenas "enter". Depois de capturar todos os itens, o programa deve mostrar a lista de compras e a quantidade de itens da lista.

9) O **Crivo de Eratóstenes** é um algoritmo para encontrar todos os primos até um limite n. A ideia básica é primeiramente criar uma lista com os números de 2 até n. O primeiro número da lista é removido e anunciado como primo. Então, todos os múltiplos deste número são removidos da lista. O processo continua até que a lista esteja vazia.

   Por exemplo, se queremos encontrar todos os primos até 10, a lista deve iniciar da seguinte forma: [2, 3, 4, 5, 6, 7, 8, 9, 10]. O 2 é removido (primeiro elemento da lista) e anunciado como primo. Os números 4, 6, 8 e 10 são removidos, já que são múltiplos de 2. A lista remanescente contém [3, 5, 7, 9]. O processo é repetido, removendo o número 3 (novo primeiro elemento da lista) e anunciando como primo. O número 9 é então removido por ser múltiplo de 3, sobrando [5, 7]. O algoritmo continua anunciando o 5 como primo e removendo da lista. Por fim, o 7 é anunciado como primo e removido da lista, terminando o algoritmo. (veja animação em: https://pt.wikipedia.org/wiki/Crivo_de_Erat%C3%B3stenes)
   Escreva um programa que peça para o usuário entrar com um número n e utilize o Crivo de Eratóstenes para encontrar todos os primos menores ou iguais a n.

10) Faça um programa que receba a **temperatura média** de cada mês do ano e armazene-as em uma lista. Após isto, calcule a média anual das temperaturas e mostre todas as temperaturas acima da média anual, e em que mês elas ocorreram (mostrar o mês por extenso: 1 – Janeiro, 2 – Fevereiro, . . .).

11) Utilizando listas faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são:

a) "Telefonou para a vítima?"

b) "Esteve no local do crime?"

c) "Mora perto da vítima?"

d) "Devia para a vítima?"

e) "Já trabalhou com a vítima?"

O programa deve no final emitir uma **classificação sobre a participação da pessoa no crime**. Se a pessoa responder positivamente a 2 questões ela deve ser classificada como "Suspeita", entre 3 e 4 como "Cúmplice" e 5 como "Assassino". Caso contrário, ele será classificado como "Inocente".

12) Sejam dois conjuntos, A e B, com n e m elementos respectivamente. Os conjuntos não possuem elementos repetidos e não estão ordenados. Faça um programa em Python com:

a) uma função para efetuar a **intersecção entre dois conjuntos**, ou seja, os elementos em comum entre os dois conjuntos. O conjunto C conterá a intersecção de A e B.

Exemplo:

A = { 7, 2, 5, 8, 4} e B= {4, 2, 5}, C = A ∩ B = {2, 5, 4}

A = { 3, 9, 11} e B= {2, 6, 1}, C = A ∩ B = {}

b) uma função para efetuar a **união de dois conjuntos**. O conjunto C conterá todos os elementos de A e B, sem repetição.

Exemplo:
A = { 7, 2, 5, 8, 4} e B= {4, 2,5, 10}, C = A ∪ B = {7,2, 5,8, 4,10}
A = { 3, 9, 11} e B= {2, 6, 1}, C = A ∪ B = {3,9,11,2,6,1}

13) A **intercalação** é o processo utilizado para construir uma lista ordenada, de tamanho n+m, a partir de duas listas já ordenadas de tamanhos n e m. Por exemplo, a partir das sequências abaixo:

Exemplo:
A = { 1, 3, 6, 7} e B= {2, 4, 5}, a nova lista C é feita a partir de A e B:
C = { 1, 2, 3, 5, 6, 7}

Escreva um programa em Python com uma função que faça a intercalação entre duas listas.

14) Escreva um programa em Python com uma função que, dada uma lista de números inteiros com n elementos, **rearranje os elementos da lista** de tal forma que todos os elementos menores ou iguais ao primeiro fiquem à sua esquerda e todos os outros, à sua direita.

Exemplo:
Na sequência {**5**, 6, 2, 7, 9, 1, 8, 3, 7} após ser rearranjada poderá ficar na forma:
{ 2, 1, 3, 5, 6, 7, 9, 8, 7}.