Programação Script

Listas (parte 1)

Aula 07

Prof. Felipe A. Louza

Roteiro

Roteiro

Listas

Uma lista em **Python** é uma estrutura que armazena vários dados, que podem ser de um mesmo tipo ou não.

- Exemplos:

```
1 lista1 = [10, 20, 30, 40]
2 lista2 = ["programação", "ps", "itc", "python"]
3 lista3 = ["oi", 2.0, 5, [10, 20]]
```

Listas

Suponha que desejamos guardar notas de alunos.

- Com o que sabemos, como armazenaríamos 3 notas?

```
1 nota1 = float(input("Digite a nota 1"))
2 nota2 = float(input("Digite a nota 2"))
3 nota3 = float(input("Digite a nota 3"))
```

Listas

Suponha que desejamos guardar notas de alunos.

- Com o que sabemos, como armazenaríamos 3 notas?

```
1 nota1 = float(input("Digite a nota 1"))
2 nota2 = float(input("Digite a nota 2"))
3 nota3 = float(input("Digite a nota 3"))
```

Listas

- E agora, como armazenaríamos 100 notas?

```
nota1 = float(input("Digite a nota 1"))
nota2 = float(input("Digite a nota 2"))
nota3 = float(input("Digite a nota 3"))
nota4 = float(input("Digite a nota 4"))
nota5 = float(input("Digite a nota 5"))
...
nota100 = float(input("Digite a nota 100"))
```

- Criar 100 variáveis distintas não é uma solução elegante .

Listas

- E agora, como armazenaríamos 100 notas?

```
1 nota1 = float(input("Digite a nota 1"))
2 nota2 = float(input("Digite a nota 2"))
3 nota3 = float(input("Digite a nota 3"))
4 nota4 = float(input("Digite a nota 4"))
5 nota5 = float(input("Digite a nota 5"))
6 ...
7 nota100 = float(input("Digite a nota 100"))
```

- Criar 100 variáveis distintas não é uma solução elegante .

Listas

- E agora, como armazenaríamos 100 notas?

```
1 nota1 = float(input("Digite a nota 1"))
2 nota2 = float(input("Digite a nota 2"))
3 nota3 = float(input("Digite a nota 3"))
4 nota4 = float(input("Digite a nota 4"))
5 nota5 = float(input("Digite a nota 5"))
6 ...
7 nota100 = float(input("Digite a nota 100"))
```

- Criar 100 variáveis distintas não é uma solução elegante .

Roteiro

Definição de Listas

Coleção ordenada de objetos referenciados por um identificador único.

```
identificador = [dado_1, dado_2, ..., dado_n]
```

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
```

- Características:
  - Acesso por meio de um índice inteiro
  - Listas podem ser modificadas
  - Pode se incluir e remover itens de listas

Definição de Listas

Coleção ordenada de objetos referenciados por um identificador único.

```
identificador = [dado_1, dado_2, ..., dado_n]
```

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
```

- Características:
  - Acesso por meio de um índice inteiro.
  - Listas podem ser modificadas.
  - Pode-se incluir e remover itens de listas.

Definição de Listas

Coleção ordenada de objetos referenciados por um identificador único.

```
identificador = [dado_1, dado_2, ..., dado_n]
```

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
```

- Características:
  - Acesso por meio de um índice inteiro.
  - Listas podem ser modificadas.
  - Pode-se incluir e remover itens de listas.

Definição de Listas

Coleção ordenada de objetos referenciados por um identificador único.

```
identificador = [dado_1, dado_2, ..., dado_n]
```

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
```

- Características:
  - Acesso por meio de um índice inteiro.
  - Listas podem ser modificadas.
  - Pode-se incluir e remover itens de listas.

Exemplo de listas

- Uma lista de inteiros.

```
1 x = [2, 45, 12, 9, -2]
```

- Listas podem conter dados de tipos diferentes.

```
1 x = [2, "qwerty", 45.99087, 0, 'a']
```

- Listas podem conter outras listas:

```
1 x = [2, [4,5], [9]]
```

- Listas podem ser vazias:

```
1 x = []
```

Exemplo de listas

- Uma lista de inteiros.

```
1 x = [2, 45, 12, 9, -2]
```

- Listas podem conter dados de tipos diferentes.

```
1 x = [2, "qwerty", 45.99087, 0, 'a']
```

- Listas podem conter outras listas:

```
1 x = [2, [4,5], [9]]
```

- Listas podem ser vazias:

```
1 x = []
```

Exemplo de listas

- Uma lista de inteiros.

```
1 x = [2, 45, 12, 9, -2]
```

- Listas podem conter dados de tipos diferentes.

```
1 x = [2, "qwerty", 45.99087, 0, 'a']
```

- Listas podem conter outras listas:

```
1 x = [2, [4,5], [9]]
```

- Listas podem ser vazias:

```
1 x = []
```

Exemplo de listas

- Uma lista de inteiros.

```
1 x = [2, 45, 12, 9, -2]
```

- Listas podem conter dados de tipos diferentes.

```
1 x = [2, "qwerty", 45.99087, 0, 'a']
```

- Listas podem conter outras listas:

```
1 x = [2, [4,5], [9]]
```

- Listas podem ser vazias:

```
1 x = []
```

Definição de Listas

- Objetos em uma lista não precisam ser únicos.

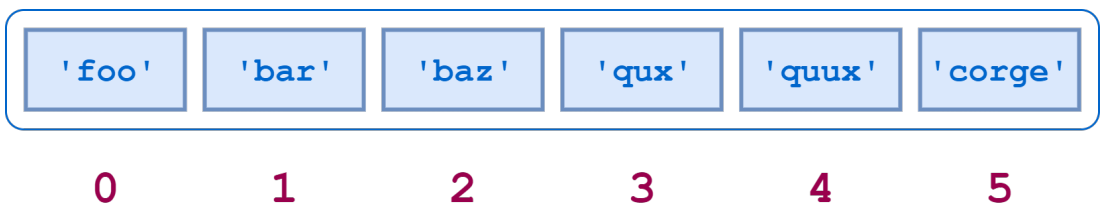
```
>>> lista = ['bark', 'meow', 'woof', 'bark', 'cheep', 'bark']
>>> lista
['bark', 'meow', 'woof', 'bark', 'cheep', 'bark']
```

Definição de Listas

Elementos de uma lista podem ser acessados por um índice inteiro.

- A primeira posição: `lista[0]`

```
>>> lista = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> lista[0]
'foo'
>>> lista[2]
'baz'
>>> lista[5]
'corge'
```

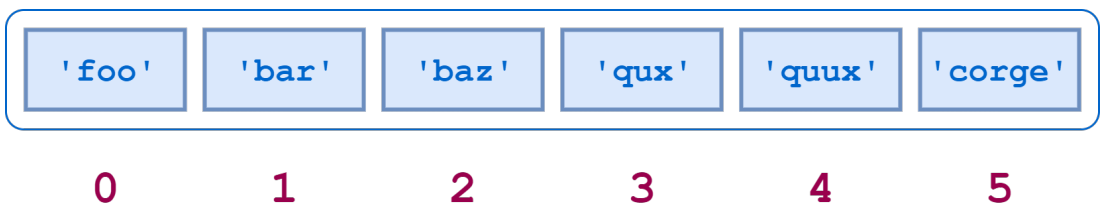


Definição de Listas

Elementos de uma lista podem ser acessados por um índice inteiro.

- A primeira posição: `lista[0]`

```
>>> lista = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> lista[0]
'foo'
>>> lista[2]
'baz'
>>> lista[5]
'corge'
```

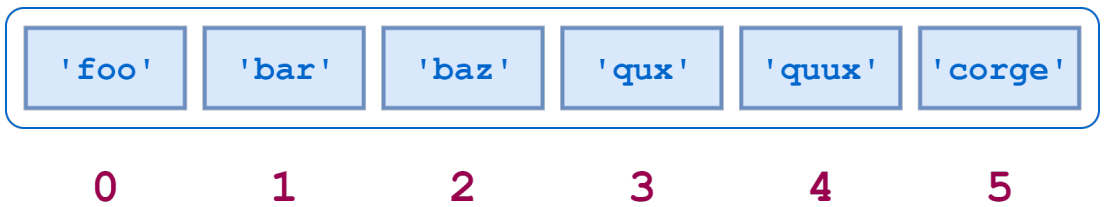


Definição de Listas

Cuidado:

- Ocorre um erro se tentarmos acessar uma posição inexistente na lista, por exemplo `lista[6]`.

```
>>> lista = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> lista[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

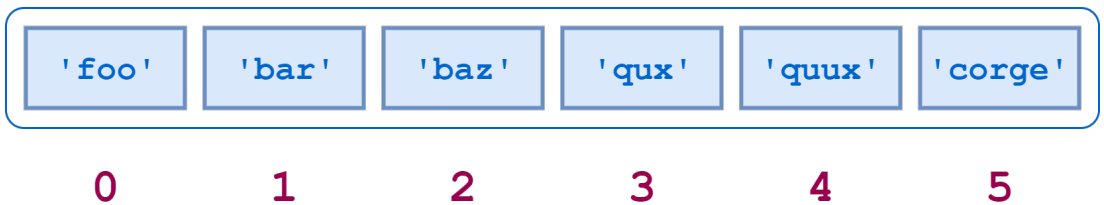


Definição de Listas

Cuidado:

- Ocorre um erro se tentarmos acessar uma posição inexistente na lista, por exemplo `lista[6]`.

```
>>> lista = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> lista[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```



Definição de Listas

No exemplo de notas: tamanho `n = 5`

- Os índices válidos são de 0 até 4 (5-1).

```
1 notas = [8.0, 5.5, 9.3, 7.6, 3.1]
2 print(notas[0])
3 print(notas[1])
4 print(notas[2])
5 print(notas[3])
6 print(notas[4])
```

```
1 8.0
2 5.5
3 9.3
4 7.6
5 3.1
```

Definição de Listas

No exemplo de notas: tamanho `n = 5`

- Os índices válidos são de 0 até 4 (5-1).

```
1 notas = [8.0, 5.5, 9.3, 7.6, 3.1]
2 print(notas[0])
3 print(notas[1])
4 print(notas[2])
5 print(notas[3])
6 print(notas[4])
```

```
1 8.0
2 5.5
3 9.3
4 7.6
5 3.1
```

Definição de Listas

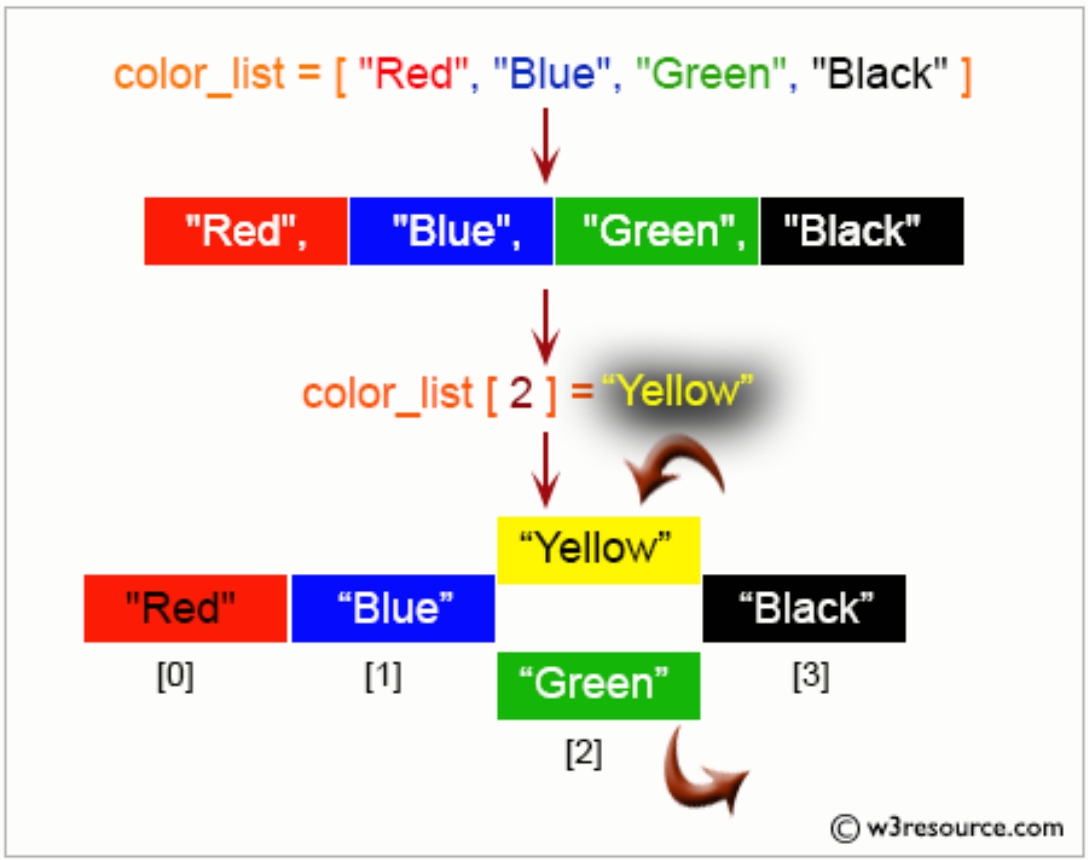
Cada elemento de uma lista tem o mesmo comportamento que uma variável simples.

```
>>> notas = [8.0, 5.5, 9.3, 7.6, 3.1]

>>> print(notas[0]+2)
10.0

>>> notas[3] = 0.5

>>> print(notas)
[8.0, 5.5, 9.3, 0.5, 3.1]
```

- O 4º elemento foi modificado.

Definição de Listas

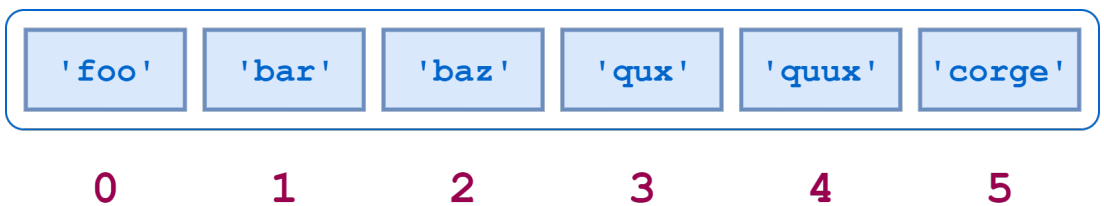
- Exemplo:



Definição de Listas

- É possível acessar os elementos na ordem inversa.

```
1 >>> lista = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
2
3 >>> lista[-1]
4 'corge'
5 >>> lista[-2]
6 'quux'
7 >>> lista[-5]
8 'bar'
```



Definição de Listas

Cuidado:

- Ocorre um erro se tentarmos acessar uma posição inexistente na lista, por exemplo `lista[-6]`.

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
print(notas[-1])
print(notas[-2])
print(notas[-3])
print(notas[-4])
print(notas[-5])
print(notas[-6])
```

```
8.0
5.5
9.3
7.6
3.1
IndexError: list index out of range
```

Definição de Listas

Cuidado:

- Ocorre um erro se tentarmos acessar uma posição inexistente na lista, por exemplo `lista[-6]`.

```
1 notas = [8.0, 5.5, 9.3, 0.5, 3.1]
2 print(notas[-1])
3 print(notas[-2])
4 print(notas[-3])
5 print(notas[-4])
6 print(notas[-5])
7 print(notas[-6])
```

```
1 8.0
2 5.5
3 9.3
4 7.6
5 3.1
6 IndexError: list index out of range
```

Definição de Listas

Cuidado:

- Ocorre um erro se tentarmos acessar uma posição inexistente na lista, por exemplo `lista[-6]`.

```
1 notas = [8.0, 5.5, 9.3, 0.5, 3.1]
2 print(notas[-1])
3 print(notas[-2])
4 print(notas[-3])
5 print(notas[-4])
6 print(notas[-5])
7 print(notas[-6])
```

```
1 8.0
2 5.5
3 9.3
4 7.6
5 3.1
6 IndexError: list index out of range
```

Usando uma Lista

- O índice usado para acessar um elemento da lista pode ser uma variável inteira.

```
>>> lista= [4.5, 8.6, 9, 7.8, 7]
>>> i = 0
>>> while(i < 5):
...   print(lista[i])
...   i+=1
```

```
4.5
8.6
9
7.8
7
```

Usando uma Lista

- Quais valores estarão armazenados em cada posição da lista após a execução deste código abaixo?

```
1 lista = [0, 0, 0, 0, 0]
2
3 i = 0
4 while(i < 5):
5    lista[i] = i+1
6    i+=1
7
8 print(lista)
```

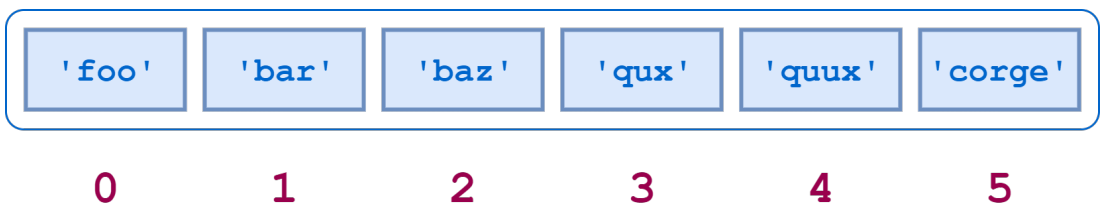
Roteiro

O comando `for`

Podemos percorrer todos os elementos de uma lista com o comando `for`:

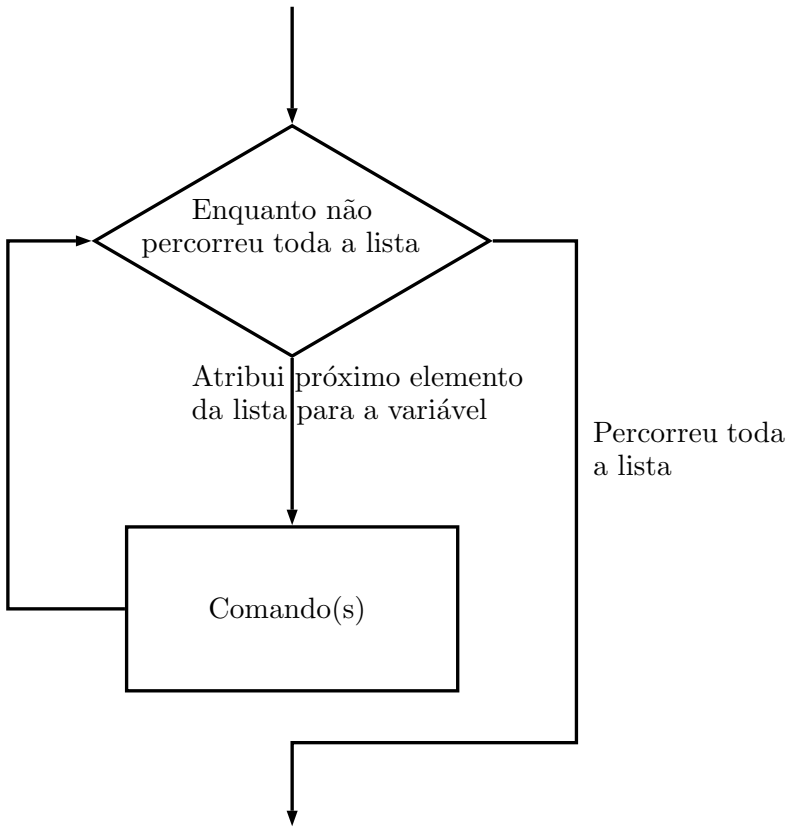
```
1 for item in lista:
2   print(item)
```

- Para cada elemento da lista, em ordem, é atribuído este elemento à variável `item` e então é executado o bloco de comandos.



O comando `for`

- Passo 1: Verifica se percorreu toda a lista.
  - Se não percorreu, `item` recebe o **próximo elemento** da `lista`.
  - Se percorreu, encerra o laço.
- Passo 2: Executa comandos.
- Passo 3: Volta ao Passo 1.



O comando `for`

- O programa abaixo usa o laço `for` para imprimir números de uma lista.

```
1 # Imprime todos os números de uma lista
2 lista_numeros = [1, 2, 3, 4, 5]
3
4 for item in lista_numeros:
5   print(item)
```

```
1 1
2 2
3 3
4 4
5 5
```

O comando `for`

Observe que `item` possuí apenas uma cópia de um valor da lista.

- O que acontece se alterarmos `item`?

```
1 # Imprime todos os números de uma lista
2 lista_numeros = [1, 2, 3, 4, 5]
3
4 for item in lista_numeros:
5   item = 0
6   print(item)
7
8 print(lista_numeros)
```

```
1 0
2 0
3 0
4 0
5 0
6 [1, 2, 3, 4, 5]
```

– Nada!

O comando `for`

Observe que `item` possuí apenas uma cópia de um valor da lista.

- O que acontece se alterarmos `item`?

```
1 # Imprime todos os números de uma lista
2 lista_numeros = [1, 2, 3, 4, 5]
3
4 for item in lista_numeros:
5   item = 0
6   print(item)
7
8 print(lista_numeros)
```

```
1 0
2 0
3 0
4 0
5 0
6 [1, 2, 3, 4, 5]
```

  - Nada!

A função `range`

Em Python o comando `range(n)` gera um intervalo $[0, n)$, isto é, valores de 0 até $n - 1$.

```
>>> type(range(10))
<class 'range'>
```

- Podemos iterar com o `for` em um intervalo:

```
for i in range(10):
  print(i)
```

  - A cada iteração `i` assume um valor do intervalo.
  - Qual seria o código equivalente usando `while`?

A função `range`

Em Python o comando `range(n)` gera um intervalo $[0, n)$, isto é, valores de 0 até $n - 1$.

```
>>> type(range(10))
<class 'range'>
```

- Podemos iterar com o `for` em um intervalo:

```
for i in range(10):
  print(i)
```

  - A cada iteração `i` assume um valor do intervalo.
  - Qual seria o código equivalente usando `while`?

A função `range`

Em Python o comando `range(n)` gera um intervalo $[0, n)$, isto é, valores de 0 até $n - 1$.

```
>>> type(range(10))
<class 'range'>
```

- Podemos iterar com o `for` em um intervalo:

```
for i in range(10):
  print(i)
```

  - A cada iteração `i` assume um valor do intervalo.
  - Qual seria o código equivalente usando `while`?

A função `range`

Alterando o `início`:

- Podemos especificar um intervalo de valores na função `range`:
  - `range(inicio, fim)`: gera-se números de *inicio* até *fim* − 1.

```
1 inicio, inicio+1, inicio+2, ..., fim-1
```

- O programa abaixo imprime os números de 5 até 9.

```
1 # Imprime todos os números de 5 a 9
2 for i in range(5, 10):
3   print(i)
```

  - Qual seria o código equivalente usando `while`?

A função `range`

Alterando o `início`:

- Podemos especificar um intervalo de valores na função `range`:
  - `range(inicio, fim)`: gera-se números de *inicio* até *fim* − 1.

```
1 inicio, inicio+1, inicio+2, ..., fim-1
```

- O programa abaixo imprime os números de 5 até 9.

```
1 # Imprime todos os números de 5 a 9
2 for i in range(5, 10):
3   print(i)
```

  - Qual seria o código equivalente usando `while`?

A função `range`

Alterando o `início`:

- Podemos especificar um intervalo de valores na função `range`:
  - `range(inicio, fim)`: gera-se números de *inicio* até *fim* − 1.

```
1 inicio, inicio+1, inicio+2, ..., fim-1
```

- O programa abaixo imprime os números de 5 até 9.

```
1 # Imprime todos os números de 5 a 9
2 for i in range(5, 10):
3   print(i)
```

  - Qual seria o código equivalente usando `while`?

A função `range`

Alterando o `início`:

- Podemos especificar um intervalo de valores na função `range`:
  - `range(inicio, fim)`: gera-se números de *inicio* até *fim* − 1.

```
1 inicio, inicio+1, inicio+2, ..., fim-1
```

- O programa abaixo imprime os números de 5 até 9.

```
1 # Imprime todos os números de 5 a 9
2 for i in range(5, 10):
3   print(i)
```

  - Qual seria o código equivalente usando `while`?

A função `range`

Alterando o passo:

- Podemos definir um passo a ser considerado no intervalo de valores [*inicio*, *fim*).
  - `range(inicio, fim, passo)`: gera-se valores a partir de *inicio* com incremento de *passo* até *fim* − 1.

```
1 inicio, inicio+passo, inicio+2*passo, ..., fim-1
```

- O programa abaixo imprime os **números pares** entre 0 e menores que 13.

```
1 # Imprime todos os números pares entre 0 e 13
2 for i in range(0, 13, 2):
3   print(i)
```

  - Qual seria o código equivalente usando `while`?

A função `range`

Alterando o passo:

- Podemos definir um passo a ser considerado no intervalo de valores [*inicio*, *fim*).
  - `range(inicio, fim, passo)`: gera-se valores a partir de *inicio* com incremento de *passo* até *fim* − 1.

```
1 inicio, inicio+passo, inicio+2*passo, ..., fim-1
```

- O programa abaixo imprime os **números pares** entre 0 e menores que 13.

```
1 # Imprime todos os números pares entre 0 e 13
2 for i in range(0, 13, 2):
3   print(i)
```

  - Qual seria o código equivalente usando `while`?

A função `range`

Alterando o passo:

- Podemos definir um passo a ser considerado no intervalo de valores [*inicio*, *fim*).
  - `range(inicio, fim, passo)`: gera-se valores a partir de *inicio* com incremento de *passo* até *fim* − 1.

```
1 inicio, inicio+passo, inicio+2*passo, ..., fim-1
```

- O programa abaixo imprime os **números pares** entre 0 e menores que 13.

```
1 # Imprime todos os números pares entre 0 e 13
2 for i in range(0, 13, 2):
3   print(i)
```

  - Qual seria o código equivalente usando `while`?

Exemplo

Calculando o valor de $n!$ em **Python**:

```
1  def main():
2    n = int(input("Digite o valor de n: "))
3    res = fatorial(n)
4    print("Fatorial de {} é {}".format(n, res))
5
6  def fatorial(n):
7
8    fat = 1 #corresponde a 0!
9    for i in range(1, n+1):
10     #no fim do laço devemos ter i!
11     fat = fat*i
12
13   return fat
14
15 main()
```

Roteiro

Listas: `len`

Podemos descobrir o tamanho de uma lista (número de elementos) com a função `len(lista)`:

```
1 len(lista)
```

- Exemplos:

```
1  >>> lista1 = [16, 5, 4, 4, 7, 9]
2  >>> len(lista1)
3  6
4
5  >>> lista2 = []
6  >>> len(lista2)
7  0
8
9  >>> lista3 = [1, 2, 3, [4, 5]]
10 >>> len(lista3)
11 4
```

  - Sublistas de uma lista são consideradas como um elemento simples.

Listas: `len`

Podemos descobrir o tamanho de uma lista (número de elementos) com a função `len(lista)`:

```
1 len(lista)
```

- Exemplos:

```
1  >>> lista1 = [16, 5, 4, 4, 7, 9]
2  >>> len(lista1)
3  6
4
5  >>> lista2 = []
6  >>> len(lista2)
7  0
8
9  >>> lista3 = [1, 2, 3, [4, 5]]
10 >>> len(lista3)
11 4
```

  - Sublistas de uma lista são consideradas como um elemento simples.

Listas: `len`

Podemos descobrir o tamanho de uma lista (número de elementos) com a função `len(lista)`:

```
1 len(lista)
```

- Exemplos:

```
1  >>> lista1 = [16, 5, 4, 4, 7, 9]
2  >>> len(lista1)
3  6
4
5  >>> lista2 = []
6  >>> len(lista2)
7  0
8
9  >>> lista3 =  [1, 2, 3, [4, 5]]
10 >>> len(lista3)
11 4
```

  - Sublistas de uma lista são consideradas como um elemento simples.

Listas: `len`

Podemos usar a função `len` junto com o laço `while` para percorrer todas as posições de uma lista:

```
1 lista = [6, 5, 3, 4, 7]
2
3 i = 0
4 while(i < len(lista)):
5     print(lista[i])
6     i += 1
```

Listas: `len`

Podemos usar a função `len` junto com o laço `while` para percorrer todas as posições de uma lista:

```
1 lista = [6, 5, 3, 4, 7]
2
3 i = 0
4 while(i < len(lista)):
5     print(lista[i])
6     i += 1
```

Listas: `len`

Da mesma forma, podemos usar a função `len` junto com o laço `for`:

```
1 lista = [6, 5, 3, 4, 7]
2
3 for i in range(len(lista)):
4     print(lista[i])
```

Listas: `len`

Da mesma forma, podemos usar a função `len` junto com o laço `for`:

```
1 lista = [6, 5, 3, 4, 7]
2
3 for i in range(len(lista)):
4     print(lista[i])
```

Listas: `len`

Agora podemos alterar valores na lista com o comando `for`:

- O que acontece se alterarmos `item`?

```
1 lista = [6, 5, 3, 4, 7]
2
3 for i in range(len(lista)):
4   print(lista[i])
5   lista[i] = i+1
6
7 print(lista)
```

```
1 6
2 5
3 3
4 4
5 7
6 [1, 2, 3, 4, 5]
```

Listas: `len`

Agora podemos alterar valores na lista com o comando `for`:

- O que acontece se alterarmos `item`?

```
1 lista = [6, 5, 3, 4, 7]
2
3 for i in range(len(lista)):
4   print(lista[i])
5   lista[i] = i+1
6
7 print(lista)
```

```
1 6
2 5
3 3
4 4
5 7
6 [1, 2, 3, 4, 5]
```

Listas: append

Podemos acrescentar um item no final de uma lista.

- Isto é feito pela função `append()`

```
lista.append(item)
```

- Exemplo:

```
>>> lista = [6, 5]
>>> lista.append(98)

>>> lista
[6, 5, 98]
```

  - Formalmente, este tipo de função é chamada de método.

Listas: append

Podemos acrescentar um item no final de uma lista.

- Isto é feito pela função `append()`

```
lista.append(item)
```

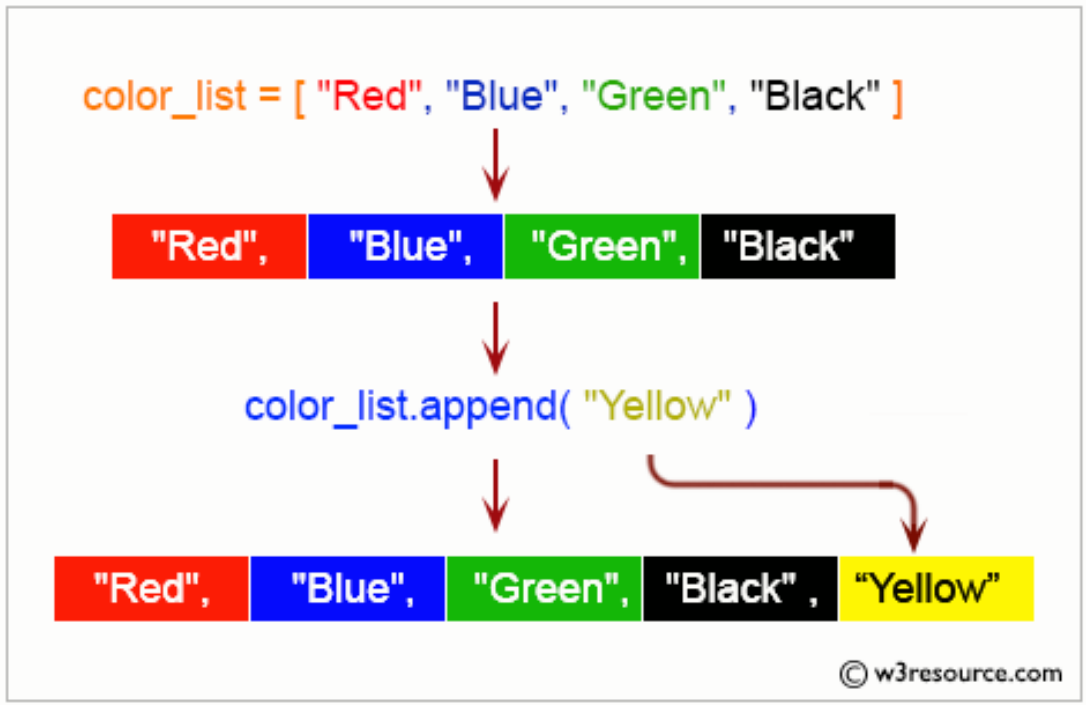
- Exemplo:

```
>>> lista = [6, 5]
>>> lista.append(98)

>>> lista
[6, 5, 98]
```

  - Formalmente, este tipo de função é chamada de `método`.

Listas: append

- Exemplo:



Preenchendo uma lista

- A combinação de uma lista vazia que vai sofrendo "appends" permite ler dados e preencher uma lista com estes dados:

```
1  # lista vazia
2  notas=[]
3
4  n = int(input("Entre com o numero de notas: "))
5
6
7  for i in range(n):
8      dado = float(input("Entre com a nota {}: ".format(i)))
9      notas.append(dado)
10
11 print(notas)
```

  - Como calcular a média das notas?

Preenchendo uma lista

- A combinação de uma lista vazia que vai sofrendo "appends" permite ler dados e preencher uma lista com estes dados:

```
1  # lista vazia
2  notas=[]
3
4  n = int(input("Entre com o numero de notas: "))
5
6
7  for i in range(n):
8      dado = float(input("Entre com a nota {}: ".format(i)))
9      notas.append(dado)
10
11 print(notas)
```

  - Como calcular a média das notas?

Preenchendo uma lista

- A combinação de uma lista vazia que vai sofrendo "appends" permite ler dados e preencher uma lista com estes dados:

```
1  # lista vazia
2  notas=[]
3
4  n = int(input("Entre com o numero de notas: "))
5
6
7  for i in range(n):
8      dado = float(input("Entre com a nota {}: ".format(i)))
9      notas.append(dado)
10
11 print(notas)
```

  - Como calcular a média das notas?

Exemplo

- Resposta:

```
...
# Calculando a média
soma = 0
for i in range(len(notas)):
  soma = soma + notas[i]

media = soma/n
print(format(media, ".1f"))
```

  - Python pussuí muitas funções prontas:

```
# Calcula a média
from statistics import mean

print(format(mean(notas), ".1f"))
```

Exemplo

- Resposta:

```
...
# Calculando a média
soma = 0
for i in range(len(notas)):
  soma = soma + notas[i]

media = soma/n
print(format(media, ".1f"))
```

  - **Python** pussuí muitas funções prontas:

```
# Calcula a média
from statistics import mean

print(format(mean(notas), ".1f"))
```

Listas: `del`

Podemos remover a lista o item da posição especificada.

- Isso é feito com o método `del()`:

```
del(lista[posição])
```

- Exemplo:

```
>>> lista = [40, 99, 30, 10, 40]

>>> del(lista[2])
>>> lista
[40, 99, 10, 40]
```

Listas: `del`

Podemos remover a lista o item da posição especificada.

- Isso é feito com o método `del()`:

```
del(lista[posição])
```

- Exemplo:

```
>>> lista = [40, 99, 30, 10, 40]

>>> del(lista[2])
>>> lista
[40, 99, 10, 40]
```

Listas: `del`

Cuidado:

- Caso a posição não exista em `lista` o método gera uma exceção.
- Exemplo:

```
>>> del(lista[100])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Listas: `clear`

Podemos remover todos os item em uma lista.

- Isso é feito com o método `clear()`:

```
1 lista.clear()
```

- Exemplo:

```
1 >>> lista = ['d','a','b','c','a']
2 >>> lista.clear()
3 >>> lista
4 []
```

Listas: `clear`

Podemos remover todos os item em uma lista.

- Isso é feito com o método `clear()`:

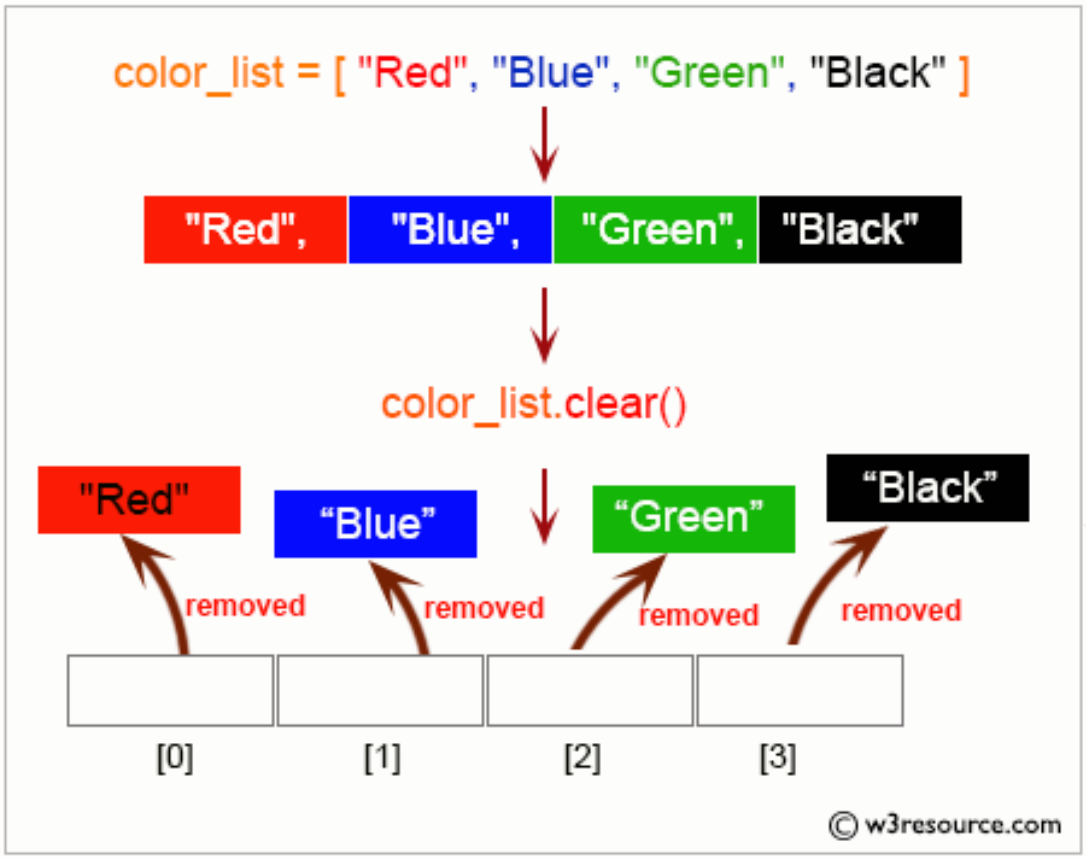
```
1 lista.clear()
```

- Exemplo:

```
1 >>> lista = ['d','a','b','c','a']
2 >>> lista.clear()
3 >>> lista
4 []
```

Listas: `clear`

- Exemplo:



Roteiro

Exemplo 1

Escreva um programa em **Python** que leia do teclado uma sequência de valores inteiros $a_1, a_2, a_3 \ldots$ até que o usuário digite 0. Em seguida imprima essa sequência na ordem inversa.

Exemplo 1

- Primeiro, vamos ler os números na lista `a`:

```
1 a = []
2
3 valor = 1
4 while (valor!=0):
5   valor = int(input())
6   a.append(valor)
```

- Agora, vamos adicionar em uma outra lista `b` os valores de `a` em ordem inversa.

```
1 b = []
2
3 i = len(a)-1
4 while(i>=0):
5   b.append(a[i])
6   i-=1
7
8 #imprime a resposta
9 print(b)
```

Exemplo 1

- Primeiro, vamos ler os números na lista a:

```
1 a = []
2
3 valor = 1
4 while (valor!=0):
5   valor = int(input())
6   a.append(valor)
```

- Agora, vamos adicionar em uma outra lista b os valores de a em ordem inversa.

```
1 b = []
2
3 i = len(a)-1
4 while(i>=0):
5   b.append(a[i])
6   i-=1
7
8 #imprime a resposta
9 print(b)
```

Exemplo 1

- Primeiro, vamos ler os números na lista `a`:

```
a = []

valor = 1
while (valor!=0):
  valor = int(input())
  a.append(valor)
```

- Agora, vamos adicionar em uma outra lista `b` os valores de `a` em ordem inversa.

```
b = []

i = len(a)-1
while(i>=0):
  b.append(a[i])
  i-=1

#imprime a resposta
print(b)
```

Exemplo 1

- Primeiro, vamos ler os números na lista `a`:

```
1 a = []
2
3 valor = 1
4 while (valor!=0):
5   valor = int(input())
6   a.append(valor)
```

- Agora, vamos adicionar em uma outra lista `b` os valores de `a` em ordem inversa.

```
1 b = []
2
3 i = len(a)-1
4 while(i>=0):
5   b.append(a[i])
6   i-=1
7
8 #imprime a resposta
9 print(b)
```

Exemplo 1

- Primeiro, vamos ler os números na lista `a`:

```
1 a = []
2
3 valor = 1
4 while (valor!=0):
5   valor = int(input())
6   a.append(valor)
```

- Agora, vamos adicionar em uma outra lista `b` os valores de `a` em ordem inversa.

```
1 b = []
2
3 i = len(a)-1
4 while(i>=0):
5   b.append(a[i])
6   i-=1
7
8 #imprime a resposta
9 print(b)
```

Exemplo 1

- Primeiro, vamos ler os números na lista `a`:

```
a = []

valor = 1
while (valor!=0):
  valor = int(input())
  a.append(valor)
```

- Agora, vamos adicionar em uma outra lista `b` os valores de `a` em ordem inversa.

```
b = []

i = len(a)-1
while(i>=0):
  b.append(a[i])
  i-=1

#imprime a resposta
print(b)
```

Exemplo 1

- Função pronta do **Python**: `a.reverse()`:

```
1 a.reverse()
2 print(a)
```

Exemplo 2

Escreva um programa em **Python** que leia dois vetores de **dimensão 3**, a e b e calcule o produto escalar destes.

$$a = (a_1, a_2, a_3) \text{ e } b = (b_1, b_2, b_3)$$

$$a \cdot b = \sum_{i=1}^{3} a_i \times b_i$$

Exemplo 2

- Primeiro, temos que ler os dois vetores nas listas `a` e `b`.

```
1 a = []
2 b = []
```

- Qual comando podemos usar?

```
1  for i in range(3):
2      aux = float(input())
3      a.append(aux)
4
5  print('------------------')
6
7  for i in range(3):
8      aux = float(input(''))
9      b.append(aux)
10
11 #calculando o produto interno
12 ...
```

Exemplo 2

- Primeiro, temos que ler os dois vetores nas listas `a` e `b`.

```
1 a = []
2 b = []
```

- Qual comando podemos usar?

```
1  for i in range(3):
2      aux = float(input())
3      a.append(aux)
4
5  print('------------------')
6
7  for i in range(3):
8      aux = float(input(''))
9      b.append(aux)
10
11 #calculando o produto interno
12 ...
```

Exemplo 2

- Primeiro, temos que ler os dois vetores nas listas `a` e `b`.

```
1 a = []
2 b = []
```

- Qual comando podemos usar?

```
1  for i in range(3):
2      aux = float(input())
3      a.append(aux)
4
5  print('------------------')
6
7  for i in range(3):
8      aux = float(input(''))
9      b.append(aux)
10
11 #calculando o produto interno
12 ...
```

Exemplo 2

- Como computar o produto interno dos dois vetores?

$$a = (a_1, a_2, a_3) \text{ e } b = (b_1, b_2, b_3)$$

$$a \cdot b = \sum_{i=1}^{3} a_i \times b_i$$

```
#calculando o produto interno
result = 0
for i in range(3):
    result = result + a[i]*b[i]

print("Produto Interno:", result)
```

Exemplo 2

- Como computar o produto interno dos dois vetores?

$$a = (a_1, a_2, a_3) \text{ e } b = (b_1, b_2, b_3)$$

$$a \cdot b = \sum_{i=1}^{3} a_i \times b_i$$

```
1 #calculando o produto interno
2 result = 0
3 for i in range(3):
4     result = result + a[i]*b[i]
5
6 print("Produto Interno:", result)
```

Exemplo 2

- Como computar o produto interno dos dois vetores?

$$a = (a_1, a_2, a_3) \text{ e } b = (b_1, b_2, b_3)$$

$$a \cdot b = \sum_{i=1}^{3} a_i \times b_i$$

```
1 #calculando o produto interno
2 result = 0
3 for i in range(3):
4     result = result + a[i]*b[i]
5
6 print("Produto Interno:", result)
```

Exemplo 2

- Como computar o produto interno dos dois vetores?

$$a = (a_1, a_2, a_3) \text{ e } b = (b_1, b_2, b_3)$$

$$a \cdot b = \sum_{i=1}^{3} a_i \times b_i$$

```
#calculando o produto interno
result = 0
for i in range(3):
    result = result + a[i]*b[i]

print("Produto Interno:", result)
```

Exemplo 2

- Como computar o produto interno dos dois vetores?

$$a = (a_1, a_2, a_3) \text{ e } b = (b_1, b_2, b_3)$$

$$a \cdot b = \sum_{i=1}^{3} a_i \times b_i$$

```
1 #calculando o produto interno
2 result = 0
3 for i in range(3):
4     result = result + a[i]*b[i]
5
6 print("Produto Interno:", result)
```

Exemplo 3

Escreva um programa em **Python** que leia duas listas com 5 inteiros cada e verifique quais elementos da primeira lista são iguais a algum elemento da segunda lista.

- Exemplo:

```
l1 = [1, 3, 5, 8, 9]
l2 = [5, 9, 10, 1, 0]
```

- Resultado:

```
[1, 5, 9]
```

Exemplo 3

Escreva um programa em **Python** que leia duas listas com 5 inteiros cada e verifique quais elementos da primeira lista são iguais a algum elemento da segunda lista.

- Exemplo:

```
l1 = [1, 3, 5, 8, 9]
l2 = [5, 9, 10, 1, 0]
```

- Resultado:

```
[1, 5, 9]
```

Exemplo 3

- Primeiro, fazemos a leitura das duas listas.

```
l1 = []
l2 = []
```

```
for i in range(5):
    aux = float(input())
    l1.append(aux)

print('------------------')

for i in range(5):
    aux = float(input())
    l2.append(aux)
  ...
```

Exemplo 3

- Primeiro, fazemos a leitura das duas listas.

```
1  l1 = []
2  l2 = []
```

```
1  for i in range(5):
2      aux = float(input())
3      l1.append(aux)
4
5  print('------------------')
6
7  for i in range(5):
8      aux = float(input())
9      l2.append(aux)
10   ...
```

Exemplo 3

- Para cada elemento de `l1` testamos todos os outros elementos de `l2` para saber se são iguais, e adicionamos em `l3`.

```
l3 = []

for i in range(len(l1)):
  for j in range(len(l2)):
    if(l1[i] == l2[j]):
      l3.append(l1[i])
```

Exemplo 3

- Para cada elemento de `l1` testamos todos os outros elementos de `l2` para saber se são iguais, e adicionamos em `l3`.

```
1 l3 = []
2
3 for i in range(len(l1)):
4   for j in range(len(l2)):
5     if(l1[i] == l2[j]):
6       l3.append(l1[i])
```

Exemplo 3

- E como verificar se nenhum elemento foi encontrado??
- Usamos uma variável indicadora (comum) para decidir ao final dos laços encaixados, se nenhum elemento em comum foi encontrado.

```
l3 = []
comum = False   #Assumimos que não hajam elementos comuns

for i in range(len(l1)):
  for j in range(len(l2)):
    if(l1[i] == l2[j]):
      l3.append(l1[i])
      comum=True  # Descobrimos que há elemento comum

if not comum:
  print("Nenhum elemento em comum")
else:
  print(l3)
```

Exemplo 3

- E como verificar se nenhum elemento foi encontrado??
- Usamos uma variável indicadora (`comum`) para decidir ao final dos laços encaixados, se nenhum elemento em comum foi encontrado.

```
l3 = []
comum = False   #Assumimos que não hajam elementos comuns

for i in range(len(l1)):
  for j in range(len(l2)):
    if(l1[i] == l2[j]):
      l3.append(l1[i])
      comum=True  # Descobrimos que há elemento comum

if not comum:
  print("Nenhum elemento em comum")
else:
  print(l3)
```

Exemplo 3

- E como verificar se nenhum elemento foi encontrado??
- Usamos uma variável indicadora (comum) para decidir ao final dos laços encaixados, se nenhum elemento em comum foi encontrado.

```
l3 = []
comum = False    #Assumimos que não hajam elementos comuns

for i in range(len(l1)):
  for j in range(len(l2)):
    if(l1[i] == l2[j]):
      l3.append(l1[i])
      comum=True  # Descobrimos que há elemento comum

if not comum:
  print("Nenhum elemento em comum")
else:
  print(l3)
```

Exemplo 3

- E como verificar se nenhum elemento foi encontrado??
- Usamos uma variável indicadora (comum) para decidir ao final dos laços encaixados, se nenhum elemento em comum foi encontrado.

```
l3 = []
comum = False   #Assumimos que não hajam elementos comuns

for i in range(len(l1)):
  for j in range(len(l2)):
    if(l1[i] == l2[j]):
      l3.append(l1[i])
      comum=True  # Descobrimos que há elemento comum

if not comum:
  print("Nenhum elemento em comum")
else:
  print(l3)
```

Exemplo 3

- E como verificar se nenhum elemento foi encontrado??
- Usamos uma variável indicadora (comum) para decidir ao final dos laços encaixados, se nenhum elemento em comum foi encontrado.

```
1  l3 = []
2  comum = False   #Assumimos que não hajam elementos comuns
3
4  for i in range(len(l1)):
5    for j in range(len(l2)):
6      if(l1[i] == l2[j]):
7        l3.append(l1[i])
8        comum=True  # Descobrimos que há elemento comum
9
10 if not comum:
11   print("Nenhum elemento em comum")
12 else:
13   print(l3)
```

Exemplo 3

- E como verificar se nenhum elemento foi encontrado??
- Usamos uma variável indicadora (comum) para decidir ao final dos laços encaixados, se nenhum elemento em comum foi encontrado.

```
1  l3 = []
2  comum = False   #Assumimos que não hajam elementos comuns
3
4  for i in range(len(l1)):
5    for j in range(len(l2)):
6      if(l1[i] == l2[j]):
7        l3.append(l1[i])
8        comum=True  # Descobrimos que há elemento comum
9
10 if not comum:
11   print("Nenhum elemento em comum")
12 else:
13   print(l3)
```

Fim

Dúvidas?

Leitura complementar

Leitura complementar:

1. `https://panda.ime.usp.br/cc110/static/cc110/09-listas.html`
2. `https://panda.ime.usp.br/pensepy/static/pensepy/09-Listas/listas.html`

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da Universidade Estadual de Campinas.