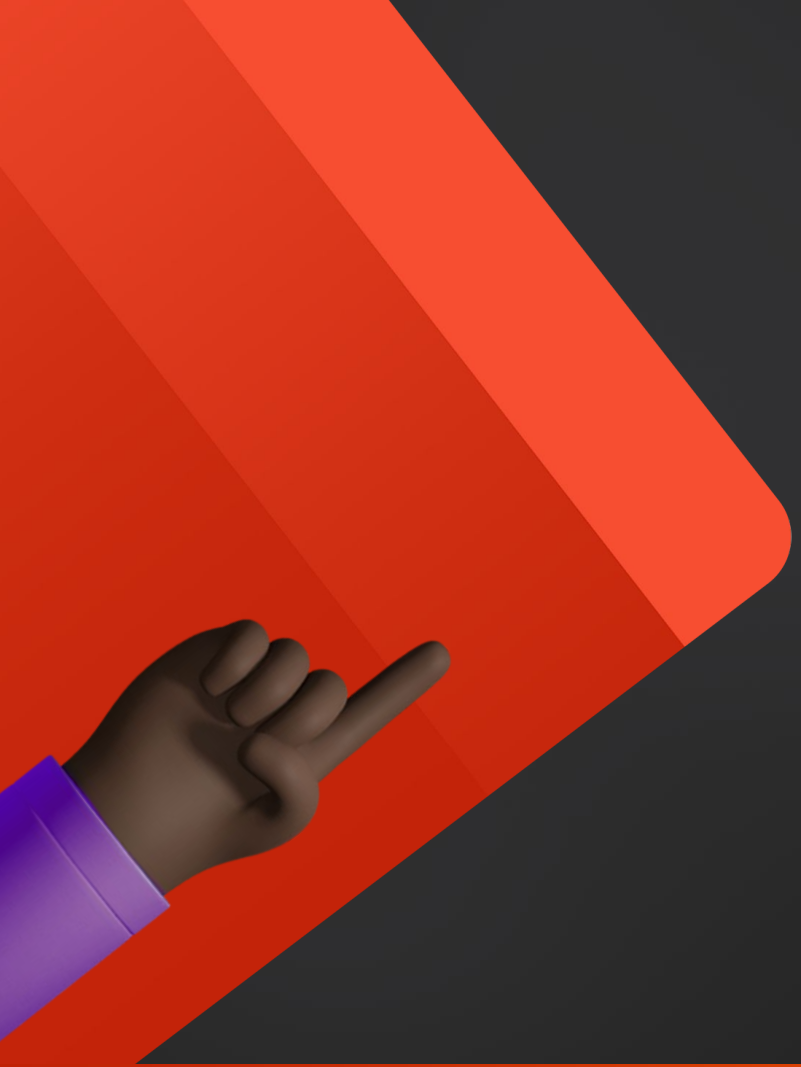
DESTRAVE O CONHECIMENTO

/mesttra



Nasser Boan

Tuplas x Listas

Tuplas x Listas

- Ambas as estruturas de dados são utilizadas para guardar tipos heterogêneos de dados.
- Ambas as estruturas são ordenadas, ou seja, elas mantêm a ordem dos dados que foram colocados nela.
- Ambas são objetos iteráveis, podemos acessar seus valores de forma iterativa.
- Ambas podem ser indexadas utilizando colchetes.

```
[61]  1  import sys
      2  a_list = list()
      3  a_tuple = tuple()
      4  a_list = [1,2,3,4,5]
      5  a_tuple = (1,2,3,4,5)
      6  print(f'TAMANHO DE UMA LISTA {sys.getsizeof(a_list)} BYTES')
      7  print(f'TAMANHO DE UMA LISTA {sys.getsizeof(a_tuple)} BYTES')

    TAMANHO DE UMA LISTA 112 BYTES
    TAMANHO DE UMA LISTA 96 BYTES
```

Tuplas x Listas

```
[67]  1  import sys, platform
      2  import time
      3
      4  start_time = time.time()
      5  b_list = list(range(10000000))
      6  end_time = time.time()
      7  print("Instantiation time for LIST:", end_time - start_time)
      8
      9  start_time = time.time()
     10  b_tuple = tuple(range(10000000))
     11  end_time = time.time()
     12  print("Instantiation time for TUPLE:", end_time - start_time)
     13
     14  start_time = time.time()
     15  for item in b_list:
     16    aa = b_list[20000]
     17  end_time = time.time()
     18  print("Lookup time for LIST: ", end_time - start_time)
     19
     20  start_time = time.time()
     21  for item in b_tuple:
     22    aa = b_tuple[20000]
     23  end_time = time.time()
     24  print("Lookup time for TUPLE: ", end_time - start_time)
     25

Instantiation time for LIST: 0.37953782081604004
Instantiation time for TUPLE: 0.39007043838500977
Lookup time for LIST:  1.0335209369659424
Lookup time for TUPLE:  0.9247586727142334
```

Dicionários

- Os “dicionários” são outra estrutura de dados em python que também são conhecidas como “memória associativas” ou “vetores associativos”;
- Diferentemente das listas e tuplas, os dicionários são indexados utilizando chaves (keys).
- Construímos um dicionário utilizando chaves {}. Os dicionários são mutáveis e iteráveis.

```
[74]  1  meu_dict = {}
      2  type(meu_dict)

      dict
```

Dicionários

- Podemos adicionar um conjunto de “chave:valor” utilizando a sintaxe abaixo.

```
[76]  1  meu_dict[ chave ] = [ valores ]
      2  meu_dict
```

```
{'clientes': ['Maria', 'João', 'Gilberto', 'Gabriela', 'Marcelo']}
```

Dicionários

- Podemos também colocar outros dicionários como valores de uma chave.

```
[82]  1  meu_dict['endereços'] = {
      2      'maria':'águas claras',
      3      'joão':'asa norte',
      4      'gilberto':'asa sul',
      5      'gabriela':'samambaia',
      6      'marcelo':'planaltina'
      7  }
      8
      9  meu_dict
```

```
{'clientes': ['Maria', 'João', 'Gilberto', 'Gabriela', 'Marcelo'],
 'endereços': {'gabriela': 'samambaia',
  'gilberto': 'asa sul',
  'joão': 'asa norte',
  'marcelo': 'planaltina',
  'maria': 'águas claras'}}
```

Dicionários

- O valor associado a uma chave pode ser alterado através de uma reatribuição.

```
[84]  1  meu_dict['valor_total_faturado'] = 50000
      2  meu_dict
```

```
{'clientes': ['Maria', 'João', 'Gilberto', 'Gabriela', 'Marcelo'],
 'endereços': {'gabriela': 'samambaia',
  'gilberto': 'asa sul',
  'joão': 'asa norte',
  'marcelo': 'planaltina',
  'maria': 'águas claras'},
 'valor_total_faturado': 50000}
```

```
[85]  1  meu_dict['valor_total_faturado'] = 122658
      2  meu_dict
```

```
{'clientes': ['Maria', 'João', 'Gilberto', 'Gabriela', 'Marcelo'],
 'endereços': {'gabriela': 'samambaia',
  'gilberto': 'asa sul',
  'joão': 'asa norte',
  'marcelo': 'planaltina',
  'maria': 'águas claras'},
 'valor_total_faturado': 122658}
```

Dicionários

- A indexação de um dicionário é através de suas chaves.

```
[86]  1  meu_dict['clientes']

['Maria', 'João', 'Gilberto', 'Gabriela', 'Marcelo']
```

Dicionários

- Podemos ainda fazer uma indexação mais complexa, indexando o resultado da primeira indexação.

Dicionários

- Indexando um valor associado a uma chave podemos acessar os métodos desse valor.

```
[88]  1  meu_dict['clientes'].append('Nasser')
      2  meu_dict
```

```
{'clientes': ['Maria', 'João', 'Gilberto', 'Gabriela', 'Marcelo', 'Nasser'],
 'endereços': {'gabriela': 'samambaia',
  'gilberto': 'asa sul',
  'joão': 'asa norte',
  'marcelo': 'planaltina',
  'maria': 'águas claras'},
 'valor_total_faturado': 122658}
```

Dicionários

- Indexando um valor associado a uma chave podemos acessar os métodos desse valor.

```
[91]  1  meu_dict['endereços']['nasser'] = 'águas claras'
      2  meu_dict
```

```
{'clientes': ['Maria', 'João', 'Gilberto', 'Gabriela', 'Marcelo', 'Nasser'],
 'endereços': {'gabriela': 'samambaia',
  'gilberto': 'asa sul',
  'joão': 'asa norte',
  'marcelo': 'planaltina',
  'maria': 'águas claras',
  'nasser': 'águas claras'},
 'valor_total_faturado': 122650}
```


1 meu_dict.
clear builtin_function_or_method
copy
fromkeys
get
items
keys
pop
popitem
setdefault
update
values
[ ] 1
[ ] 1
[ ] 1
[ ] 1

Dicionários

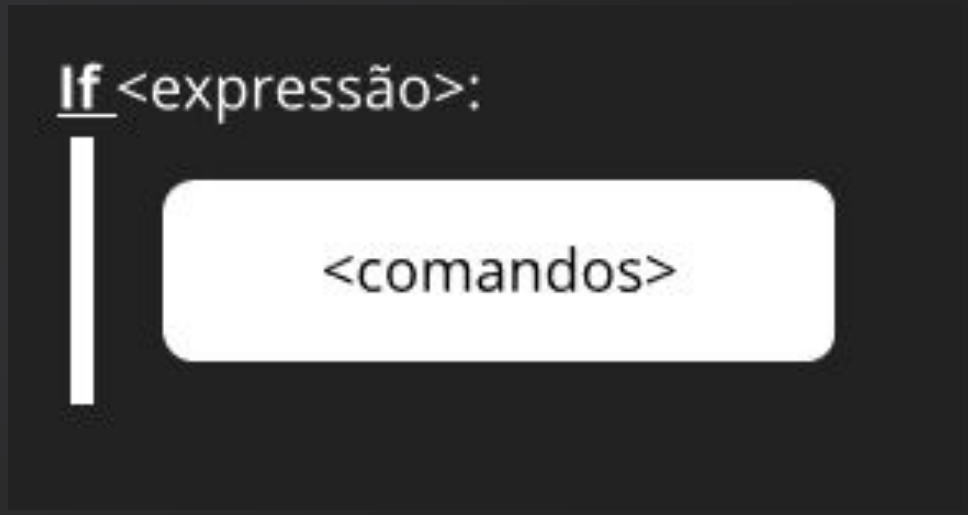
Método	Resultado
.keys()	Retorna todas as chaves de um dicionário
.values()	Retorna todos os valores de um dicionário.
.pop('chave_de_exemplo')	Elimina 'chave_de_exemplo' e retorna seu valor associado.
.items()	Retorna as chaves e valores de um dicionário.
.get('chave_de_exemplo')	Retorna o valor de 'chave_de_exemplo'.
.clear()	Limpa todo o dicionário

Estruturas Condicionais

Estruturas Condicionais

- Na vida real tomamos a partir da análise de algumas condições.
- Exemplo: Se eu tiver pelo menos R$ 50,00 então irei ao cinema.
- Essa é uma expressão lógica “Tenho R$ 50,00?” isso pode ser respondido com “sim” ou “não”.
- Em linguagem de programação uma parte do código pode ser executada ou não de acordo com o resultado dessas expressões lógicas, a estrutura que torna isso real chamamos de condicional.

if / else



if / else

- Monte um algoritmo que avalie o valor da variável “pagamento” ...
- ... se pagamento for maior que 1000 printe a frase “comprar relógio”
- Teste seu algoritmo com os valores 10, 999, 999.99, 1000, 1001 e “bacon”.

if / else

- Monte um algoritmo que avalie o valor da variável “pagamento” ...
- ... se pagamento for maior que 1000 printe a frase “comprar relógio” ...
- ... se pagamento for menor que 1000 printe a frase “compre mês que vem”.
- Teste seu algoritmo com os valores 10, 999, 999.99, 1000, 1001 e “bacon”.

if / else

if / else

1. Monte um algoritmo que avalie o valor da variável “clima” ...
2. ... se “clima” for ensolarado então avalie a variável “vento” ...
    a) ... se “vento” for maior ou igual a 2.0 printe “Não jogar, muito vento”.
    b) ... se “vento” for menor que 2.0 printe “Ótimo dia para um jogo”.
3. ... para todos os outros casos de “clima” então avalie a variável “chance_de_chuva” ...
    a) ... se “chance_de_chuva” for maior ou igual a 80 printe “Não jogar, pode chover”.
    b) ... se “chance_de_chuva” for menor que 80 printe “Vamos jogar e aquecer o dia”.

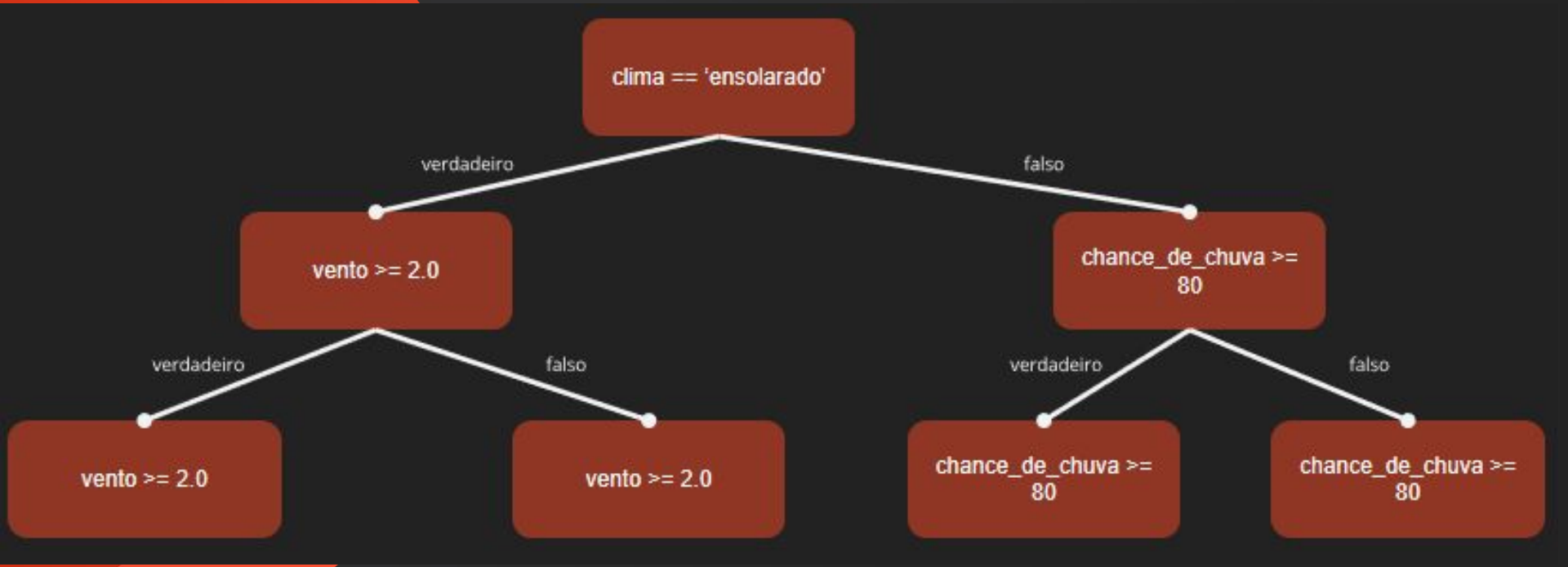
if / else

Teste seu algoritmo com os valores:

clima = “ensolarado”, “nublado”, “chuvoso”;

vento = 1.2, 3.0, 1.999

chance_de_chuva = 10, 80, 50


clima == 'ensolarado'
verdadeiro
falso
vento >= 2.0
chance_de_chuva >= 80
verdadeiro
falso
verdadeiro
falso
vento >= 2.0
vento >= 2.0
chance_de_chuva >= 80
chance_de_chuva >= 80

if / elif / else


If <expressão>:
<comandos>
elif <expressão>:
<comandos>
else :
<comandos>

if / elif / else

1. Monte um algoritmo que avalie o valor da variável “bpm” ...
2. ... se bpm for maior ou igual a 40 e menor ou igual a 60 printe “Largo”
3. ... se bpm for maior que 60 e menor ou igual a 80 printe “Adagio”
4. ... se bpm for maior ou igual a 140 e menor ou igual a 200 printe “Presto”
5. ... para todas as outras situações printe “Troppo Lento”.

Avalie seu algoritmo usando bpm = 20, 45, 80, 60, 110, 113, 118 e 200

Estruturas de Repetição

Estruturas de Repetição

- As estruturas de repetição (ou loops) são bastante utilizadas quando queremos executar um bloco de código até atingirmos uma condição específica, que pode ser o final de um objeto (lista, por exemplo).

- O loop ‘for’ percorre todo um objeto iterável e executa um bloco de código para cada um dos valores associados a uma variável determinada no próprio loop.

for

for <item> **in** <objeto iterável>:

    <comandos>

```
[107]  1  ls = [1,2,3,4,5,6,7,8,9,10]
       2
       3  for valor in ls:
       4      print(valor)
```

```
1
2
3
4
5
```

for

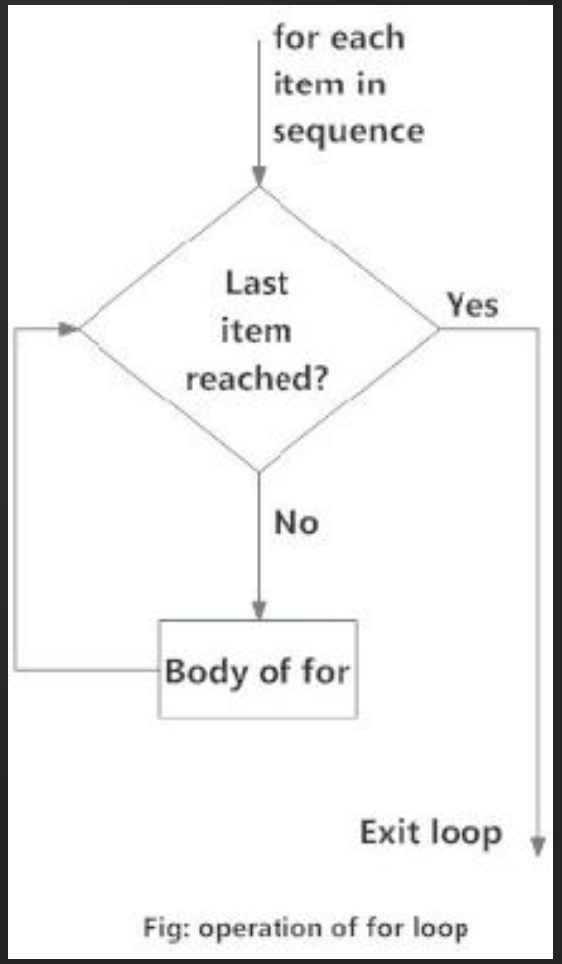


Fig: operation of for loop

for

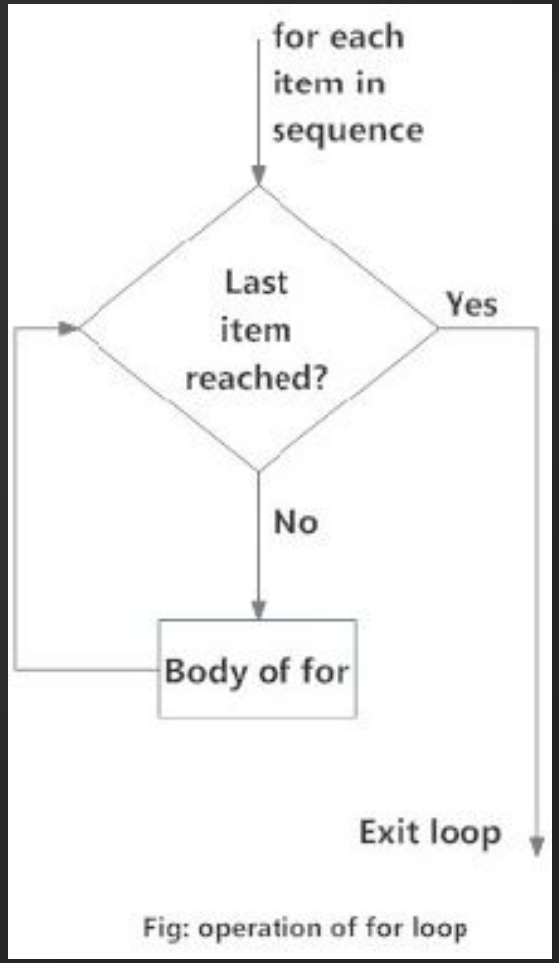


Fig: operation of for loop

```
ls = [1,2,3,4,5,6,7,8,9,10]
```

```
1
8
27
64
125
216
343
512
729
1000
```

for

- Podemos manipular o processo de iteração de um for utilizando uma condição no bloco de código.

```
[109]  1  ## for com if
       2
       3  for valor in ls:
       4      if valor % 2 == 0:
       5          print('Valor Múltiplo de 2')
       6      else:
       7          print(valor)
```

```
1
Valor Múltiplo de 2
3
Valor Múltiplo de 2
5
Valor Múltiplo de 2
7
Valor Múltiplo de 2
9
Valor Múltiplo de 2
```

for

- Podemos utilizar a palavra ‘break’ para interromper um processo de iteração sobre o objeto iterável.

```
[110]  1  ## for com break
       2
       3  new_ls = list(range(1,500))
       4
       5  soma = 0
       6
       7  for valor in new_ls:
       8
       9    if soma > 100:
      10      break
      11
      12    if valor % 3.5 == 0:
      13      print(f"O valor {valor} é válido!")
      14      print(f"SOMA antes", soma)
      15      soma += valor
      16      print(f"SOMA depois {soma}\n----------")
```

```
O valor 7 é válido!
SOMA antes 0
SOMA depois 7
----------
O valor 14 é válido!
SOMA antes 7
SOMA depois 21
----------
O valor 21 é válido!
SOMA antes 21
SOMA depois 42
----------
O valor 28 é válido!
SOMA antes 42
SOMA depois 70
----------
O valor 35 é válido!
SOMA antes 70
SOMA depois 105
```

for

- Podemos utilizar o ‘else’ para executar um bloco de código ao final das iterações.

```
[113]  1  ## for com else
       2  cidades = ["Brasília", "Bananal", "São Paulo", "São Luís"]
       3
       4  for cidade in cidades:
       5      print(cidade)
       6
       7  else:
       8      print("Acabaram as cidades")
```

```
Brasília
Bananal
São Paulo
São Luís
Acabaram as cidades
```

- Compreensão do for em uma lista:

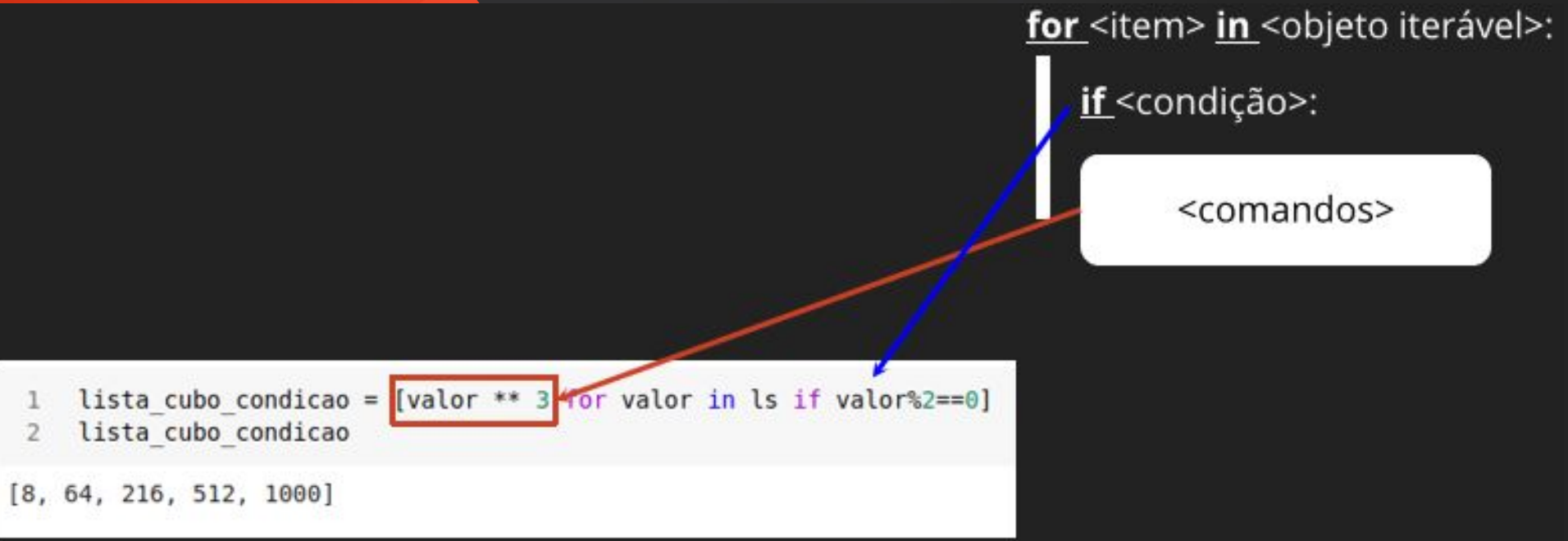
```
for <item> in <objeto iterável>:
    <comandos>
```

```
[55]  1  ## compreensão de lista
      2
      3  lista_cubo = [valor**3 for valor in ls]
      4  lista_cubo
```

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

- Compreensão do for em uma lista:

- Compreensão do for em uma lista:

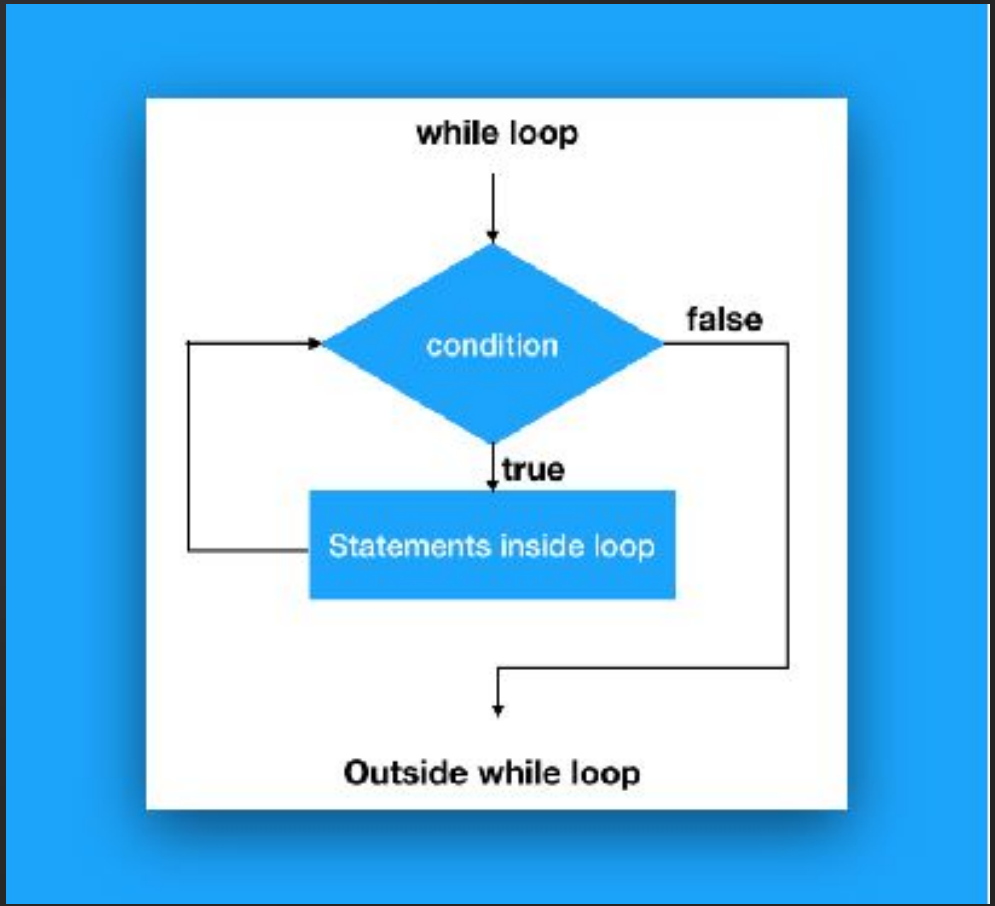


while

- Outra estrutura de repetição é o ‘while’;
- Ele executa o bloco de código inúmeras vezes até que sua condição criadora se torne falsa;
- ou que ele seja parado com ‘break’.

```
enquanto <expressão ou condição> faça:
    <lista de comandos>
fimenquanto
```

while


while loop
condition
false
true
Statements inside loop
Outside while loop

while

```
[18]  1  i = 10
      2
      3  while i <= 100:
      4      print(f"O valor de 'i' é {i}.")
      5      i += 10
```

```
O valor de 'i' é 10.
O valor de 'i' é 20.
O valor de 'i' é 30.
O valor de 'i' é 40.
O valor de 'i' é 50.
O valor de 'i' é 60.
O valor de 'i' é 70.
O valor de 'i' é 80.
O valor de 'i' é 90.
O valor de 'i' é 100.
```

while

- Podemos antecipadamente parar a execução de um loop ‘while’ com a palavra ‘break’, nesse caso a condição pode continuar verdadeira, mas ele é interrompido com o while.

```
[115]  1  i = 10
       2
       3  while i <= 100:
       4    print(f"O valor de 'i' é {i}.")
       5
       6    if i == 60:
       7      print('Cheguei a 60, parando o loop.')
       8      break
       9
      10    i += 10
```

```
O valor de 'i' é 10.
O valor de 'i' é 20.
O valor de 'i' é 30.
O valor de 'i' é 40.
O valor de 'i' é 50.
O valor de 'i' é 60.
Cheguei a 60, parando o loop.
```

while

- Podemos utilizar a palavra reservada ‘else’ para executar um bloco de código quando a condição de um ‘while’ for avaliada como falsa.

```
[120] 1  clientes = 0
      2  valor_recebido = 0
      3
      4  while valor_recebido < 100:
      5    print('Venda mais!')
      6    clientes += 1
      7    valor_recebido = clientes * 4.99
      8  else:
      9    print('Meta alcançada!!')
     10    print('QTD DE CLIENTES',clientes)
```

```
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Meta alcançada!!
QTD DE CLIENTES 21
```

while

```
primeira_lista = [1,2,3,4,5]
segunda_lista = [10,20,30,40,50]

for i in primeira_lista:
  for j in segunda_lista:
    print(i,j)
```

```
1 10
1 20
1 30
1 40
1 50
2 10
2 20
2 30
2 40
2 50
3 10
3 20
3 30
3 40
3 50
4 10
4 20
4 30
4 40
4 50
5 10
5 20
5 30
5 40
5 50
```

```
[126]
i = 0

ls = []

while i < 10:
  for valor in primeira_lista:
    ls.append(valor)

  i += 1
len(ls)
```

```
50
```

while

- Podemos utilizar a palavra reservada ‘else’ para executar um bloco de código quando a condição de um ‘while’ for avaliada como falsa.

```
[120] 1  clientes = 0
      2  valor_recebido = 0
      3
      4  while valor_recebido < 100:
      5    print('Venda mais!')
      6    clientes += 1
      7    valor_recebido = clientes * 4.99
      8  else:
      9    print('Meta alcançada!!')
     10    print('QTD DE CLIENTES',clientes)
```

```
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Venda mais!
Meta alcançada!!
QTD DE CLIENTES 21
```

while

```
[126]  1  i = 0
       2
       3  ls = []
       4
       5  while i < 10:
       6      for valor in primeira_lista:
       7          ls.append(valor)
       8
       9      i += 1
      10  len(ls)
```

```
50
```

```
1  primeira_lista = [1,2,3,4,5]
2  segunda_lista = [10,20,30,40,50]
3
4  for i in primeira_lista:
5      for j in segunda_lista:
6          print(i,j)
```

```
1 10
1 20
1 30
1 40
1 50
2 10
2 20
2 30
2 40
2 50
3 10
3 20
3 30
3 40
3 50
4 10
4 20
4 30
4 40
4 50
5 10
5 20
5 30
5 40
5 50
```

Obrigada

/mesttra mesttra.com