



LISTAS

Listas são um dos 4 tipos de dados embutidos no Python usados para armazenar coleções de dados, os outros 3 são **Tuple** , **Set** e **Dictionary** , todos com diferentes características e uso.

As listas são usadas para armazenar vários itens em uma única variável. As listas são criadas usando **colchetes**.

Exemplo

```
lista = ["maçã", "banana", "cereja"]  
print(lista)  
  
#Resultado:  
['maçã', 'banana', 'cereja']
```

Lista de itens

Os itens da lista são ordenados, alteráveis e permitem valores duplicados. Os itens da lista são indexados, o primeiro item tem índice `[0]` , o segundo item tem índice `[1]` e assim por diante.

Ordem

As listas estão ordenadas, ou seja, significa que os itens têm uma ordem definida e essa ordem não mudará. Se você adicionar novos itens a uma lista, os novos itens serão colocados no final da lista.



Nota: Existem alguns **métodos de lista** que mudam a ordem, mas em geral: a ordem dos itens não muda.

Mutável

A lista pode ser alterada, o que significa que podemos alterar, adicionar e remover itens de uma lista após sua criação.

Valores duplicados

Como as listas são indexadas, elas podem ter itens com o mesmo valor:

Exemplo

```
lista = ["maçã", "banana", "cereja", "maçã", "cereja"]
print(lista)

#Resultado:
['maçã', 'banana', 'cereja', 'maçã', 'cereja']
```

Comprimento da lista

Para determinar quantos itens uma lista possui, use a `len()` função:

Exemplo

```
lista = ["maçã", "banana", "cereja"]
print(len(lista))

#Resultado:
3
```

Itens de lista - tipos de dados

Os itens da lista podem ser de qualquer tipo de dados:

```
lista1 = ["maçã", "banana", "cereja"]
lista2 = [1, 5, 7, 9, 3]
lista3 = [True, False, False]
print(lista1, lista2, lista3)

#Resultado:
['maçã', 'banana', 'cereja'] [1, 5, 7, 9, 3] [True, False, False]
```

Uma lista pode conter diferentes tipos de dados:

```
lista1 = ["abc", 34, True, 40, "homem"]
print(lista1)

#Resultado:
['abc', 34, True, 40, 'homem']
```

Type()

Da perspectiva do Python, as listas são definidas como objetos com o tipo de dados 'lista':

```
lista = ["maçã", "banana", "cereja"]
print(type(lista))

#Resultado:
<class 'list'>
```

O construtor list ()

Também é possível usar o construtor list () ao criar uma nova lista.

Usando o construtor `list()` para fazer uma lista:

```
lista = list(("maçã", "banana", "cereja")) # note os parênteses duplos
print(lista)

#Resultado:
['maçã', 'banana', 'cereja']
```

Itens de acesso

Os itens da lista são indexados e você pode acessá-los referindo-se ao número do índice:

Exemplo

```
lista = ["maçã", "banana", "cereja"]
print(lista[1]) #0 primeiro item tem índice 0.

#Resultado:
banana
```

Indexação Negativa

Indexação negativa significa começar do fim, o índice '-1' refere-se ao último item e o '-2' refere-se ao penúltimo item e assim sucessivamente.

Exemplo

```
lista = ["maçã", "banana", "cereja"]
print(lista[-1]) #Note que contagem negativa começa com -1

#Resultado:
cereja
```

Gama de Índices

Você pode especificar um intervalo de índices, especificando onde começar e onde terminar o intervalo. Ao especificar um intervalo, o valor de retorno será uma nova lista com os itens especificados.

Exemplo

```
lista = ["maçã", "banana", "cereja", "laranja", "kiwi", "melão", "manga"]
print(lista[2:5])

#Resultado:
['cereja', 'laranja', 'kiwi']
```



Nota: A pesquisa começará no índice 2 (incluído) e terminará no índice 5 (não incluído). Lembre-se de que o primeiro item tem índice 0.

Ao omitir o valor inicial, o intervalo começará no primeiro item:

Exemplo

Este exemplo retorna os itens do início até, mas 4(não incluído):

```
lista = ["maçã", "banana", "cereja", "laranja", "kiwi", "melão", "manga"]
print(lista[:4])

#Resultado:
['maçã', 'banana', 'cereja', 'laranja']
```

Ao omitir o valor final, o intervalo irá para o final da lista:

Exemplo

Este exemplo retorna os itens da "cereja" até o final:

```
lista = ["maçã", "banana", "cereja", "laranja", "kiwi", "melão", "manga"]
print(lista[2:])

#Resultado:
['cereja', 'laranja', 'kiwi', 'melão', 'manga']
```

Faixa de índices negativos

Especifique índices negativos se você deseja iniciar a pesquisa a partir do final da lista:

Exemplo

Este exemplo retorna os itens de "orange" (-4) a, mas NÃO incluindo "mango" (-1):

```
lista = ["maçã", "banana", "cereja", "laranja", "kiwi", "melão", "manga"]
print(lista[-4:-1])

#Resultado:
['laranja', 'kiwi', 'melão']
#Note que o último dígito não é incluído, mas sim o anterior a ele.
```

Verifique se o item existe

Para determinar se um item especificado está presente em uma lista, use a palavra-chave `in`:

Exemplo

```
lista = ["maçã", "banana", "cereja"]
if "maçã" in lista:
    print("Sim, 'maçã' está na lista")

#Resultado:
Sim, 'maçã' está na lista
```



Nota: O espaço na terceira linha é devido ao fato de ser um novo bloco, se faz necessário por conta da indentação da linguagem.

Alterar o valor do item

Para alterar o valor de um item específico, consulte o número do índice:

Exemplo

Mude o segundo item:

```
lista = ["maçã", "banana", "cereja"]
lista[1] = "maracujá"
print(lista)
```

```
#Resultado:
['maçã', 'maracujá', 'cereja']
```

Alterar um intervalo de valores de item

Para alterar o valor dos itens dentro de um intervalo específico, defina uma lista com os novos valores e consulte o intervalo de números de índice onde deseja inserir os novos valores:

Exemplo

Altere os valores "banana" e "cereja" com os valores "groselha preta" e "melancia":

```
lista = ["maçã", "banana", "cereja", "laranja", "kiwi", "manga"]
lista[1:3] = ["maracujá", "melancia"]
print(lista)
```

```
#Resultado:
['maçã', 'maracujá', 'melancia', 'laranja', 'kiwi', 'manga']
```

Se você inserir *mais* itens do que substituir, os novos itens serão inseridos onde você especificou e os itens restantes serão movidos de acordo:

Exemplo

Altere o segundo valor substituindo-o por *dois* novos valores:

```
lista = ["maçã", "banana", "cereja"]
lista[1:2] = ["maracujá", "melancia"]
print(lista)
```

```
#Resultado:
['maçã', 'maracujá', 'melancia', 'cereja']
#Note que o último índice não é incluído, portanto apenas o 1 mudará e nesse caso serão
#adicionados 2 valores.
```



Nota: O comprimento da lista mudará quando o número de itens inseridos não corresponder ao número de itens substituídos.

Se você inserir *menos* itens do que substituir, os novos itens serão inseridos onde você especificou e os itens restantes serão movidos de acordo:

Exemplo

Altere o segundo e o terceiro valor, substituindo-o por *um* valor:

```
lista = ["maçã", "banana", "cereja"]
lista[1:3] = ["melancia"]
print(lista)
```

```
#Resultado:
['maçã', 'melancia']
```

Inserir itens

Para inserir um novo item da lista, sem substituir nenhum dos valores existentes, podemos usar o método `insert()`.

Exemplo

```
lista = ["maçã", "banana", "cereja"]
lista.insert(2, "melancia")
print(lista)
```

```
#Resultado:
['maçã', 'banana', 'melancia', 'cereja']
#A lista agora contém 4 itens.
```

Itens Anexos

Para adicionar um item ao final da lista, use o método `append()` :

Exemplo

```
lista = ["maçã", "banana", "cereja"]
lista.append("laranja")
print(lista)
```

```
#Resultado:  
['maçã', 'banana', 'cereja', 'laranja']
```

Extend List

Para anexar elementos de *outra lista* à lista atual, use o método `extend()`.

Exemplo

```
lista = ["maçã", "banana", "cereja"]  
tropical = ["manga", "abacaxi", "mamão"]  
lista.extend(tropical)  
print(lista)  
  
#Resultado:  
['maçã', 'banana', 'cereja', 'manga', 'abacaxi', 'mamão']  
#Note que os itens são adicionados ao final da lista.
```

Adicionar qualquer iterável

O método `extend()` não precisa anexar *listas*, você pode adicionar qualquer objeto iterável (tuples, sets, dictionaries etc.).

Exemplo

```
lista = ["maçã", "banana", "cereja"]  
atuple = ("kiwi", "laranja")  
lista.extend(atuple)  
print(lista)  
  
#Resultado:  
['maçã', 'banana', 'cereja', 'kiwi', 'laranja']
```

Remover Item Especificado

O método `remove()` remove o item especificado.

Exemplo

```
lista = ["maçã", "banana", "cereja"]  
lista.remove("banana")  
print(lista)
```



```
#Resultado:  
['maçã', 'cereja']
```

Remover Índice Especificado

O método `pop()` remove o índice especificado.

Exemplo

```
lista = ["maçã", "banana", "cereja"]  
lista.pop(1)  
print(lista)  
  
#Resultado:  
['maçã', 'cereja']
```

Se você não especificar o índice, o `pop()` método remove o último item.

```
lista = ["maçã", "banana", "cereja"]  
lista.pop()  
print(lista)  
  
#Resultado:  
['maçã', 'banana']
```

A palavra-chave `del` também remove o índice especificado:

Exemplo

```
lista = ["maçã", "banana", "cereja"]  
del lista[0]  
print(lista)  
  
#Resultado:  
['banana', 'cereja']
```

A `del` palavra-chave também pode excluir a lista completamente.

```
lista = ["maçã", "banana", "cereja"]  
del lista  
print(lista)  
  
#O resultado será um erro devido ao fato de lista não estar definida, pois foi criada e  
em seguida deletada.
```

Limpe a lista

O método `clear()` esvazia a lista. A lista ainda permanece, mas não tem conteúdo.

Exemplo

```
lista = ["maçã", "banana", "cereja"]
lista.clear()
print(lista)

#Resultado:
[]
```

Loop através de uma lista

Você pode percorrer os itens da lista usando um `for` loop:

Exemplo

```
lista = ["maçã", "banana", "cereja"]
for x in lista:
    print(x)

#Resultado:
maçã
banana
cereja
#Veja que a variável x é definida na lista
```

Loop através de números de índice

Você também pode percorrer os itens da lista, referindo-se ao seu número de índice.

Use as funções `range()` e `len()` para criar um iterável adequado.

Exemplo

```
lista = ["maçã", "banana", "cereja"]
for i in range(len(lista)):
    print(lista[i])

#Resultado:
maçã
```

```
banana
cereja
#0 iterável criado no exemplo acima é [0,1,2]
```

Usando um Loop While

Você pode percorrer os itens da lista usando um loop `while`.

Use a função `len()` para determinar o comprimento da lista, então comece em 0 e faça um loop pelos itens da lista consultando seus índices.

Lembre-se de aumentar o índice em 1 após cada iteração.

Exemplo

Imprima todos os itens, usando um loop `while` para percorrer todos os números do índice.

```
lista = ["maçã", "banana", "cereja"]
i = 0
while i < len(lista):
    print(lista[i])
    i = i + 1

#Resultado:
maçã
banana
cereja
```

Looping usando compreensão de lista

Compreensão de lista oferece a sintaxe mais curta para percorrer listas:

Exemplo

Um pequeno `for` loop manual que imprimirá todos os itens em uma lista:

```
lista = ["maçã", "banana", "cereja"]
[print(x) for x in lista]

#Resultado:
maçã
banana
cereja
```

Compreensão de lista

A compreensão de listas oferece uma sintaxe mais curta quando você deseja criar uma nova lista com base nos valores de uma lista existente.

A sintaxe

O valor de retorno é uma nova lista, deixando a lista antiga inalterada.

```
#lista = ["a", "b", "c"]  
#novalista = [expressão for item in iterável if condição == True]
```

Exemplo

```
frutas = ["maçã", "banana", "cereja", "kiwi", "manga"]  
novalista = []  
for x in frutas:  
    if "a" in x:  
        novalista.append(x)  
print(novalista)  
  
#Resultado:  
['maçã', 'banana', 'manga']
```

Com a compreensão de lista, você pode fazer tudo isso com apenas uma linha de código:

Exemplo

```
frutas = ["maçã", "banana", "cereja", "kiwi", "manga"]  
novalista = [x for x in frutas if "a" in x]  
print(novalista)  
  
#Resultado:  
['maçã', 'banana', 'cereja', 'manga']
```

Condição

A *condição* é como um filtro que aceita apenas os itens avaliados para `True`.

```
frutas = ["maçã", "banana", "cereja", "kiwi", "manga"]
novalista = [x for x in frutas if x != "maçã"]
print(novalista)
```

A condição `if x != "maçã"` retornará `True` para todos os elementos exceto "maçã", fazendo com que a nova lista contenha todas as frutas, exceto "maçã".

A *condição* é opcional e pode ser omitida:

```
frutas = ["maçã", "banana", "cereja", "kiwi", "manga"]
novalista = [x for x in frutas]
print(novalista)

#
```

Iterável

O *iterável* pode ser qualquer objeto iterável, como uma lista, tupla, conjunto etc.

Você pode usar a função `range()` para criar um iterável:

```
lista = [x for x in range(10)]
print(lista)

#Resultado:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Mesmo exemplo, mas com uma condição: Aceite apenas números menores que 5.

```
lista = [x for x in range(10) if x < 5]
print(lista)

#Resultado:
[0, 1, 2, 3, 4]
```

Expressão

A *expressão* é o item atual na iteração, mas também é o resultado, que você pode manipular antes que termine como um item de lista na nova lista:

Defina os valores na nova lista para maiúsculas:

```
frutas = ["maçã", "banana", "cereja", "kiwi", "manga"]
lista = [x.upper() for x in frutas]
print(lista)
```

```
#Resultado:
['MAÇÃ', 'BANANA', 'CEREJA', 'KIWI', 'MANGA']
```

Você pode definir o resultado como quiser: Defina todos os valores na nova lista como 'olá':

```
frutas = ["maçã", "banana", "cereja", "kiwi", "manga"]
lista = ['olá' for x in frutas]
print(lista)
```

```
#Resultado:
['olá', 'olá', 'olá', 'olá', 'olá']
```

A *expressão* também pode conter condições, não como um filtro, mas como uma forma de manipular o resultado:

Exemplo

```
frutas = ["maçã", "banana", "cereja", "kiwi", "manga"]
lista = [x if x != "banana" else "laranja" for x in frutas]
print(lista)
```

```
#Resultado:
['maçã', 'laranja', 'cereja', 'kiwi', 'manga']
```

Classificar lista alfanumericamente

Os objetos de lista têm um método `sort()` que classifica a lista alfanumericamente, de forma ascendente, por padrão:

Exemplo

Classifique a lista em ordem alfabética:

```
lista = ["laranja", "manga", "kiwi", "abacaxi", "banana"]
lista2 = [100, 50, 65, 82, 23]
lista.sort()
lista2.sort()
print(lista)
print(lista2)
```

```
#Resultado:  
['abacaxi', 'banana', 'kiwi', 'laranja', 'manga']  
[23, 50, 65, 82, 100]
```

Classificar em ordem decrescente

Para classificar em ordem decrescente, use o argumento de palavra-chave `reverse`
`= True`:

Exemplo

Classifique a lista em ordem decrescente:

```
lista = ["laranja", "manga", "kiwi", "abacaxi", "banana"]  
lista2 = [100, 50, 65, 82, 23]  
lista.sort(reverse = True)  
lista2.sort(reverse = True)  
print(lista)  
print(lista2)  
  
#Resultado:  
['manga', 'laranja', 'kiwi', 'banana', 'abacaxi']  
[100, 82, 65, 50, 23]
```

Função de classificação personalizada

Você também pode personalizar sua própria função usando o argumento de palavra-chave `key = function`.

A função retornará um número que será usado para classificar a lista (o número mais baixo primeiro):

Exemplo

Classifique a lista com base na proximidade do número de 50:

```
def func(n):  
    return abs(n - 50)  
lista = [100, 50, 65, 82, 23]  
lista.sort(key = func)  
print(lista)  
  
#Resultado:  
[50, 65, 23, 82, 100]
```

Classificação que não diferencia maiúsculas de minúsculas

Por padrão, o método `sort()` diferencia maiúsculas de minúsculas, resultando em todas as letras maiúsculas sendo classificadas antes das letras minúsculas:

Exemplo

A classificação com distinção entre maiúsculas e minúsculas pode dar um resultado inesperado:

```
lista = ["banana", "Laranja", "Kiwi", "cereja"]
lista.sort()
print(lista)

#Resultado:
['Kiwi', 'Laranja', 'banana', 'cereja']
```

Felizmente, podemos usar funções integradas como funções-chave ao classificar uma lista. Portanto, se você quiser uma função de classificação que não diferencia maiúsculas de minúsculas, use `str.lower` como uma função-chave:

Exemplo

Faça uma classificação da lista sem distinção entre maiúsculas e minúsculas:

```
lista = ["banana", "Laranja", "Kiwi", "cereja"]
lista.sort(key = str.lower)
print(lista)

#Resultado:
['banana', 'cereja', 'Kiwi', 'Laranja']
```

Ordem reversa

O método `reverse()` inverte a ordem de classificação atual dos elementos.

Exemplo

```
lista = ["banana", "laranja", "kiwi", "cereja"]
lista.reverse()
print(lista)
```


Copiar uma lista

Existem maneiras de fazer uma cópia de uma lista.

Utilizando o método `copy()`:

```
lista = ["maçã", "banana", "cereja"]
novalista = lista.copy()
print(novalista)
```

Utilizando o método integrado `list()`:

```
lista = ["maçã", "banana", "cherry"]
novalista = list(lista)
print(lista)
```

Junção de Listas

Existem várias maneiras de juntar, ou concatenar, duas ou mais listas em Python.

1. Utilizando o operador `+`:

```
lista1 = ["a", "b", "c"]
lista2 = [1, 2, 3]
lista3 = lista1 + lista2
print(lista3)
```

```
#Resultado:
['a', 'b', 'c', 1, 2, 3]
```

2. Anexando todos os itens da lista2 à lista1, um por um:

```
lista1 = ["a", "b", "c"]
lista2 = [1, 2, 3]
for x in lista2:
    lista1.append(x)
print(lista1)
```

```
#Resultado:
['a', 'b', 'c', 1, 2, 3]
```

3. Utilizando o método `extend()` , cujo objetivo é adicionar elementos de uma lista a outra lista:

```
lista1 = ["a", "b" , "c"]
lista2 = [1, 2, 3]
lista1.extend(lista2)
print(lista1)

#Resultado:
['a', 'b', 'c', 1, 2, 3]
```

Note que os 3 exemplos apresentados resultam na mesma saída, alcançada de maneira diferente. A construção do código depende de alguns fatores e como programador, terá de tomar as decisões.

Coleções Python (matrizes)

Existem quatro tipos de coleta de dados na linguagem de programação Python:

- **Lista** - é uma coleção ordenada e mutável. Permite membros duplicados.
- **Tupla** - é uma coleção ordenada e imutável. Permite membros duplicados.
- **Set** - é uma coleção não ordenada, imutável * e não indexada. Nenhum membro duplicado.
- **Dictionary** - é uma coleção ordenada ** e mutável. Nenhum membro duplicado.

* Os itens definidos não podem ser alterados, mas você pode remover e / ou adicionar itens sempre que desejar.

** A partir do Python versão 3.7, os dicionários são solicitados. No Python 3.6 e anteriores, os dictionary's são ordenados.

Ao escolher um tipo de coleção, é útil entender as propriedades desse tipo. Escolher o tipo certo para um determinado conjunto de dados pode significar retenção de significado e, pode significar um aumento na eficiência ou segurança.

Métodos de lista

Python tem um conjunto de métodos embutidos que você pode usar em listas.

Método	Descrição
--------	-----------

insert()	Adiciona um elemento na posição especificada
append()	Adiciona um elemento no final da lista
extend()	Adiciona os elementos de uma lista (ou qualquer iterável), ao final da

lista atual

remove()	Remove o item com o valor especificado
clear()	Remove todos os elementos da lista
pop()	Remove o elemento na posição especificada
copy()	Retorna uma cópia da lista
count()	Retorna o número de elementos com o valor especificado
index()	Retorna o índice do primeiro elemento com o valor especificado
reverse()	Inverte a ordem da lista
sort()	Classifica a lista