# When to use Python's "list" constructor

Contents

[When to use Python's "list" constructor 1](#_Toc149654682)

[Какво прави? list 1](#_Toc149654683)

[Не използвай за създаване на празни списъци list() 1](#_Toc149654684)

[Четирите употреби на list 2](#_Toc149654685)

[Shallow copying lists - Shallow списъци за копиране 2](#_Toc149654686)

[Turning lazy iterables into lists - Превръщане на мързеливите iterables в списъци 2](#_Toc149654687)

[Превръщане на всеки iterable в списък 3](#_Toc149654688)

[Using as a factory function list 3](#_Toc149654689)

[При сливане на iterables използвайте вместо \*list() **When merging iterables use \* instead of list()** 5](#_Toc149654690)

[Embrace list, but don't overuse it 6](#_Toc149654691)

Кога трябва да използвате вградената функция на Python, за да създадете списък?

И кога трябва да използвате квадратни скоби (...), за да създадете нов списък вместо това? **list** [ ]

Конструкторът е една от вградените функции на Python, която е, странно, често недостатъчно използвана и прекомерно използвана. Нека да разгледаме кога трябва да използвате конструктора и кога не. listlist

## Какво прави? list

Има два начина да използвате вградената функция (или "callable", ако предпочитате, тъй като технически това е клас). list

Функцията приема един аргумент, който трябва да бъде iterable.list

>>> a\_new\_list = list(my\_iterable)

Функцията ще премине през дадената iterable и ще направи нов списък. list

Подобно на , , , и много конструкторни функции в Python, също могат да бъдат извикани без никакви аргументи: strintdictlist

>>> **list**()

[ ]

Когато не са дадени аргументи, връща празен списък. List

## Не използвай за създаване на празни списъци list()

Създаването на празен списък е много често срещана операция в Python.

Като цяло избягвам да използвам за създаване на нов списък. list()

Препоръчвам да използвате квадратни скоби (синтаксиса "списък литерално"), за да направите нов празен списък:

colors = [ ]

Не препоръчвам използването на вградената функция :list

colors = **list**()

Защо не?

Дзен на Python казва:

"Трябва да има един, и за предпочитане само един, очевиден начин да го направя".

Най-често се виждат квадратни скоби, използвани за създаване на празен списък, така че виждам това като "един очевиден начин". Обикновено, ако има специален синтаксис за нещо, този начин става "единственият очевиден начин" в Python.

Създаването на празен списък с помощта на буквалния синтаксис на списъка е по-бързо и бих казал, че е по-идиоматично и по-четливо.

## Четирите употреби на list

Ако обикновено не се използва за създаване на празен списък, кога се използва? list

Има 4 сценария за употреба, които препоръчвам да използвате вградената функция: list

1. Copying (shallowly) one list to another - Копиране на един списък в друг
2. Convert a lazy iterable into a list - Конвертиране на мързелив iterable в списък
3. Turning *any* iterable into a list - Превръщане на всеки iterable в списък
4. To make a list-creating factory function- Създаване на фабрична функция за създаване на списък

Нека разгледаме всеки един от тях поотделно.

## Shallow copying lists - Shallow списъци за копиране

Използването на функцията е предпочитаният от мен начин да направя shallow копие на списък. list

Предпочитам това:

new\_list = **list**(old\_list)

Пред това:

new\_list = old\_list.copy()

*Защо?*

Две причини:

1. Този първи подход работи на всеки iterable, а не само на списъци (повечето iterables не разполагат с метод) copy
2. Когато използвате , е по-очевидно, че полученият обект е списък, вместо някаква друга структура от данни. list(...)

## Turning lazy iterables into lists - Превръщане на мързеливите iterables в списъци

Много операции в Python връщат [iterables](https://www.pythonmorsels.com/iterable/). Конструкторът е удобен за превръщане на мързеливи lazy iterables в конкретни списъци: list

lines\_with\_numbers = list(enumerate(lines))

За разлика от итераторите, списъците няма да изчезнат, след като бъдат преплетени само веднъж.

Имайте предвид, че в известен смисъл това е просто още един вкус на използване за плитко копиране: копираме всеки елемент в lazy iterable в нов списък. list

## Превръщане на всеки iterable в списък

Обичам [разбирането на списъка](https://www.pythonmorsels.com/what-are-list-comprehensions/) на Python, но те не винаги са правилният инструмент за задачата.

Аз много предпочитам това:

rows = list(csv\_reader)

Пред това:

rows = [row for row in csv\_reader]

Благодарение на [duck-typing](https://www.pythonmorsels.com/duck-typing/), всеки обект, който може да бъде завъртян с loop или с разбиране, също може да бъде примка с вградения конструктор: iterable е iterable . forlist

Обсъждам тази конкретна прекомерна употреба на разбирания повече в този скрийнкаст за [прекомерното използване на разбирания](https://www.pythonmorsels.com/when-should-you-not-use-list-comprehension/) и в [тази статия](https://treyhunner.com/2019/03/abusing-and-overusing-list-comprehensions-in-python/#Using_comprehensions_when_a_more_specific_tool_exists).

Може би по-противоречиво е, че аз също предпочитам това:

lines = list(my\_file)

Пред това:

lines = my\_file.readlines()

Със , това е ясно, че ние правим loop над файл и съхраняване на всеки ред в нов списък (ако приемем, че знаете, че loop над файл осигурява линии)./ With , it's clear that we're looping over a file and storing each line in a new list (assuming you know that looping over a file provides lines). / Със това не е толкова очевидно, че създаваме списък с линии. Всъщност много нови Pythonistas биха могли да предположат, че ще върнат итератор (list(my\_file)my\_file.readlines()readlines [много вградени функции връщат итератори](https://www.pythonmorsels.com/generators-are-iterators/#helper-functions-return-an-iterator) в края на краищата).

Този случай на употреба е наистина само вариация на последните две, но това е достатъчно важна концепция, която си струва да се повтори по 3 различни начина! list

## Using as a factory function list

Тези първи три причини за използване са свързани с вземането на стар iterable и вземане на нов списък от него. Това е най-удобната причина да се използва и много често виждам хората да забравят, че това е чудесен инструмент точно за тази цел! Това обаче не е единствената употреба на списъка. listlistlist

Функцията също е удобен начин да се направи " factory function ", която създава нов празен списък. list

Класът [defaultdict](https://docs.python.org/3/library/collections.html#collections.defaultdict) от модула може да се използва за създаване на речник със стойности по подразбиране, но изисква нещо като фабрична функция: тя се нуждае от [извикване](https://www.pythonmorsels.com/callables/), подадено в него, което ще създаде тези стойности по подразбиране. collections

Например, да кажем, че правим функция, която приема низ и връща речник, групиращ думи по техните дължини:

**def** group\_by\_lengths(text):

lengths = { }

**for** word in text.split( ):

length = **len**(word)

**if** length **in** lengths:

lengths[length].**append**(word)

**else**:

lengths[length] = [word]

Вместо ръчно да подбираме всеки нов ключ към списък с един елемент, бихме могли да направим обект, подобен на речник, където всяка стойност по подразбиране е празен списък, като използваме с като фабрична функция: collections.defaultdictlist

**from** collections **import** defaultdict

**def** group\_by\_lengths(text):

lengths = defaultdict(**list**)

**for** word **in** text.**split**( ):

length = **len**(word)

lengths[length].**append**(word)

Обектът, на който подаваме, трябва да бъде извикваем, който може да бъде извикан с 0 аргумента и ще върне стойност по подразбиране за всеки нов ключ. Вградената функция просто се случва да бъде една такава извикваща. defaultdictlist

Извикването връща празен списък, така че всеки път, когато се осъществи достъп до липсващ ключ, той ще завърши със стойност на празен списък. list

>>> **from** collections **import** defaultdict

>>> weekdays = defaultdict(**list**)

>>> weekdays["Wednesday"].**append**("February 15, 2023")

>>> weekdays

defaultdict(<class 'list'>, {'Wednesday': ['February 15, 2023']})

>>> weekdays["Tuesday"]

[ ]

Така че, макар да не препоръчвам да се обръщате директно, за да направите празен списък, тази функция може да бъде доста удобна, когато се използва като фабрична функция. list()list

## При сливане на iterables използвайте вместо \*list() **When merging iterables use \* instead of list()**

Понякога може да се окажете с няколко итерации, които искате да обедините в един списък.

Ако и двете ви iterables са списъци, можете просто да ги съедините:

values = list1 + list2

Но ако някоя от тях може да не е списък, вероятно ще конвертирате всеки от тях в списък и след това ще ги съедините:

values = list(args) + list(kwargs.values( ))

Това създава 3 списъка, 2 от които временни. Правим списък от , правим списък от и след това ги свързваме, за да направим третия списък, на който след това присвояваме. **argskwargs.values()values**

Вместо да преобразувам в списъци и да обединявам, обикновено предпочитам да използвам оператора заедно със синтаксиса на литерала на списъка, за да разопаковам всеки итерируем в един нов списък:\*

values = [\*args, \*kwargs.values()]

Това директно разопакова всеки iterable в нов списък, който избягва създаването на временни списъци.

Също така намирам за малко по-лесно да се чете с един поглед, отколкото , въпреки че новите програмисти на Python вероятно го намират за по-малко четлив за четене с един поглед, така че помислете предпазливо за предпочитания от вас подход.

[\*a, \*b( )] - list(a) + list(b)

Можете да намерите [Повече подробности за това използване тук](https://www.pythonmorsels.com/unpacking-iterables-iterables/)\* и [общи съвети за използване на \* и \*\* в Python](https://treyhunner.com/2018/10/asterisks-in-python-what-they-are-and-how-to-use-them/) тук.

Обърнете внимание, че не препоръчвам да използвате за преобразуване на един iterable в друг:\*

lines = [\*my\_file]

Това работи, но намирам това за по-четливо (почти сигурно е по-очевидно за нов програмист на Python):

lines = list(my\_file)

Конструкторът ми се струва "единственият очевиден начин" за превръщане на всеки iterable в списък. Това е почти единствената цел. list

Вместо това операторът е предназначен за разопаковане на един итерируем в друг.\*

## Embrace list, but don't overuse it

Вграденият конструктор на Python е супер удобен. Той е кратък и целта му е доста ясна: вземете iterable и го превърнете в списък.list

Ако имам генератор, файлов обект или генеричен iterable, чийто точен тип не е задължително да знам и бих искал да го конвертирам в списък, функцията е това, към което винаги се стремя. list

Независимо дали копирате списък, превръщате всеки общ iterable в списък, или имате нужда от фабрична функция, която връща празен списък, запомнете конструктора! Но се уверете, че не прекалявате:

Буквалният синтаксис на списъка на Python е много предпочитан, когато правите празни списъци. listlist