Python list vs. array – when to use?

Asked 16 years, 2 months ago Modified 2 years, 4 months ago

Viewed 398k times

465

If you are creating a 1d array, you can implement it as a list, or else use the 'array' module in the standard library. I have always used lists for 1d arrays.

What is the reason or circumstance where I would want to use the array module instead?

Is it for performance and memory optimization, or am I missing something obvious?

python arrays list

Share

Improve this question

Follow

edited Aug 17, 2022 at 8:18

LeopardShark

4,368 ● 4 ● 20 ● 36

asked Oct 6, 2008 at 20:17

Corey Goldberg

60.5k ● 29 ● 133 ● 144

11 Answers

Sorted by: Highest score (default)

Basically, Python lists are very flexible and can hold completely heterogeneous, arbitrary data, and they can be appended to very efficiently, in amortized constant time. If you need to shrink and grow your list time-efficiently and without hassle, they are the way to go. But they use **a lot more space than C arrays**, in part because each item in the list requires the construction of an individual Python object, even for data that could be represented with simple C types (e.g. `float` or `uint64_t`).

The `array.array` type, on the other hand, is just a thin wrapper on C arrays. It can hold only homogeneous data (that is to say, all of the same type) and so it uses only `sizeof(one object) * length` bytes of memory. Mostly, you should use it when you need to expose a C array to an extension or a system call (for example, `ioctl` or `fctnl`).

`array.array` is also a reasonable way to represent a **mutable** string in Python 2.x (`array('B', bytes)`). However, Python 2.6+ and 3.x offer a mutable *byte* string as `bytearray`.

However, if you want to do **math** on a homogeneous array of numeric data, then you're much better off using NumPy, which can automatically vectorize operations on complex multi-dimensional arrays.

To make a long story short: `array.array` is useful when you need a homogeneous C array of data for

reasons *other than doing math*.

Share Improve this answer Follow

edited Aug 21, 2020 at 2:30

answered Oct 6, 2008 at 23:11

Dan Lenski
79.6k ● 13 ● 84 ● 127

15 Does numpy.ndarray have the same memory footprint as array.array? – Gordon Bean Oct 28, 2014 at 22:49

11 @Gordon, it should be very similar in the case of a large, contiguous array: they both will require `sizeof(element)` ×(number of elements) bytes, plus a small fixed header for overhead. However, ndarray has some advanced options for dealing with discontiguous and sparse arrays, and I think some pluggable strategies for allocating memory for large arrays... some of these advanced features will make it user *less* memory, while others will improve performance by using *more* memory. – Dan Lenski Oct 29, 2014 at 4:57

1 One can lookup the i'th element of an array in a constant time, whereas in linked list, it takes order 'n' in the worst case. What's the lookup time of i'th element in a python list? – Nithish Inpursuit Ofhappiness Jul 19, 2017 at 2:16

14 @NithishInpursuitOfhappiness, a Python list is not a linked list. It's represented internally as an array and has the same time complexity characteristics as Java's ArrayList. Thus, getting and setting the i'th element of a Python list takes **constant time**. Appending an element to a Python list takes **amortized constant time** because the array size is doubled when it runs out of space. Inserting an element into or removing from the middle of a Python list takes **O(n) time**

because elements need to be shifted. For reference, see: wiki.python.org/moin/TimeComplexity – geofflee Nov 5, 2017 at 1:17

1 @Timo, that's exactly what the example in the answer already shows. – Dan Lenski Dec 24, 2020 at 0:52

77

For almost all cases the normal list is the right choice. The arrays module is more like a thin wrapper over C arrays, which give you kind of strongly typed containers (see docs), with access to more C-like types such as signed/unsigned short or double, which are not part of the built-in types. I'd say use the arrays module only if you really need it, in all other cases stick with lists.

Share Improve this answer

Follow

edited Mar 12, 2013 at 5:58

Etaoin

8,694 ● 2 ● 29 ● 44

answered Oct 6, 2008 at 20:24

André

13.3k ● 3 ● 35 ● 45

3 Possible, never used it really though, but would be interesting to run some micro benchmarks. – André Oct 6, 2008 at 20:43

14 Actually, I did a quick test - I timed summing a list with 100M entries and the same test with the corresponding array, and the list was actually about 10% faster. – Moe Oct 6, 2008 at 20:45

44 Lists are faster, because operations on array "raw" data need to continuously create and destroy python objects

when reading from or writing to the array. – tzot Oct 6, 2008 at 21:37

9 @Moe, as I pointed out in my answer above, Python's built-in `array` is **not meant for doing math**. If you try NumPy's `ndarray` for summing an array of 10^8 numbers, it will completely blow `list` away. @tzot has the right idea about why the built-in `array` is slow for math. – Dan Lenski Aug 28, 2014 at 6:46

3 I just tested it, numpy is 86.6x faster on my machine. – Mark Sep 23, 2016 at 12:34

60

The array module is kind of one of those things that you probably don't have a need for if you don't know why you would use it (and take note that I'm not trying to say that in a condescending manner!). Most of the time, the array module is used to interface with C code. To give you a more direct answer to your question about performance:

Arrays are more efficient than lists for some uses. If you need to allocate an array that you KNOW will not change, then arrays can be faster and use less memory. GvR has an optimization anecdote in which the array module comes out to be the winner (long read, but worth it).

On the other hand, part of the reason why lists eat up more memory than arrays is because python will allocate a few extra elements when all allocated elements get used. This means that appending items to lists is faster. So if you plan on adding items, a list is the way to go.

TL;DR I'd only use an array if you had an exceptional optimization need or you need to interface with C code (and can't use pyrex).

Share Improve this answer

Follow

edited Jul 25, 2015 at 15:26

Dennis van der Schagt

2,414 ● 2 ● 28 ● 34

answered Oct 7, 2008 at 14:00

Jason Baker

198k ● 138 ● 382 ● 520

1 +1 for concrete example and mentioning speed benefit. The top answer made me wonder, "Is there a time-memory tradeoff?" and "Is there any use for this that's not a very esoteric low-memory case?" – leewz Feb 5, 2014 at 20:48

can you pls exlain "allocate a few extra elements when all allocated elements get used" .what do you mean by used and where will it allocate – Nirbhay Garg Jul 24, 2020 at 13:15

21

> It's a trade off !

pros of each one :

list

- flexible
- can be heterogeneous

array (ex: numpy array)

- array of uniform values
- homogeneous
- compact (in size)
- efficient (functionality and speed)
- convenient

Share Improve this answer Follow

edited Mar 15, 2017 at 10:10

answered Jan 30, 2017 at 20:18

Mohammad Mahdi KouchakYazdi

1,386 ● 2 ● 23 ● 29

10 the question is reffering to array module in python; not numpy arrays. They do not have lots of pros except size efficiency. They are not faster. – NONONONONO Jul 11, 2019 at 2:06

1 A comment above states that arrays are slower, because when trying to retrieve their elements, they recreate Python objects. – Yaroslav Nikitenko Feb 14, 2022 at 7:59

15

My understanding is that arrays are stored more efficiently (i.e. as contiguous blocks of memory vs. pointers to Python objects), but I am not aware of any performance benefit. Additionally, with arrays you must store primitives of the same type, whereas lists can store anything.

Share Improve this answer

Follow

answered Oct 6, 2008 at 20:22

Ben Hoffstein

103k ● 8 ● 106 ● 121

9

This answer will sum up almost all the queries about when to use List and Array:

1. The main difference between these two data types is the operations you can perform on them. For example, you can divide an array by 3 and it will divide each element of array by 3. Same can not be done with the list.
2. The list is the part of python's syntax so it doesn't need to be declared whereas you have to declare the array before using it.
3. You can store values of different data-types in a list (heterogeneous), whereas in Array you can only store values of only the same data-type (homogeneous).
4. Arrays being rich in functionalities and fast, it is widely used for arithmetic operations and for storing a large amount of data - compared to list.

5. Arrays take less memory compared to lists.

Share Improve this answer

Follow

edited Sep 10, 2019 at 13:51

answered Sep 10, 2019 at 13:26

Dipen Gajjar

1,448 ● 16 ● 24

8

The standard library arrays are useful for binary I/O, such as translating a list of ints to a string to write to, say, a wave file. That said, as many have already noted, if you're going to do any real work then you should consider using NumPy.

Share Improve this answer

Follow

answered Oct 7, 2008 at 13:47

giltay

2,025 ● 1 ● 12 ● 5

7

With regard to performance, here are some numbers comparing python lists, arrays and numpy arrays (all with Python 3.7 on a 2017 Macbook Pro). The end result is that the python list is fastest for these operations.

```
# Python list with append()
np.mean(timeit.repeat(setup="a = []", stmt="a.append(1
repeat=5000)) * 1000
# 0.054 +/- 0.025 msec

# Python array with append()
np.mean(timeit.repeat(setup="import array; a = array.a
```

```
stmt="a.append(1.0)", number=1000, repeat=5000)) * 100
# 0.104 +/- 0.025 msec

# Numpy array with append()
np.mean(timeit.repeat(setup="import numpy as np; a = n
stmt="np.append(a, [1.0])", number=1000, repeat=5000))
# 5.183 +/- 0.950 msec

# Python list using +=
np.mean(timeit.repeat(setup="a = []", stmt="a += [1.0]
repeat=5000)) * 1000
# 0.062 +/- 0.021 msec

# Python array using +=
np.mean(timeit.repeat(setup="import array; a = array.a
array.array('f', [1.0]) ", number=1000, repeat=5000))
# 0.289 +/- 0.043 msec

# Python list using extend()
np.mean(timeit.repeat(setup="a = []", stmt="a.extend([
repeat=5000)) * 1000
# 0.083 +/- 0.020 msec

# Python array using extend()
np.mean(timeit.repeat(setup="import array; a = array.a
stmt="a.extend([1.0]) ", number=1000, repeat=5000)) *
# 0.169 +/- 0.034
```

Share Improve this answer

Follow

answered Jun 16, 2020 at 1:38

Hephaestus

5,073 ● 6 ● 43 ● 57

6

If you're going to be using arrays, consider the numpy or scipy packages, which give you arrays with a lot more flexibility.

answered Oct 6, 2008 at 20:30

Share Improve this answer

Follow

Alex Coventry
70.7k ● 5 ● 39 ● 40

5

Array can only be used for specific types, whereas lists can be used for any object.

Arrays can also only data of one type, whereas a list can have entries of various object types.

Arrays are also more efficient for some numerical computation.

Share Improve this answer

Follow

answered Oct 6, 2008 at 20:25

Hortitude
14k ● 16 ● 60 ● 72

4 The builtin python arrays are not performance-wise efficient, only memory-wise. – tzot Oct 6, 2008 at 21:40

There ARE instances where arrays are more efficient in terms of processing. See my post below: stackoverflow.com/questions/176011/… – Jason Baker Oct 7, 2008 at 14:03

1

An important difference between numpy array and list is that array slices are views on the original array. This means that the data is not copied, and any modifications to the view will be reflected in the source array.

Share Improve this answer

answered Jul 17, 2018 at 0:46

Follow

vivek
583 ● 8 ● 17

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.