Python Sets vs Lists

Asked 10 years, 5 months ago Active 1 month ago Viewed 152k times

193

In Python, which data structure is more efficient/speedy? Assuming that order is not important to me and I would be checking for duplicates anyway, is a Python set slower than a Python list?

python list performance data-structures set Edit tags

56

edited Aug 12 '19 at 5:59
user11768920

asked May 14 '10 at 0:55
Mantas Vidutis
14.6k 20 72 90

10 Answers

Active | Oldest | Votes

236

It depends on what you are intending to do with it.

Sets are significantly faster when it comes to determining if an object is present in the set (as in `x in s`), but are slower than lists when it comes to iterating over their contents.

You can use the timeit module to see which is faster for your situation.

edited Sep 29 '16 at 10:25
smerlin
5,790 3 29 52

answered May 14 '10 at 1:04
Michael Aaron Safyan
85.2k 13 127 192

6 For your point: "Sets are significantly faster ", what is the underlying implementation that makes it faster? – overexchange Jul 13 '15 at 13:50

Scripting languages like to hide the underlying implementations, but this apparent simplicity is not always a good thing, you do need some 'data structure' awareness when you design a piece of software. – Christophe Roussy Aug 2 '16 at 14:40

5 Set is not significantly slower than list while iterating. – omerfarukdogan Jan 23 '18 at 9:47

43 Sets and lists both have linear time iteration. To say that one is "slower" than the other is misguided and has confused new programmers who read this answer. – habnabit Mar 16 '18 at 1:34

@habnabit if you are saying that they both have linear time iteration. Does this mean they have the same iteration time? What is the difference then? – Mohammed Noureldin May 5 at 14:37

8

`Set` wins due to near instant 'contains' checks: https://en.wikipedia.org/wiki/Hash_table

List implementation: usually an array, low level close to the metal good for iteration and **random access by element index**.

Set implementation: https://en.wikipedia.org/wiki/Hash_table, it does not iterate on a list, but finds the element by computing a **hash** from the key, so it depends on the nature of the key elements and the hash function. Similar to what is used for dict. I suspect `list` could be faster if you have very few elements (< 5), the larger element count the better the `set` will perform for a contains check. It is also fast for element addition and removal. Also always keep in mind that building a set has a cost !

NOTE: If the `list` is already sorted, searching the `list` could be quite fast on small lists, but with more data a `set` is faster for contains checks.

edited Aug 13 at 8:22

answered Aug 2 '16 at 14:35
Christophe Roussy
12.7k 2 69 73

8 Close to the metal? What does that even mean in the context of Python? How is a list closer to the metal than a set? – roganjosh Jun 21 '18 at 18:32

@roganjosh, python still runs on a machine and some implementations like list as 'array' are closer to what the hardware is good at: stackoverflow.com/questions/176011/…, but it always depends on what you want to achieve, it is good to know a bit about the implementations, not just the abstractions. – Christophe Roussy May 14 '19 at 16:46

0

Sets are faster, morover you get more functions with sets, such as lets say you have two sets :

```
set1 = {"Harry Potter", "James Bond", "Iron Man"}
set2 = {"Captain America", "Black Widow", "Hulk", "Harry Potter", "James Bond"}
```

We can easily join two sets:

```
set3 = set1.union(set2)
```

Find out what is common in both:

```
set3 = set1.intersection(set2)
```

Find out what is different in both:

```
set3 = set1.difference(set2)
```

And much more! Just try them out, they are fun! Moreover if you have to work on the different values within 2 list or common values within 2 lists, I prefer to convert your lists to sets, and many programmers do in that way. Hope it helps you :-)

answered May 22 at 7:24
Shakhyar Gogoi
26 2

3

I was interested in the results when checking, with CPython, if a value is one of a small number of literals. `set` wins in Python 3 vs `tuple`, `list` and `or`:

```
from timeit import timeit

def in_test1():
  for i in range(1000):
    if i in (314, 628):
      pass

def in_test2():
  for i in range(1000):
    if i in [314, 628]:
      pass

def in_test3():
  for i in range(1000):
    if i in {314, 628}:
```

```
      pass

def in_test4():
  for i in range(1000):
    if i == 314 or i == 628:
      pass

print("tuple")
print(timeit("in_test1()", setup="from __main__ import in_test1", number=100000))
print("list")
print(timeit("in_test2()", setup="from __main__ import in_test2", number=100000))
print("set")
print(timeit("in_test3()", setup="from __main__ import in_test3", number=100000))
print("or")
print(timeit("in_test4()", setup="from __main__ import in_test4", number=100000))
```

Output:

```
tuple
4.735646052286029
list
4.7308746771886945
set
3.5755991376936436
or
4.687681658193469
```

For 3 to 5 literals, `set` still wins by a wide margin, and `or` becomes the slowest.

In Python 2, `set` is always the slowest. `or` is the fastest for 2 to 3 literals, and `tuple` and `list` are faster with 4 or more literals. I couldn't distinguish the speed of `tuple` vs `list`.

When the values to test were cached in a global variable out of the function, rather than creating the literal within the loop, `set` won every time, even in Python 2.

These results apply to 64-bit CPython on a Core i7.

edited Oct 25 '19 at 22:06

answered Oct 25 '19 at 20:20

Pedro Gimeno
1,796 1 14 25

0

```
from datetime import datetime
listA = range(10000000)
setA = set(listA)
tupA = tuple(listA)
#Source Code

def calc(data, type):
start = datetime.now()
if data in type:
print ""
end = datetime.now()
print end-start

calc(9999, listA)
calc(9999, tupA)
calc(9999, setA)
```

Output after comparing 10 iterations for all 3 : Comparison

answered Sep 19 '19 at 2:33

Harshal SG
165 4

3

tl;dr

Data structures (DS) are important because they are used to perform operations on data which basically implies: *take some input*, *process it*, and *give back the output*.

Some data structures are more useful than others in some particular cases. Therefore, it is quite unfair to ask which (DS) is more efficient/speedy. It is like asking which tool is more efficient between a knife and fork. I mean all depends on the situation.

Lists

A list is **mutable sequence**, **typically used to store collections of homogeneous items**.

Sets

A set object is an **unordered collection of distinct hashable objects**. It is commonly used to test membership, remove duplicates from a sequence, and compute mathematical operations such as intersection, union, difference, and symmetric difference.

Usage

From some of the answers, it is clear that a list is quite faster than a set when iterating over the values. On the other hand, a set is faster than a list when checking if an item is contained within it. Therefore, the only thing you can say is that a list is better than a set for some particular operations and vice-versa.

answered Jul 11 '19 at 6:18

lmiguelvargasf
33k 24 141 152

156

Lists are slightly faster than sets when you just want to iterate over the values.

Sets, however, are significantly faster than lists if you want to check if an item is contained within it. They can only contain unique items though.

It turns out tuples perform in almost exactly the same way as lists, except for their immutability.

Iterating

```
>>> def iter_test(iterable):
...     for i in iterable:
...         pass
...
>>> from timeit import timeit
>>> timeit(
...     "iter_test(iterable)",
...     setup="from __main__ import iter_test; iterable = set(range(10000))",
...     number=100000)
12.666952133178711
>>> timeit(
...     "iter_test(iterable)",
...     setup="from __main__ import iter_test; iterable = list(range(10000))",
...     number=100000)
9.917098999023438
>>> timeit(
...     "iter_test(iterable)",
...     setup="from __main__ import iter_test; iterable = tuple(range(10000))",
```

```
...     number=100000)
9.865639209747314
```

Determine if an object is present

```
>>> def in_test(iterable):
...     for i in range(1000):
...         if i in iterable:
...             pass
...
>>> from timeit import timeit
>>> timeit(
...     "in_test(iterable)",
...     setup="from __main__ import in_test; iterable = set(range(1000))",
...     number=10000)
0.5591847896575928
>>> timeit(
...     "in_test(iterable)",
...     setup="from __main__ import in_test; iterable = list(range(1000))",
...     number=10000)
50.18339991569519
>>> timeit(
...     "in_test(iterable)",
...     setup="from __main__ import in_test; iterable = tuple(range(1000))",
...     number=10000)
51.597304821014404
```

edited Oct 26 '18 at 11:23

answered Jul 30 '13 at 10:51

Ellis Percival
2,979 1 18 25

7 I have found that (Initializing set -> 5.5300979614257812) (Initializing list -> 1.8846848011016846) (Initializing tuple -> 1.8730108737945557) Items of size 10,000 on my intel core i5 quad core with 12GB RAM. This should be take into consideration also. – ThePracticalOne Sep 29 '14 at 17:39

5 I've updated the code to remove the object creation now. The setup phase of the timeit loops is only called once (docs.python.org/2/library/timeit.html#timeit.Timer.timeit). – Ellis Percival Sep 30 '14 at 10:09

0

I would recommend a Set implementation where the use case is limit to referencing or search for existence and Tuple implementation where the use case requires you to perform iteration. A list is a low-level implementation and requires significant memory overhead.

answered May 7 '18 at 8:35

user7763294

1 Indeed, the proper distinction between when to use Sets and when to use Tuple is indeed of utmost importance. I wouldn't be worried about the involved memory overheads, footprints unless I am scripting a lower-level API. – user11768920 Aug 11 '19 at 22:31

7

List performance:

```
>>> import timeit
>>> timeit.timeit(stmt='10**6 in a', setup='a = range(10**6)', number=100000)
0.008128150348026608
```

Set performance:

```
>>> timeit.timeit(stmt='10**6 in a', setup='a = set(range(10**6))', number=100000)
0.005674857488571661
```

You may want to consider **Tuples** as they're similar to lists but can't be modified. They take up slightly less memory and are faster to access. They aren't as flexible but are more efficient than lists. Their normal use is to serve as dictionary keys.

Sets are also sequence structures but with two differences from lists and tuples. Although sets do have an order, that order is arbitrary and not under the programmer's control. The second difference is that the elements in a set must be unique.

`set` by definition. [python | wiki].

```
>>> x = set([1, 1, 2, 2, 3, 3])
>>> x
{1, 2, 3}
```

answered Aug 25 '13 at 22:43

user2601995
5,017 7 32 37

4 First off, you should update to the `set` built-in type link (docs.python.org/2/library/stdtypes.html#set) not the deprecated `sets` library. Second, "Sets are also sequence structures", read the following from the built-in type link: "Being an unordered collection, sets do not record element position or order of insertion. Accordingly, sets do not support indexing, slicing, or other sequence-like behavior." – Seaux Feb 5 '14 at 2:25

7 `range` is not `list`. `range` is a special class with custom `__contains__` magic method. – Ryne Wang Apr 1 '18 at 7:40

@RyneWang this is true, but only for Python3. In Python2 range returns a normal list (that's why exists horrible things like `xrange`) – Manoel Vilela Dec 11 '18 at 17:55

5

This post is hidden. It was deleted 3 years ago by TheLostMind.

It depends on what you mean by "checking for duplicates anyway". What percentage of input is unique? Also slower to do what? Try writing (for each of set and list) the code that does what you want to do, and time it. You can then ask here to have your code and timing methodology checked/improved.

answered May 14 '10 at 1:18

John Machin
72.7k 10 117 172

comments disabled on deleted / locked posts / reviews