In Python, when to use a Dictionary, List or Set?

Asked 9 years, 9 months ago Active 4 months ago Viewed 173k times

294

When should I use a dictionary, list or set?

Are there scenarios that are more suited for each data type?

python list dictionary data-structures set

157

edited Jul 1 '19 at 23:44
dreftymac
25.9k 23 100 161

asked Aug 15 '10 at 20:22
Blankman
224k 281 692 1074

11 Answers

Active | Oldest | Votes

603

A `list` keeps order, `dict` and `set` don't: when you care about order, therefore, you must use `list` (if your choice of containers is limited to these three, of course;-).

`dict` associates with each key a value, while `list` and `set` just contain values: very different use cases, obviously.

`set` requires items to be hashable, `list` doesn't: if you have non-hashable items, therefore, you cannot use `set` and must instead use `list` .

`set` forbids duplicates, `list` does not: also a crucial distinction. (A "multiset", which maps duplicates into a different count for items present more than once, can be found in `collections.Counter` -- you could build one as a `dict` , if for some weird reason you couldn't import `collections` , or, in pre-2.7 Python as a `collections.defaultdict(int)` , using the items as keys and the associated value as the count).

Checking for membership of a value in a `set` (or `dict` , for keys) is blazingly fast (taking about a constant, short time), while in a list it takes time proportional to the list's length in the average and worst cases. So, if you have hashable items, don't care either way about order or duplicates, and want speedy membership checking, `set` is better than `list` .

answered Aug 15 '10 at 20:30
Alex Martelli
706k 142 1110 1315

5 Note that Python 3.7 has dict ordered by default – Gigi Bayte 2 Oct 7 '18 at 5:05

171

- Do you just need to know whether or not you've already *got* a particular value, but without ordering (and you don't need to store duplicates)? Use a set.
- Do you need to associate values with keys, so you can look them up efficiently (by key) later on? Use a dictionary.

answered Aug 15 '10 at 20:24

Jon Skeet
1187k 758 8453
8751

30 This should be the "TL;DR" part to the above answer by Alex M. :-) – Alex Boschmans Feb 23 '15 at 10:42

9 I think it is the other way around.. Alex should donate his votes to Jon. This answer covers pretty much everything and it is much more concise and clear. – mehmet May 2 '16 at 15:59

if, however, you like to know the time complexity difference between these data types Alex's explanation works better – kcEmenike Jun 3 at 13:34

19

When you want an unordered collection of unique elements, use a `set`. (For example, when you want the set of all the words used in a document).

When you want to collect an immutable ordered list of elements, use a `tuple`. (For example, when you want a (name, phone_number) pair that you wish to use as an element in a set, you would need a tuple rather than a list since sets require elements be immutable).

When you want to collect a mutable ordered list of elements, use a `list`. (For example, when you want to append new phone numbers to a list: [number1, number2, ...]).

When you want a mapping from keys to values, use a `dict`. (For example, when you want a telephone book which maps names to phone numbers: `{'John Smith' : '555-1212'}`). Note the keys in a dict are unordered. (If you iterate through a dict (telephone book), the keys (names) may show up in any order).

edited Nov 9 '12 at 13:31

answered Aug 15 '10 at 20:25

unutbu
644k 129 1438
1435

Uh, he didn't ask about tuples. – habnabit Aug 15 '10 at 20:29

18

- Use a dictionary when you have a set of unique keys that map to values.
- Use a list if you have an ordered collection of items.
- Use a set to store an unordered set of items.

answered Aug 15 '10 at 20:25

755k 159 1752
1848

6

In short, use:

`list` - if you require an ordered sequence of items.

`dict` - if you require to relate values with keys

`set` - if you require to keep unique elements.

Detailed Explanation

List

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

A list implements all of the common sequence operations:

- `x in l` and `x not in l`
- `l[i]` , `l[i:j]` , `l[i:j:k]`
- `len(l)` , `min(l)` , `max(l)`
- `l.count(x)`
- `l.index(x[, i[, j]])` - index of the 1st occurrence of `x` in `l` (at or after `i` and before `j` indeces)

A list also implements all of the mutable sequence operations:

- `l[i] = x` - item `i` of `l` is replaced by `x`
- `l[i:j] = t` - slice of `l` from `i` to `j` is replaced by the contents of the iterable `t`
- `del l[i:j]` - same as `l[i:j] = []`
- `l[i:j:k] = t` - the elements of `l[i:j:k]` are replaced by those of `t`
- `del l[i:j:k]` - removes the elements of `s[i:j:k]` from the list
- `l.append(x)` - appends `x` to the end of the sequence
- `l.clear()` - removes all items from `l` (same as del `l[:]`)
- `l.copy()` - creates a shallow copy of `l` (same as `l[:]`)
- `l.extend(t)` or `l += t` - extends `l` with the contents of `t`
- `l *= n` - updates `l` with its contents repeated `n` times
- `l.insert(i, x)` - inserts `x` into `l` at the index given by `i`
- `l.pop([i])` - retrieves the item at `i` and also removes it from `l`
- `l.remove(x)` - remove the first item from `l` where `l[i]` is equal to x
- `l.reverse()` - reverses the items of `l` in place

Dictionary

A dictionary maps hashable values to arbitrary objects. A dictionary is a mutable object. The main operations on a dictionary are storing a value with some key and extracting the value given the key.

In a dictionary, you cannot use as keys values that are not hashable, that is, values containing lists, dictionaries or other mutable types.

Set

A set is an unordered collection of distinct hashable objects. A set is commonly used to include membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference.

answered Jul 11 '19 at 21:18
lmiguelvargasf
28.4k 23 128 137

5

Although this doesn't cover `set` s, it is a good explanation of `dict` s and `list` s:

> Lists are what they seem - a list of values. Each one of them is numbered, starting from zero - the first one is numbered zero, the second 1, the third 2, etc. You can remove values from the list, and add new values to the end. Example: Your many cats' names.
>
> Dictionaries are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - tare similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing. You can add, remove, and modify the values in dictionaries. Example: telephone book.

http://www.sthurlow.com/python/lesson06/

edited Aug 19 '15 at 18:43
jscs
61k 12 139 179

answered Oct 22 '12 at 0:11
Goose
1,310 7 25 38

4

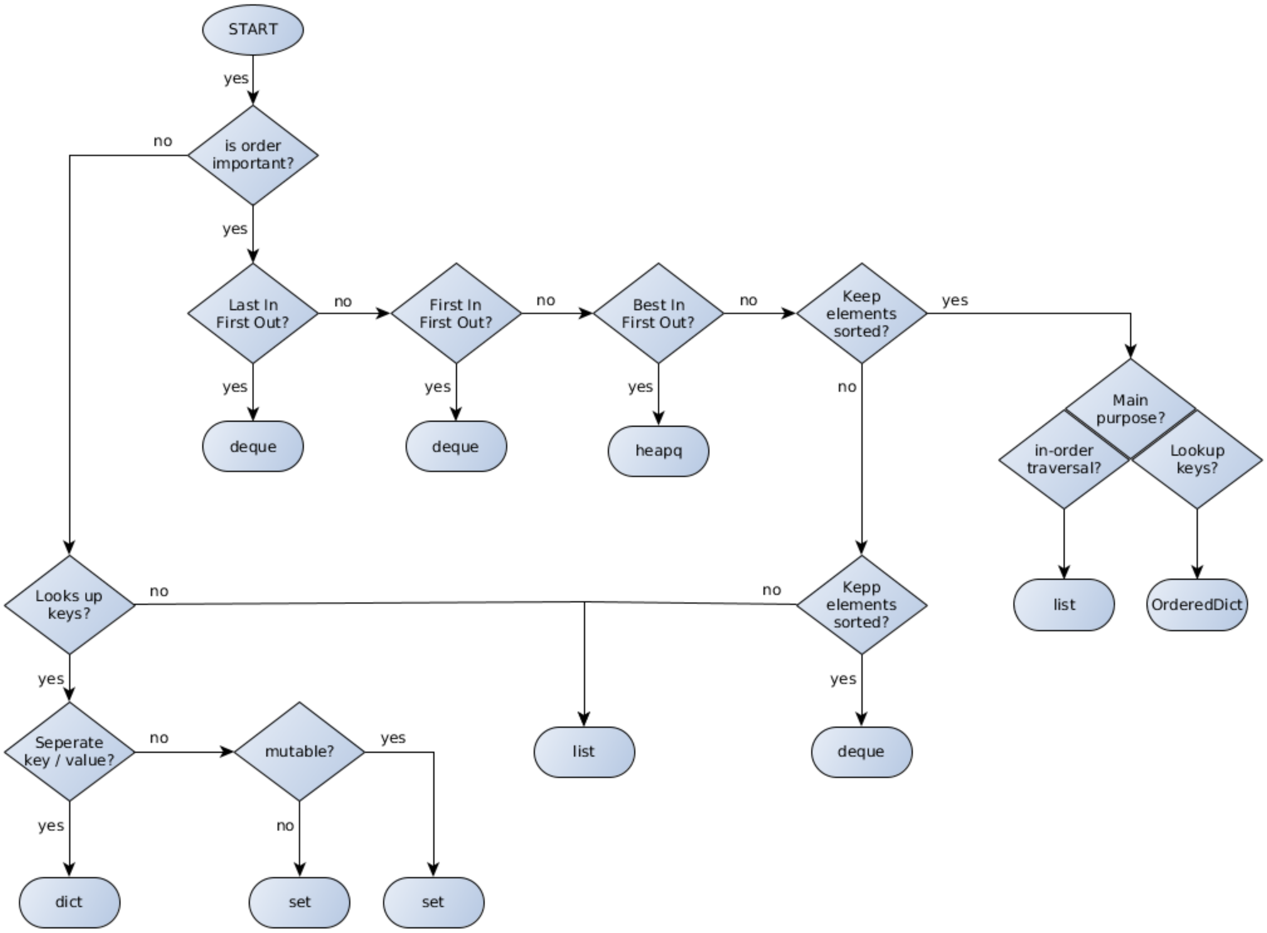
For C++ I was always having this flow chart in mind: In which scenario do I use a particular STL container?, so I was curious if something similar is available for Python3 as well, but I had no luck.

What you need to keep in mind for Python is: There is no single Python standard as for C++. Hence there might be huge differences for different Python interpreters (e.g. CPython, PyPy). The following flow chart is for CPython

Additionally I found no good way to incorporate the following data structures into the diagram: `bytes` , `byte arrays` , `tuples` , `named_tuples` , `ChainMap` , `Counter` , and `arrays` .

- `OrderedDict` and `deque` are available via `collections` module.
- `heapq` is available from the `heapq` module
- `LifoQueue` , `Queue` , and `PriorityQueue` are available via the `queue` module which is designed for concurrent (threads) access. (There is also a `multiprocessing.Queue` available but I don't know the differences to `queue.Queue` but would assume that it should be used when concurrent access from processes is needed.)
- `dict` , `set` , `frozen_set` , and `list` are builtin of course

For anyone I would be grateful if you could improve this answer and provide a better diagram in every aspect. Feel free and welcome.



PS: the diagram has been made with yed. The graphml file [is here]

edited Aug 31 '18 at 8:01

community wiki
2 revs
math

In combination with *lists*, *dicts* and *sets*, there are also another interesting python objects,

> Ordered dictionaries are just like regular dictionaries but they remember the order that items were inserted. When iterating over an ordered dictionary, the items are returned in the order their keys were first added.

OrderedDicts could be useful when you need to preserve the order of the keys, for example working with documents: It's common to need the vector representation of all terms in a document. So using *OrderedDicts* you can efficiently verify if a term has been read before, add terms, extract terms, and after all the manipulations you can extract the ordered vector representation of them.

answered May 8 '18 at 15:25
Federico Caccia
885 6 23

1

Lists are what they seem - a list of values. Each one of them is numbered, starting from zero - the first one is numbered zero, the second 1, the third 2, etc. You can remove values from the list, and add new values to the end. Example: Your many cats' names.

Tuples are just like lists, but you can't change their values. The values that you give it first up, are the values that you are stuck with for the rest of the program. Again, each value is numbered starting from zero, for easy reference. Example: the names of the months of the year.

Dictionaries are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - tare similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing. You can add, remove, and modify the values in dictionaries. Example: telephone book.

answered Jun 18 '17 at 8:17
Nitish Kumar Pal
1,418 11 18

1

When use them, I make an exhaustive cheatsheet of their methods for your reference:

```
class ContainerMethods:
    def __init__(self):
        self.list_methods_11 = {
                    'Add':{'append','extend','insert'},
                    'Subtract':{'pop','remove'},
                    'Sort':{'reverse', 'sort'},
                    'Search':{'count', 'index'},
                    'Entire':{'clear','copy'},
                            }
        self.tuple_methods_2 = {'Search':'count','index'}

        self.dict_methods_11 = {
```

```
                    'Extract':{'get','setdefault',},
                    'Entire':{ 'clear', 'copy','fromkeys'},
                            }
        self.set_methods_17 ={
                    'Add':{['add', 'update'],
['difference_update','symmetric_difference_update','intersection_update']},
                    'Subtract':{'pop', 'remove','discard'},
                    'Relation':{'isdisjoint', 'issubset', 'issuperset'},
                    'operation':{'union' 'intersection','difference',
'symmetric_difference'}
                    'Entire':{'clear', 'copy'}}
```

answered Dec 6 '17 at 11:06

Calculus

11.4k 7 39 68

1

Dictionary: A python dictionary is used like a hash table with key as index and object as value.

List: A list is used for holding objects in an array indexed by position of that object in the array.

Set: A set is a collection with functions that can tell if an object is present or not present in the set.

answered Jan 20 at 5:48

Srinivas P

92 1 6