**Dicts are now ordered, get used to it**

There were several moments over the last few weeks when I heard people discuss differences between Python [lists](https://docs.python.org/3/library/stdtypes.html#list) and [dicts](https://docs.python.org/3/library/stdtypes.html" \l "dict) and one of the first ones mentioned was that lists are ordered and dicts are not.

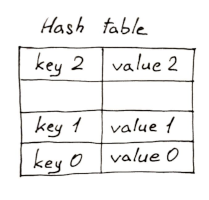
Well, not anymore. Quoting the docs referenced above:

Changed in version 3.7: Dictionary order is guaranteed to be insertion order. This behavior was an implementation detail of CPython from 3.6.

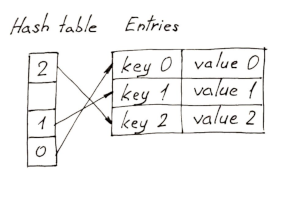
So if you want to discuss *fundamental* differences you can pretty much only point out that dict values are accessible by keys, which could be of any immutable type, while list values are indexed with integers. That's it :-)

**How and why**

A plain hash table holds both keys and values in a pseudo random order determined by hashes calculated from keys. It is also sparse, with unoccupied holes in a pre-allocated array:



Since version 3.6 CPython holds keys and values in a separate dense array, while the hash table itself only holds *indexes* into that:



Since the entries array is populated sequentially, it naturally ensures the order.

As far as I know the initial reason for this change was saving space by sharing hash tables of multiple dicts with the same set of keys (which in Python means instances of the same class, for example). I don't know the exact politics of how this useful property progressed from an implementation detail to a guaranteed behavior.

**dict[int] vs. list**

Does it mean that a dict with int keys is the same as list? There's still a few practical differences.

One obvious difference is the API. You'd have to always mention indexes explicitly when working with a dict[int], there's no such thing as .append(value) or .pop() with no argument. Frankly, I can't see any point in trying to make it work :-)

Also, dicts are bigger, by an order of magnitude:

In [79]: getsizeof(['' **for** i **in** range(100000)])

Out[79]: 824472

In [80]: getsizeof({i: '' **for** i **in** range(100000)})

Out[80]: 5242984

Surprisingly, I couldn't find any consistent difference in speed in neither insertion of new values in a list and a dict[int], nor in traversing them. My gut feeling tells me it's mostly due to the fact that a hash value of an int is the int itself, so there's no time wasted on hashing.

**Not sets though!**

Yep, to my surprise sets are still unordered:

In [3]: list({'one': True, 'two': True, 'three': True, 'four': True, 'five': True})

Out[3]: ['one', 'two', 'three', 'four', 'five']

In [4]: list({'one', 'two', 'three', 'four', 'five'})

Out[4]: ['three', 'five', 'four', 'one', 'two']

Until this moment I thought of sets as basically thin wrappers around actual dicts operating on their keys and neglecting values. Turns out, they have their own implementation.