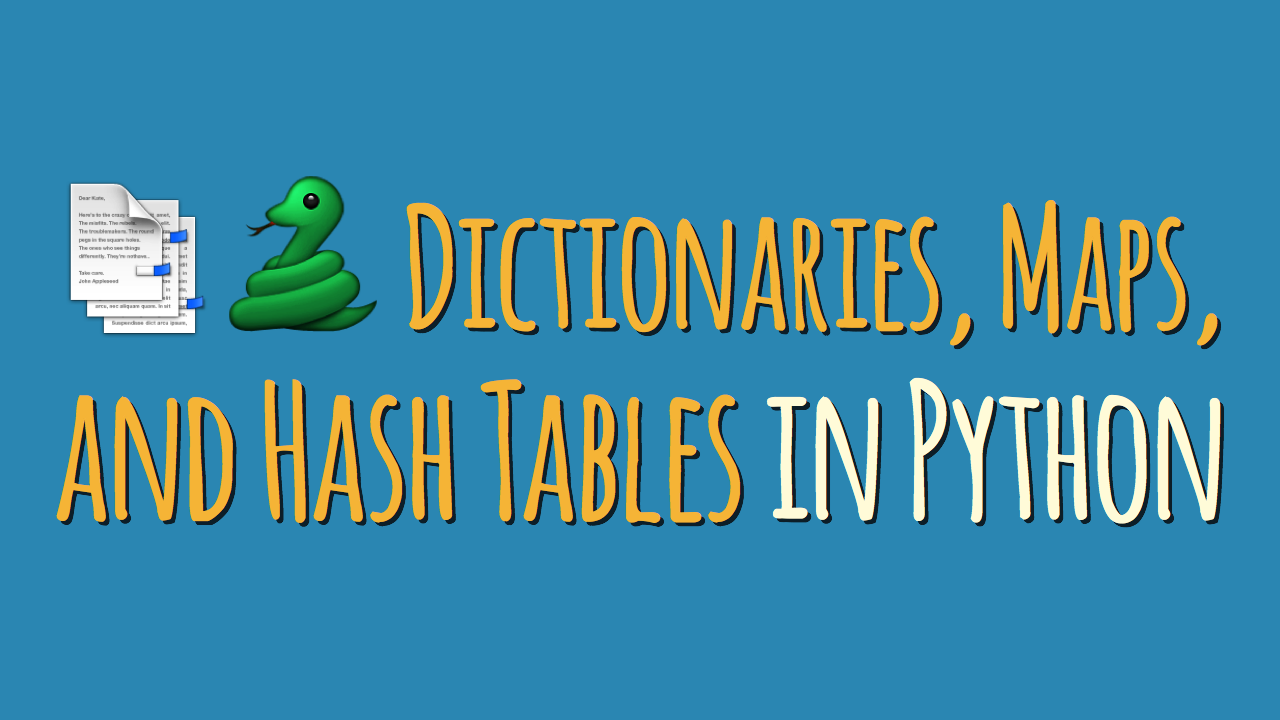
**Dictionaries, Maps, and Hash Tables in Python**

<https://dbader.org/blog/python-dictionaries-maps-and-hashtables>

By Dan Bader — Get free updates of new posts [here](https://www.getdrip.com/forms/80014959/submissions/new).

Need a dictionary, map, or hash table to implement an algorithm in your Python program? Read on to see how the Python standard library can help you.



In Python, dictionaries (or “dicts”, for short) are a central data structure:

Dicts store an arbitrary number of objects, each identified by a unique dictionary *key*. Dictionaries are often also called *maps*, *hashmaps*, *lookup tables*, or *associative arrays*. They allow the efficient lookup, insertion, and deletion of any object associated with a given key.

To give a more practical explanation—*phone books* are a decent real-world analog for dictionaries:

Phone books allow you to quickly retrieve the information (phone number) associated with a given key (a person’s name). Instead of having to read a phonebook front to back in order to find someone’s number you can jump more or less directly to a name and look up the associated number.

This analogy breaks down somewhat when it comes to how the information is organized to allow for fast lookups. But the fundamental performance characteristics hold:

Dictionaries allow you to quickly find the information associated with a given key.

**Python Dictionaries, Hashmaps, and Hash Tables**

The dictionary abstract data type is one of the most frequently used and most important data structures in computer science. Because of this importance Python features a robust dictionary implementation as one of its built-in data types ([dict](https://docs.python.org/3/library/stdtypes.html" \l "mapping-types-dict" \t "_blank)).

Python even provides some useful syntactic sugar for working with dictionaries in your programs. For example, the curly-braces dictionary expression syntax ({}) and dictionary comprehensions allow you to conveniently define new dictionaries:

phonebook = {

'bob': 7387,

'alice': 3719,

'jack': 7052,

}

squares = {x: x \* x for x in range(10)}

Python’s dictionaries are indexed by keys that can be of any [hashable](https://docs.python.org/3/glossary.html#term-hashable) type. A hashable object has a hash value which never changes during its lifetime (see \_\_hash\_\_), and it can be compared to other objects (see \_\_eq\_\_).

In addition, hashable objects which compare equal must have the same hash value. Immutable types like strings and numbers work well as dictionary keys. You can also use tuples as dictionary keys as long as they contain only hashable types themselves.

**✅ Built-in** [**dict**](https://docs.python.org/3/library/stdtypes.html#mapping-types-dict) **type**

For most use cases you’ll face Python’s built-in dictionary implementation will do everything you need. Dictionaries are highly optimized and underlie many parts of the language, for example class attributes and variables in a stack frame are both stored internally in dictionaries.

Python dictionaries are based on a well-tested and finely tuned hash table implementation that provides the performance characteristics you’d expect: *O(1)* time complexity for lookup, insert, update, and delete operations in the average case.

There’s little reason to not use the standard dict implementation included with Python. However, specialized third-party dictionary data structures exist, for example skip lists or B-tree based dictionary implementations.

>>> phonebook = {'bob': 7387, 'alice': 3719, 'jack': 7052}

>>> phonebook['alice']

3719

Interestingly, **Python ships with a number of specialized dictionary implementations in its standard library**. These specialized dictionaries are all based on the built-in dictionary implementation (and share its performance characteristics) but add some convenience features:

**✅** [**collections.OrderedDict**](https://docs.python.org/3/library/collections.html#collections.OrderedDict) **– Remember the insertion order of keys**

A dictionary subclass that remembers the insertion order of keys added to the collection.

While standard dict instances [preserve the insertion order of keys in CPython 3.6+](https://mail.python.org/pipermail/python-dev/2016-September/146327.html) this is just a side effect of the CPython implementation and not defined in the language spec. If key order is important for your algorithm to work it’s best to communicate this clearly by using the OrderDict class.

OrderedDict is not a built-in part of the core language and must be imported from the collections module in the standard library.

>>> import collections

>>> d = collections.OrderedDict(one=1, two=2, three=3)

>>> d

OrderedDict([('one', 1), ('two', 2), ('three', 3)])

>>> d['four'] = 4

>>> d

OrderedDict([('one', 1), ('two', 2), ('three', 3), ('four', 4)])

>>> d.keys()

odict\_keys(['one', 'two', 'three', 'four'])

**✅** [**collections.defaultdict**](https://docs.python.org/3/library/collections.html#collections.defaultdict) **– Return default values for missing keys**

Another dictionary subclass that accepts a default value in its constructor that will be returned if a requested key cannot be found in a defaultdict instance. This can save some typing and make the programmer’s intention more clear compared to using the get() methods or catching a KeyError exception in regular dictionaries.

>>> from collections import defaultdict

>>> dd = defaultdict(list)

# Accessing a missing key creates it and initializes it

# using the default factory, i.e. list() in this example:

>>> dd['dogs'].append('Rufus')

>>> dd['dogs'].append('Kathrin')

>>> dd['dogs'].append('Mr Sniffles')

>>> dd['dogs']

['Rufus', 'Kathrin', 'Mr Sniffles']

**✅** [**collections.ChainMap**](https://docs.python.org/3/library/collections.html#collections.ChainMap) **– Search multiple dictionaries as a single mapping**

This data structure groups multiple dictionaries into a single mapping. Lookups search the underlying mappings one by one until a key is found. Insertions, updates, and deletions only affect the first mapping added to the chain.

>>> from collections import ChainMap

>>> dict1 = {'one': 1, 'two': 2}

>>> dict2 = {'three': 3, 'four': 4}

>>> chain = ChainMap(dict1, dict2)

>>> chain

ChainMap({'one': 1, 'two': 2}, {'three': 3, 'four': 4})

# ChainMap searches each collection in the chain

# from left to right until it finds the key (or fails):

>>> chain['three']

3

>>> chain['one']

1

>>> chain['missing']

KeyError: 'missing'

**✅** [**types.MappingProxyType**](https://docs.python.org/3/library/types.html#types.MappingProxyType) **– A wrapper for making read-only dictionaries**

A wrapper around a standard dictionary that provides a read-only view into the wrapped dictionary’s data. This class was added in Python 3.3 and it can be used to create immutable proxy versions of dictionaries.

>>> from types import MappingProxyType

>>> read\_only = MappingProxyType({'one': 1, 'two': 2})

>>> read\_only['one']

1

>>> read\_only['one'] = 23

TypeError: "'mappingproxy' object does not support item assignment"

**Using Dictionaries in Python: Conclusion**

All of the Python hashmap implementations I listed in this tutorial are valid implementations built into the Python standard library.

If you’re looking for a general recommendation on which mapping type to use in your Python programs, then I’d point you to the built-in dict data type. It’s a versatile and optimized dictionary implementation that’s built directly into the core language.

Only if you have special requirements that go beyond what’s provided by dict would I recommend that you use one of the other data types listed here. Yes, I still believe they’re valid options—but usually your code will be clearer and easier to maintain by other developers if it relies on standard Python dictionaries most of the time.

[*Read the full “Fundamental Data Structures in Python” article series here.*](https://dbader.org/blog/fundamental-data-structures-in-python) *This article is missing something or you found an error? Help a brother out and leave a comment below.*