

TITLE: Python Programming: Problem Solving, Packages and Libraries

Lecture PPT
Edition

Dictionaries Learning Objectives

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 1 Understand that a dictionary is not a **sequence**, so it has no index. Rather it is a **collection** or a **mapping**.
- LO 2 Explain the concept of **hashability** and **immutability** with respect to keys.
- LO 3 Create, initialize and access elements of a dictionary.
- LO 4 Realise that if you have a key you can get a value but not vice-versa.
- LO 5 **Understand** dictionary comprehension, and traversing/looping over a dictionary.
- LO 6 Use some common dictionary functions/methods.
- LO 7 Differentiate between 2.x and 3.x in respect of methods keys(), values() items()

Properties of a dictionary

- ***A dictionary*** is an unordered collection of key-value pairs.
- It is like a list, but with one important difference. In a list, the items are accessed by their position, which is an integer. While in a dictionary, the items are accessed by their keys, which can be other data types (with certain restrictions).
- Also, a list is a sequence, whereas a dictionary is a collection but not a sequence. It contains items but their order or sequence doesn't matter.
- In a dictionary, a key and its value is separated by a colon (:). Moreover, each key-value pair is separated from the next by a comma (,). Then all the pairs are enclosed in curly braces.
- Keys within a dictionary must be unique while values need not be.

- A dictionary can be visualized as a mapping between a set of indices (which are called keys) and a corresponding set of values. Each key maps to a value.
- Together, a key-value pair is often called an item of a dictionary.
- For instance, the dictionary `myDict = {'a':'apple', 'b':'baby', 'c': 'cat', 'd': 'dog'}` can be represented as follows:

KEYS		VALUES
'a'	→	'apple'
'b'	→	'baby'
'c'	→	'cat'
'd'	→	'dog'

Some other points regarding a dictionary -- 1

- 1. You must add a key and its value together in a dictionary. This means that you cannot have a dictionary, which has a key but does not have a corresponding value. You may change a value anytime, but you must have a value for each key.**
- 2. Once you have a dictionary, you can look up values by their key, but not the other way around. This is like ordinary real-world dictionary, where when you know a word, you can find its meaning but given the meaning cannot find the word.**
- 3. Dictionaries don't have a size limit. New key-value pairs can be added to a dictionary at any time. You can also modify the value of any/ all existing key(s).**
- 4. Note that a dictionary itself is mutable but the objects it stores as keys must be immutable.**

Some other points regarding a dictionary -- 2

5. In a dictionary, the keys must be unique while values need not be. So two different keys can have the same value but not vice-versa.
6. Dictionaries can grow (Items added) or shrink (Items deleted) without new copies being made. Each *key* can have just one associated *value*, but the key itself can be a *collection* of multiple objects if needed.
7. Note that the objects, a dictionary stores as values can also be mutable. (But keys cannot be mutable). Therefore, a list which is mutable can be a value of a dictionary but cannot be the key of a dictionary.
8. Note that you can use tuples as dictionary keys because they are immutable but lists cannot be used as dictionary keys because they are mutable.

Concept of hashable and dictionary keys must be hashable

- Please note that when you try to use a list as a key for a dictionary, you get an error, that is, unhashable type:'list'.
- In Python, generally speaking those objects, which are immutable are also hashable.
- Python has an inbuilt hash() function to find the hash value of an object.
- But this hash() function works only on those types which are immutable. Hence hash() does not work on lists which are mutable.
- Further note that the hash() function actually works on the id() of the object. As explained earlier, the id() of immutable objects will change if the object changes.
- As told earlier, the id() of a mutable object, such as list remains the same even when its values are changed. Hence, a list is not hashable.
- Please note that a dictionary is also not hashable because a dictionary is mutable.

Mutability of dictionary

- One important concept to understand is that a dictionary as a whole is mutable.
- This means that you can change the individual items in a dictionary. But the 'type' of variables you use for keys must be 'immutable'.
- This puts certain restrictions on what variables you can use as keys.
- You can use numbers, strings and tuples as keys but you cannot use lists and other dictionaries as keys.

Dictionary comprehension

Dictionary comprehensions are similar to list comprehensions. The difference is that you need to use curly braces instead of square braces.

The other difference is that you need to give a key-value pair separated by a colon before you use the “for” keyword.

Dictionary comprehensions are especially useful when you have some data in nested length 2 sequence, or when you can generate a nested length 2 sequence. (For instance, you can easily generate a nested sequence of a number and its square.)

The general format for dictionary comprehension is as follows:-

```
{key_expr: value_expr for var in iterable}
```

The above terms are explained below:-

key_expr: This is the expression which generates the key

value_expr: This is the expression which generates the ‘Value’

for var in: var is a temporary variable which stores each of the items in the iterable one by one.

Iterable: This is some iterable like a sequence or collection or range function etc.

In a dictionary, duplicate keys are not allowed (But duplicate values are allowed).

- **In a key-value pair in a dictionary, there are no restrictions on the value.**
- **The value can be any standard Python object (including built-in data types, standard built-in Python objects or even user-defined objects.**
- **You cannot have duplicate keys.**
- **If you do try to have duplicate keys in an assignment to a dictionary, the last assignment will be retained and the earlier one will be dropped.**

Summary of common dictionary functions

Sr	Function	Description
1	<code>cmp(dict1, dict2)</code>	Compares the elements of the two dictionaries d1 and d2.
2	<code>len(dict)</code>	Gives the number of key-value pairs in the dictionary ie the number of “items” in the dictionary.
3	<code>type(variable)</code>	The <code>type()</code> function gives the “type” of the variable given to the function.
4	<code>str(dict)</code>	Gives a “string representation” of a dictionary.
5	The ‘in’ operator	The “in” operator checks for existence of a key in a dictionary. In python 3.x you should use the “in” operator to check for a key rather than use <code>dictionary.has_key(“key_name”)</code> method.

The usage of dictionary functions is quite straight forward and hence not discussed here. The book has detailed explanation along with examples for these dictionary functions.

Summary of dictionary methods

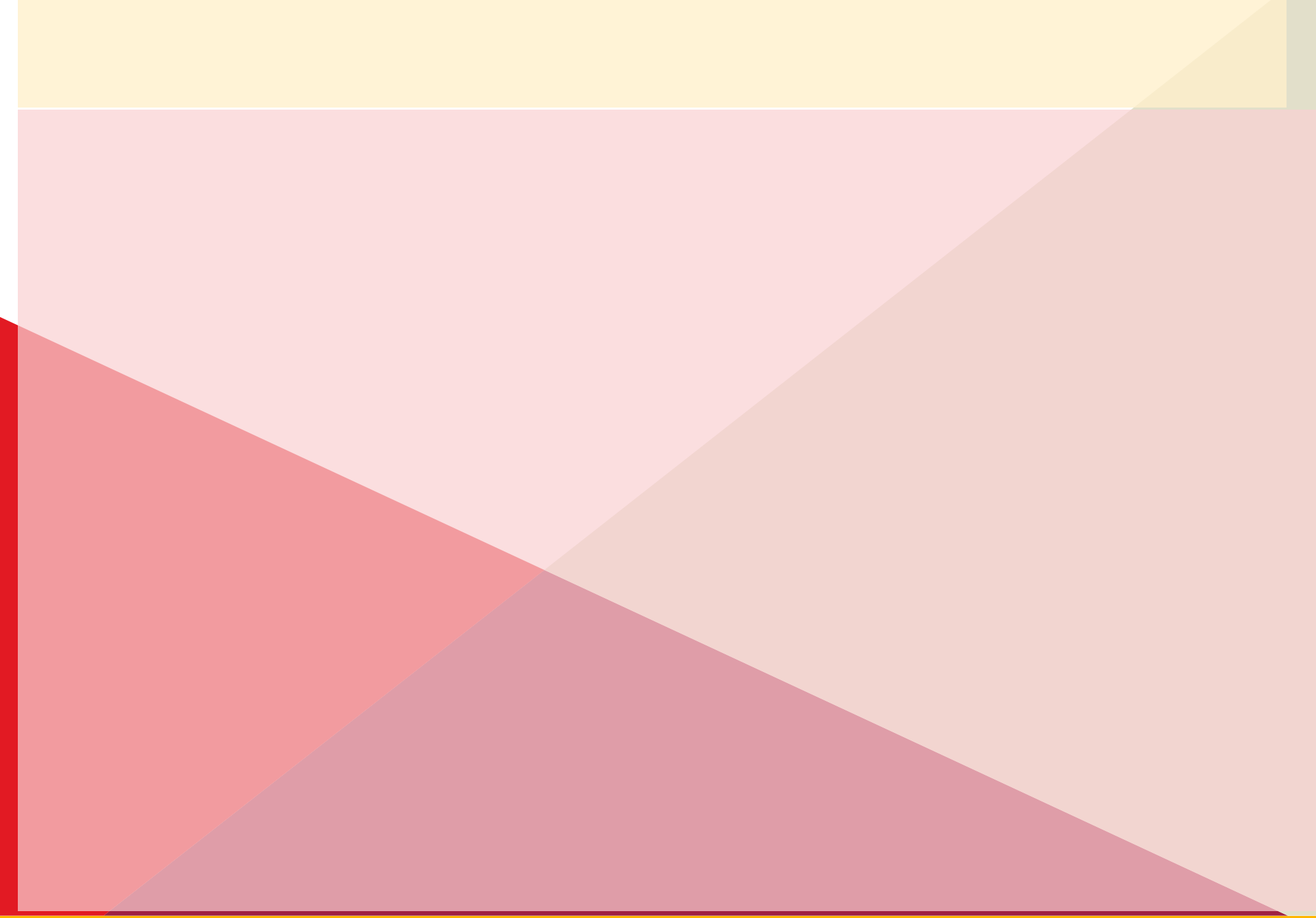
d.clear()	Removes all items ie key-value pairs of dictionary dict
dict.copy()	Returns a copy of the dictionary. It doesn't take any parameters.
dict.fromkeys(seq [, value])	Takes two parameters. First is the sequence seq which is used as key to the dictionary. The second parameter value is optional. If given, it provides the default value for each value in the key-value pair. If argument value is not provided, then each key has a corresponding value of None .
dict.has_key(some_key)	Returns True if the dictionary has the given key, that is, some_key. Else it returns False. This method is deprecated in Python 3.x. For 3.x you must use the in operator.
dict.get(some_key, default=None)	Method takes two parameters. - Parameter some_key is the key to be searched in the dictionary. If some_key exists, its value is returned - Parameter value is optional. It is the value to be returned if the key is not found in the dictionary. If the second parameter is not given, a default value of None is returned.
dict.keys()	It returns a view object. This view object is similar to a list and contains all the keys of the dictionary.
dict.items()	Method takes no parameters. It returns a view object similar to a list of tuples. Each tuple is one key-value pair.
dict.values()	Method takes no parameters. It returns a view object which is similar to a list of all the values in the dictionary.
dict.setdefault(some_key, default_value =None)	Has two parameters - some_key: It is the key to be searched - default_value:- It comes into picture only if the key being searched, that is, some_key is not found. If this parameter default_value is given some value, then that value is returned. But if it is given no value, then a default of None is returned. Note this method is similar to get(). The difference is that get() only gets value for a key, but setdefault() sets the key-value pair in case the key is not present in the dictionary
dict.update(dict2)	Here there are two dictionaries, which may or may not have common keys. If there are no common keys, then all the key-value pairs of dict2 are added to dict1. However, if there are some common and some unique keys in dict2, then for the common keys, the corresponding values of dict1 are replaced with values of dict2. For the uncommon keys, the key-value pairs of dict2 are added to dict1.

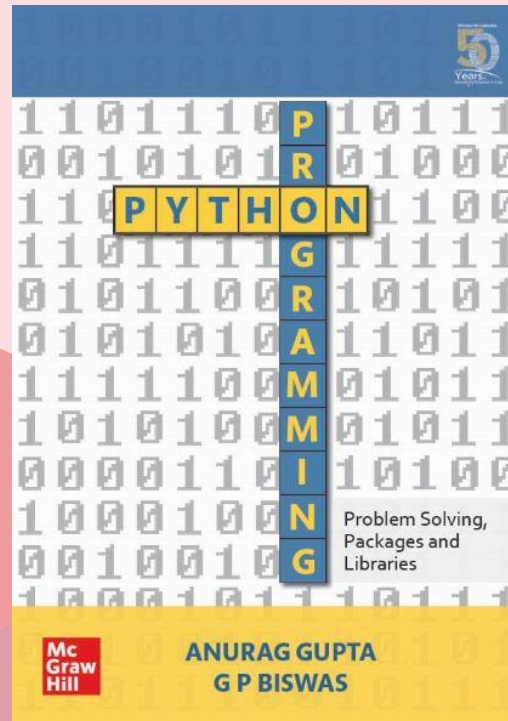
Dictionary view objects

- An important difference between Python 2.x and 3.x is the value or objects 'returned' by methods: `dict.keys()`, `dict.values()` and `dict.items()`.
- In 2.x the objects returned by these three methods are lists, but in 3.x they are view objects.

What does the expression return value is a **view-object** mean?

- It means that if we have a dictionary object say `myDict` and we find its keys using the `myDict.keys()` function then the return value of this `keys()` function is a **view object**.
- Same is the case for `myDict.values()` and `myDict.items()`.
- It further means that when we are writing a script, then we can **assign** a variable name to the **view objects** returned by these three functions.
- But you can always cast this **view object** into a list.








Because learning changes everything.®

Thank You!

For any queries or feedback contact us at:

 support.india@mheducation.com

 1800-103-5875

 www.mheducation.co.in

in

