Like lists, dictionaries are used to organize elements into collections. Unlike lists, you don't access elements inside dictionaries using their position. Instead, the data inside dictionaries take the form of pairs of keys and values.

dictionaries are mutable

How to define a dictionary?

dict={“key1” : value1, “key2” : value2, “key3” : value3}

A dictionary’s keys(key1,key2,key3) are almost arbitrary values(strings, integers, floats, tuples, and more.). Values(value1, value2, value3) that are not hashable, that is, values can be one of lists, dictionaries or other mutable types

**List of immutable types:**

int, float, decimal, complex, bool, string, tuple, range, frozenset, bytes

**List of mutable types:**

list, dict, set, bytearray, user-defined classes

All built-in immutable types are hashable, but mutable ones are not. Common hashable types include all numbers, strings (both unicode and bytes ) and tuple . Common unhashable types include list , dict and set . In order to store a key in a dict or set a hash value is needed.

toc = {"Introduction":[1,2,3],"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
print(type(toc))

<class 'dict'>

You can create an empty dictionary like an empty list:

toc = {}  
print(type(toc))

<class 'dict'>

Different kinds of methods that can be used along with dictionaries::

Method Description

clear() Removes all the elements from the dictionary

copy() Returns a copy of the dictionary

fromkeys() Returns a dictionary with the specified keys and value

get(key, default) Returns the element corresponding to key, or default if it's not present

keys() Returns a list containing the dictionary's keys

values() Returns a list of all the values in the dictionary

items() Returns a list containing a tuple for each key value pair

pop() Removes the element with the specified key

popitem() Removes the last inserted key-value pair

setdefault() Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

dict.update(other\_dictionary) Updates the dictionary with the items coming from the other dictionary. Existing entries will be replaced; new entries will be added.

list(d) Return a list of all the keys used in the dictionary d.

**Operations::**

* len(dictionary) - Returns the number of items in the dictionary
* for key in dictionary - Iterates over each key in the dictionary
* for key, value in dictionary.items() - Iterates over each key,value pair in the dictionary
* if key in dictionary - Checks whether the key is in the dictionary
* dictionary[key] - Accesses the item with key key of the dictionary
* dictionary[key] = value - Sets the value associated with key
* del dictionary[key] - Removes the item with key key from the dictionary. Raises a KeyError if key is not in the map

toc = {"Introduction":[1,2,3],"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
values=0  
print(toc.fromkeys(toc, values))  
print(toc)

output::

{'Introduction': 0, 'Chapter1': 0, 'Chapter2': 0, 'Chapter3': 0, 'Chapter4': 0, 'Epilogue': 0}

{'Introduction': [1, 2, 3], 'Chapter1': 4, 'Chapter2': 11, 'Chapter3': 25, 'Chapter4': 30, 'Epilogue': 39}

toc = {"Introduction":[1,2,3],"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
print(toc.get("Introduction"))

output::[1,2,3]

toc = {"Introduction":1,"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
print(toc.items())

dict\_items([('Introduction', 1), ('Chapter1', 4), ('Chapter2', 11), ('Chapter3', 25), ('Chapter4', 30), ('Epilogue', 39)])

toc = {"Introduction":1,"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
toc.pop("Introduction")  
print(toc)

{'Chapter1': 4, 'Chapter2': 11, 'Chapter3': 25, 'Chapter4': 30, 'Epilogue': 39}

toc = {"Introduction":1,"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
print(toc.setdefault("Introduction"))  
print(toc.setdefault("Introduction",5))  
print(toc.setdefault("key",5))   
print(toc)

1

1

5

{'Introduction': 1, 'Chapter1': 4, 'Chapter2': 11, 'Chapter3': 25, 'Chapter4': 30, 'Epilogue': 39, 'key': 5}

toc = {"(1,2,3)":1,"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
toc.update({"new key":123})  
print(toc)

{'(1,2,3)': 1, 'Chapter1': 4, 'Chapter2': 11, 'Chapter3': 25, 'Chapter4': 30, 'Epilogue': 39, 'new key': 123}

*Another way of doing that*

toc = {"(1,2,3)":1,"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
toc["new\_item"] = 123  
print(toc)

{'(1,2,3)': 1, 'Chapter1': 4, 'Chapter2': 11, 'Chapter3': 25, 'Chapter4': 30, 'Epilogue': 39, 'new\_item': 123}

What if we try to add a key that already exists? The value of that key gets updated

toc = {"(1,2,3)":1,"Chapter1":4,"Chapter2":11,"Chapter3":25,"Chapter4":30,"Epilogue": 39}  
toc["Chapter3"] = 123  
print(toc)

{'(1,2,3)': 1, 'Chapter1': 4, 'Chapter2': 11, 'Chapter3': 123, 'Chapter4': 30, 'Epilogue': 39}

The keys inside of a dictionary are unique. If we try to store two different values for the same key, we'll just replace one with the other.

**Iterating over dictionaries::**

toc = {"Introduction":1, "Chapter 1":4, "Chapter 2":11, "Chapter 3":25, "Chapter 4":30}  
for ele in toc:  
 print(ele)

Introduction

Chapter 1

Chapter 2

Chapter 3

Chapter 4

toc = {"Introduction":1, "Chapter 1":4, "Chapter 2":11, "Chapter 3":25, "Chapter 4":30}  
for ele in toc:  
 print(toc[ele])

1

4

11

25

30

toc = {"Introduction":1, "Chapter 1":4, "Chapter 2":11, "Chapter 3":25, "Chapter 4":30}  
for ele in toc.items():  
 print(ele)

('Introduction', 1)

('Chapter 1', 4)

('Chapter 2', 11)

('Chapter 3', 25)

('Chapter 4', 30)

toc = {"Introduction":1, "Chapter 1":4, "Chapter 2":11, "Chapter 3":25, "Chapter 4":30}  
for chapter,page in toc.items():  
 print("{} starts from page no {}.".format(chapter,page))

Introduction starts from page no 1.

Chapter 1 starts from page no 4.

Chapter 2 starts from page no 11.

Chapter 3 starts from page no 25.

Chapter 4 starts from page no 30.

You can iterate over dictionaries using a for loop, just like with strings, lists, and tuples. This will iterate over the sequence of keys in the dictionary. If you want to access the corresponding values associated with the keys, you could use the keys as indexes. Or you can use the **items** method on the dictionary, like **dictionary.items()**. This method returns a tuple for each element in the dictionary, where the first element in the tuple is the key and the second is the value.

If you only wanted to access the keys in a dictionary, you could use the **keys()** method on the dictionary: **dictionary.keys()**. If you only wanted the values, you could use the **values()**method: **dictionary.values()**.

Dictionaries and lists are both really useful and each have strengths in different situations. So when is it best to use a list and when is the dictionary the way to go? Think about the kind of information you can represent in each data structure. If you've got a list of information you'd like to collect and use in your script then the list is probably the right approach. For example, if you want to store a series of IP addresses to ping, you could put them all into a list and iterate over them. Or if you had a list of host names and their corresponding IP addresses, you might want to pair them as key values in a dictionary

ip\_address=["192.168.43.1","127.0.0.1"]  
host\_address={"router":"192.168.43.1" , "localhost":"127.0.0.1"}

Important Note::Because of the way dictionaries work, it's super easy and fast to search for an element in them. Let's say you have a dictionary that has usernames as keys, and the groups they belong to as values. It doesn't matter if you have 10 users or 10,000 users, accessing the entry for a given user will take the same time. Amazing, but this isn't true for lists. If you've got a list of 10 elements, and you need to check if one element is in the list, it'll be a very fast check but if your list has 10,000 elements it'll take significantly longer to check if the element you're looking for is there. So in general, you want to use dictionaries when you plan on searching for a specific element.

Another interesting difference is the types of values that we can store in lists and dictionaries. In lists, you can store any data type. In dictionaries, we can store any data type for the values but the keys are restricted to specific types. To declare a keys you can use any one of immutable data type; numbers, booleans, strings and tuples as dictionary keys. But you can't use lists or dictionaries for that.