Dictionaries

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Dictionaries are optimized to retrieve values when the key is known.

## **How to create a dictionary?**

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.

An item has a key and the corresponding value expressed as a pair, key: value.

While values can be of any data type and can repeat, keys must be of immutable type ([string](https://www.programiz.com/python-programming/string), [number](https://www.programiz.com/python-programming/numbers) or [tuple](https://www.programiz.com/python-programming/tuple) with immutable elements) and must be unique.

# empty dictionary  
my\_dict = {}  
  
# dictionary with integer keys  
my\_dict = {1: 'apple', 2: 'ball'}  
  
# dictionary with mixed keys  
my\_dict = {'name': 'John', 1: [2, 4, 3]}  
  
# using dict()  
my\_dict = dict({1:'apple', 2:'ball'})  
  
# from sequence having each item as a pair  
my\_dict = dict([(1,'apple'), (2,'ball')])

As you can see above, we can also create a dictionary using the built-in function dict().

## **How to access elements from a dictionary?**

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.

The difference while using get() is that it returns None instead of KeyError, if the key is not found.

Example:

my\_dict = {'name':'Jack', 'age': 26}

# Output: Jack

print(my\_dict['name'])

# Output: 26

print(my\_dict.get('age'))

# Trying to access keys which doesn't exist throws error

# my\_dict.get('address')

# my\_dict['address']

## **How to change or add elements in a dictionary?**

Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

my\_dict = {'name':'Jack', 'age': 26}

# update value

my\_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}

print(my\_dict)

# add item

my\_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}

print(my\_dict)

## **How to delete or remove elements from a dictionary?**

We can remove a particular item in a dictionary by using the method pop(). This method removes as item with the provided key and returns the value.

The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the clear() method.

We can also use the del keyword to remove individual items or the entire dictionary itself.

# create a dictionary

squares = {1:1, 2:4, 3:9, 4:16, 5:25}

# remove a particular item

# Output: 16

print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}

print(squares)

# remove an arbitrary item

# Output: (1, 1)

print(squares.popitem())

# Output: {2: 4, 3: 9, 5: 25}

print(squares)

# delete a particular item

del squares[5]

# Output: {2: 4, 3: 9}

print(squares)

# remove all items

squares.clear()

# Output: {}

print(squares)

# delete the dictionary itself

del squares

# Throws Error

# print(squares)

## **Python Dictionary Methods**

Methods that are available with dictionary are tabulated below. Some of them have already been used in the above examples.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [clear()](https://www.programiz.com/python-programming/methods/dictionary/clear) | Remove all items form the dictionary. |
| [copy()](https://www.programiz.com/python-programming/methods/dictionary/copy) | Return a shallow copy of the dictionary. |
| [fromkeys(seq[, v])](https://www.programiz.com/python-programming/methods/dictionary/fromkeys) | Return a new dictionary with keys from seq and value equal to v (defaults to None). |
| [get(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/get) | Return the value of key. If key doesnot exit, return d (defaults to None). |
| [items()](https://www.programiz.com/python-programming/methods/dictionary/items) | Return a new view of the dictionary's items (key, value). |
| [keys()](https://www.programiz.com/python-programming/methods/dictionary/keys) | Return a new view of the dictionary's keys. |
| [pop(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/pop) | Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError. |
| [popitem()](https://www.programiz.com/python-programming/methods/dictionary/popitem) | Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty. |
| [setdefault(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/setdefault) | If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None). |
| [update([other])](https://www.programiz.com/python-programming/methods/dictionary/update) | Update the dictionary with the key/value pairs from other, overwriting existing keys. |
| [values()](https://www.programiz.com/python-programming/methods/dictionary/values) | Return a new view of the dictionary's values |

## **Other Dictionary Operations**

### **Dictionary Membership Test**

We can test if a key is in a dictionary or not using the keyword in. Notice that membership test is for keys only, not for values.

squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True

print(1 in squares)

# Output: True

print(2 not in squares)

# membership tests for key only not value

# Output: False

print(49 in squares)

### **Iterating Through a Dictionary**

Using a for loop we can iterate though each key in a dictionary.

squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

for i in squares:

print(squares[i])

### **Built-in Functions with Dictionary**

Built-in functions like all(), any(), len(), cmp(), sorted() etc. are commonly used with dictionary to perform different tasks.

|  |  |
| --- | --- |
| **Function** | **Description** |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all keys of the dictionary are true (or if the dictionary is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any key of the dictionary is true. If the dictionary is empty, return False. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the dictionary. |
| cmp() | Compares items of two dictionaries. |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list of keys in the dictionary. |

Sets

A set contains an unordered collection of unique and immutable objects. The set data type is, as the name implies, a Python implementation of the sets as they are known from mathematics. This explains, why sets unlike lists or tuples can't have multiple occurrences of the same element.

### 

## **How to create a set?**

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like [list](https://www.programiz.com/python-programming/list), set or [dictionary](https://www.programiz.com/python-programming/dictionary), as its element.

# set of integers

my\_set = {1, 2, 3}

print(my\_set)

# set of mixed datatypes

my\_set = {1.0, "Hello", (1, 2, 3)}

print(my\_set)

# set do not have duplicates

# Output: {1, 2, 3, 4}

my\_set = {1,2,3,4,3,2}

print(my\_set)

# set cannot have mutable items

# here [3, 4] is a mutable list

# If you uncomment line #12,

# this will cause an error.

# TypeError: unhashable type: 'list'

#my\_set = {1, 2, [3, 4]}

# we can make set from a list

# Output: {1, 2, 3}

my\_set = set([1,2,3,2])

print(my\_set)

Creating an empty set is a bit tricky.

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set() function without any argument.

# initialize a with {}

a = {}

# check data type of a

# Output: <class 'dict'>

print(type(a))

# initialize a with set()

a = set()

# check data type of a

# Output: <class 'set'>

print(type(a))

## **How to change a set in Python?**

Sets are mutable. But since they are unordered, indexing have no meaning.

We cannot access or change an element of set using indexing or slicing. Set does not support it.

We can add single element using the add() method and multiple elements using the update() method. The update() method can take [tuples](https://www.programiz.com/python-programming/tuple), lists, [strings](https://www.programiz.com/python-programming/string) or other sets as its argument. In all cases, duplicates are avoided.

# initialize my\_set

my\_set = {1,3}

print(my\_set)

# if you uncomment line 9,

# you will get an error

# TypeError: 'set' object does not support indexing

#my\_set[0]

# add an element

# Output: {1, 2, 3}

my\_set.add(2)

print(my\_set)

# add multiple elements

# Output: {1, 2, 3, 4}

my\_set.update([2,3,4])

print(my\_set)

# add list and set

# Output: {1, 2, 3, 4, 5, 6, 8}

my\_set.update([4,5], {1,6,8})

print(my\_set)

## **How to remove elements from a set?**

A particular item can be removed from set using methods, discard() and remove().

The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

The following example will illustrate this.

# initialize my\_set

my\_set = {1, 3, 4, 5, 6}

print(my\_set)

# discard an element

# Output: {1, 3, 5, 6}

my\_set.discard(4)

print(my\_set)

# remove an element

# Output: {1, 3, 5}

my\_set.remove(6)

print(my\_set)

# discard an element

# not present in my\_set

# Output: {1, 3, 5}

my\_set.discard(2)

print(my\_set)

# remove an element

# not present in my\_set

# If you uncomment line 27,

# you will get an error.

# Output: KeyError: 2

#my\_set.remove(2)

Similarly, we can remove and return an item using the pop() method.

Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.

We can also remove all items from a set using clear().

# initialize my\_set

# Output: set of unique elements

my\_set = set("HelloWorld")

print(my\_set)

# pop an element

# Output: random element

print(my\_set.pop())

# pop another element

# Output: random element

my\_set.pop()

print(my\_set)

# clear my\_set

#Output: set()

my\_set.clear()

print(my\_set)

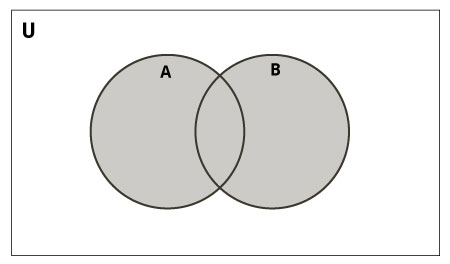
## **Python Set Operations**

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

Let us consider the following two sets for the following operations.

>>> A = {1, 2, 3, 4, 5}  
>>> B = {4, 5, 6, 7, 8}

### **Set Union**



Union of A and B is a set of all elements from both sets.

Union is performed using | operator. Same can be accomplished using the method union().

# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use | operator

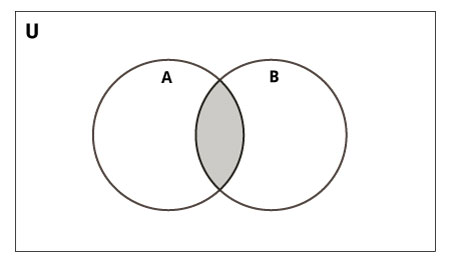
# Output: {1, 2, 3, 4, 5, 6, 7, 8}

print(A | B)

Try the following examples on Python shell.

# use union function  
>>> A.union(B)  
{1, 2, 3, 4, 5, 6, 7, 8}  
  
# use union function on B  
>>> B.union(A)  
{1, 2, 3, 4, 5, 6, 7, 8}

### **Set Intersection**



Intersection of A and B is a set of elements that are common in both sets.

Intersection is performed using & operator. Same can be accomplished using the method intersection().

# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use & operator

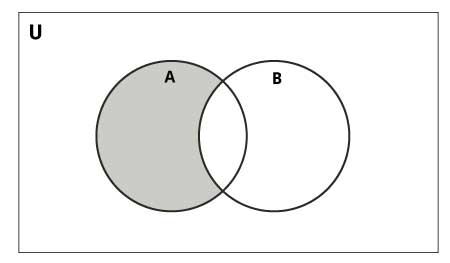
# Output: {4, 5}

print(A & B)

Try the following examples on Python shell.

# use intersection function on A  
>>> A.intersection(B)  
{4, 5}  
  
# use intersection function on B  
>>> B.intersection(A)  
{4, 5}

### **Set Difference**



Difference of A and B (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of element in B but not in A.

Difference is performed using - operator. Same can be accomplished using the method difference().

# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use - operator on A

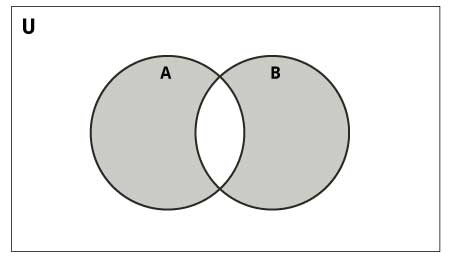
# Output: {1, 2, 3}

print(A - B)

Try the following examples on Python shell.

# use difference function on A  
>>> A.difference(B)  
{1, 2, 3}  
  
# use - operator on B  
>>> B - A  
{8, 6, 7}  
  
# use difference function on B  
>>> B.difference(A)  
{8, 6, 7}

### **Set Symmetric Difference**



Symmetric Difference of A and B is a set of elements in both A and B except those that are common in both.

Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric\_difference().

# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use ^ operator

# Output: {1, 2, 3, 6, 7, 8}

print(A ^ B)

Try the following examples on Python shell.

# use symmetric\_difference function on A  
>>> A.symmetric\_difference(B)  
{1, 2, 3, 6, 7, 8}  
  
# use symmetric\_difference function on B  
>>> B.symmetric\_difference(A)  
{1, 2, 3, 6, 7, 8}

## **Different Python Set Methods**

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with set objects.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [add()](https://www.programiz.com/python-programming/methods/set/add) | Add an element to a set |
| [clear()](https://www.programiz.com/python-programming/methods/set/clear) | Remove all elements form a set |
| [copy()](https://www.programiz.com/python-programming/methods/set/copy) | Return a shallow copy of a set |
| [difference()](https://www.programiz.com/python-programming/methods/set/difference) | Return the difference of two or more sets as a new set |
| [difference\_update()](https://www.programiz.com/python-programming/methods/set/difference_update) | Remove all elements of another set from this set |
| [discard()](https://www.programiz.com/python-programming/methods/set/discard) | Remove an element from set if it is a member. (Do nothing if the element is not in set) |
| [intersection()](https://www.programiz.com/python-programming/methods/set/intersection) | Return the intersection of two sets as a new set |
| [intersection\_update()](https://www.programiz.com/python-programming/methods/set/intersection_update) | Update the set with the intersection of itself and another |
| [isdisjoint()](https://www.programiz.com/python-programming/methods/set/isdisjoint) | Return True if two sets have a null intersection |
| [issubset()](https://www.programiz.com/python-programming/methods/set/issubset) | Return True if another set contains this set |
| [issuperset()](https://www.programiz.com/python-programming/methods/set/issuperset) | Return True if this set contains another set |
| [pop()](https://www.programiz.com/python-programming/methods/set/pop) | Remove and return an arbitary set element. Raise KeyError if the set is empty |
| [remove()](https://www.programiz.com/python-programming/methods/set/remove) | Remove an element from a set. If the element is not a member, raise a KeyError |
| [symmetric\_difference()](https://www.programiz.com/python-programming/methods/set/symmetric_difference) | Return the symmetric difference of two sets as a new set |
| [symmetric\_difference\_update()](https://www.programiz.com/python-programming/methods/set/symmetric_difference_update) | Update a set with the symmetric difference of itself and another |
| [union()](https://www.programiz.com/python-programming/methods/set/union) | Return the union of sets in a new set |
| [update()](https://www.programiz.com/python-programming/methods/set/update) | Update a set with the union of itself and others |

## **Other Set Operations**

### **Set Membership Test**

We can test if an item exists in a set or not, using the keyword in.

# initialize my\_set

my\_set = set("apple")

# check if 'a' is present

# Output: True

print('a' in my\_set)

# check if 'p' is present

# Output: False

print('p' not in my\_set)

### **Iterating Through a Set**

Using a for loop, we can iterate though each item in a set.

>>> for letter in set("apple"):  
... print(letter)  
...   
a  
p  
e  
l

### **Built-in Functions with Set**

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with set to perform different tasks.

|  |  |
| --- | --- |
| **Function** | **Description** |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all elements of the set are true (or if the set is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any element of the set is true. If the set is empty, return False. |
| [enumerate()](https://www.programiz.com/python-programming/methods/built-in/enumerate) | Return an enumerate object. It contains the index and value of all the items of set as a pair. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the set. |
| [max()](https://www.programiz.com/python-programming/methods/built-in/max) | Return the largest item in the set. |
| [min()](https://www.programiz.com/python-programming/methods/built-in/min) | Return the smallest item in the set. |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list from elements in the set(does not sort the set itself). |
| [sum()](https://www.programiz.com/python-programming/methods/built-in/sum) | Retrun the sum of all elements in the set. |

## **Python Frozenset**

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets.

Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable and can be used as keys to a dictionary.

Frozensets can be created using the function frozenset().

This datatype supports methods like copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric\_difference() and union(). Being immutable it does not have method that add or remove elements.

# initialize A and B

A = frozenset([1, 2, 3, 4])

B = frozenset([3, 4, 5, 6])

Try these examples on Python shell.

>>> A.isdisjoint(B)  
False  
>>> A.difference(B)  
frozenset({1, 2})  
>>> A | B  
frozenset({1, 2, 3, 4, 5, 6})  
>>> A.add(3)  
...  
AttributeError: 'frozenset' object has no attribute 'add'