**List Methods**

list.**append**(*x*)

Add an item to the end of the list. Equivalent to a[len(a):] = [x].

list.**extend**(*iterable*)

Extend the list by appending all the items from the iterable. Equivalent to a[len(a):] = iterable.

list.**insert**(*i*, *x*)

Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.**remove**(*x*)

Remove the first item from the list whose value is equal to *x*. It raises a ValueError if there is no such item.

list.**pop**([*i*])

Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.**clear**()

Remove all items from the list. Equivalent to del a[:].

list.**index**(*x*[, *start*[, *end*]])

Return zero-based index in the list of the first item whose value is equal to *x*. Raises a [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) if there is no such item.

The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

list.**count**(*x*)

Return the number of times *x* appears in the list.

list.**sort**(*key=None*, *reverse=False*)

Sort the items of the list in place (the arguments can be used for sort customization, see [sorted()](https://docs.python.org/3/library/functions.html#sorted) for their explanation).

list.**reverse**()

Reverse the elements of the list in place.

list.**copy**()

Return a shallow copy of the list. Equivalent to a[:].

**Set Methods**

**isdisjoint**(*other*)

Return True if the set has no elements in common with *other*. Sets are disjoint if and only if their intersection is the empty set.

**issubset**(*other*)

**set <= other**

Test whether every element in the set is in *other*.

**set < other**

Test whether the set is a proper subset of *other*, that is, set <= other and set != other.

**issuperset**(*other*)

**set >= other**

Test whether every element in *other* is in the set.

**set > other**

Test whether the set is a proper superset of *other*, that is, set >= other and set != other.

**union**(*\*others*)

**set | other | ...**

Return a new set with elements from the set and all others.

**intersection**(*\*others*)

**set & other & ...**

Return a new set with elements common to the set and all others.

**difference**(*\*others*)

**set - other - ...**

Return a new set with elements in the set that are not in the others.

**symmetric\_difference**(*other*)

**set ^ other**

Return a new set with elements in either the set or *other* but not both.

**copy**()[¶](https://docs.python.org/3.7/library/stdtypes.html#frozenset.copy)

Return a new set with a shallow copy of *s*.

**update**(*\*others*)

**set |= other | ...**

Update the set, adding elements from all others.

**intersection\_update**(*\*others*)

**set &= other & ...**

Update the set, keeping only elements found in it and all others.

**difference\_update**(*\*others*)

**set -= other | ...**

Update the set, removing elements found in others.

**symmetric\_difference\_update**(*other*)

**set ^= other**

Update the set, keeping only elements found in either set, but not in both.

**add**(*elem*)

Add element *elem* to the set.

**remove**(*elem*)

Remove element *elem* from the set. Raises [KeyError](https://docs.python.org/3.7/library/exceptions.html#KeyError) if *elem* is not contained in the set.

**discard**(*elem*)

Remove element *elem* from the set if it is present.

**pop**()

Remove and return an arbitrary element from the set. Raises [KeyError](https://docs.python.org/3.7/library/exceptions.html#KeyError) if the set is empty.

**clear**()

Remove all elements from the set.

**Dictionary Methods**

**clear**()

Remove all items from the dictionary.

**copy**()

Return a shallow copy of the dictionary.

*classmethod***fromkeys**(*seq*[, *value*])

Create a new dictionary with keys from *seq* and values set to *value*.

[fromkeys()](https://docs.python.org/3.7/library/stdtypes.html#dict.fromkeys) is a class method that returns a new dictionary. *value* defaults to None.

**get**(*key*[, *default*])

Return the value for *key* if *key* is in the dictionary, else *default*. If *default* is not given, it defaults to None, so that this method never raises a [KeyError](https://docs.python.org/3.7/library/exceptions.html#KeyError).

**items**()

Return a new view of the dictionary’s items ((key, value) pairs).

**keys**()

Return a new view of the dictionary’s keys.

**pop**(*key*[, *default*])

If *key* is in the dictionary, remove it and return its value, else return *default*. If *default* is not given and *key*is not in the dictionary, a [KeyError](https://docs.python.org/3.7/library/exceptions.html#KeyError) is raised.

**popitem**()

Remove and return a (key, value) pair from the dictionary. Pairs are returned in LIFO order.

[popitem()](https://docs.python.org/3.7/library/stdtypes.html#dict.popitem) is useful to destructively iterate over a dictionary, as often used in set algorithms. If the dictionary is empty, calling [popitem()](https://docs.python.org/3.7/library/stdtypes.html#dict.popitem) raises a [KeyError](https://docs.python.org/3.7/library/exceptions.html#KeyError).

*Changed in version 3.7:*LIFO order is now guaranteed. In prior versions, [popitem()](https://docs.python.org/3.7/library/stdtypes.html#dict.popitem) would return an arbitrary key/value pair.

**setdefault**(*key*[, *default*])

If *key* is in the dictionary, return its value. If not, insert *key* with a value of *default* and return *default*.*default* defaults to None.

**update**([*other*])

Update the dictionary with the key/value pairs from *other*, overwriting existing keys. Return None.

[update()](https://docs.python.org/3.7/library/stdtypes.html#dict.update) accepts either another dictionary object or an iterable of key/value pairs (as tuples or other iterables of length two). If keyword arguments are specified, the dictionary is then updated with those key/value pairs: d.update(red=1, blue=2).

**values**()

Return a new view of the dictionary’s values.