**List()**

**append(self, object, /)**

**Append object to the end of the list.**

**clear(self, /)**

**Remove all items from list.**

**copy(self, /)**

**Return a shallow copy of the list.**

**count(self, value, /)**

**Return number of occurrences of value.**

**extend(self, iterable, /)**

**Extend list by appending elements from the iterable.**

**index(self, value, start=0, stop=9223372036854775807, /)**

**Return first index of value.**

**Raises ValueError if the value is not present.**

**insert(self, index, object, /)**

**Insert object before index.**

**pop(self, index=-1, /)**

**Remove and return item at index (default last).**

**Raises IndexError if list is empty or index is out of range.**

**remove(self, value, /)**

**Remove first occurrence of value.**

**Raises ValueError if the value is not present.**

**reverse(self, /)**

**Reverse \*IN PLACE\*.**

**sort(self, /, \*, key=None, reverse=False)**

**Sort the list in ascending order and return None.**

**The sort is in-place (i.e. the list itself is modified) and stable (i.e. the**

**order of two equal elements is maintained).**

**If a key function is given, apply it once to each list item and sort them,**

**ascending or descending, according to their function values.**

**The reverse flag can be set to sort in descending order.**

**TUPLES()**

**count(self, value, /)**

**Return number of occurrences of value.**

**index(self, value, start=0, stop=9223372036854775807, /)**

**Return first index of value.**

**Raises ValueError if the value is not present.**

**DICTONARY()**

**clear(...)**

**D.clear() -> None. Remove all items from D.**

**copy(...)**

**D.copy() -> a shallow copy of D**

**get(self, key, default=None, /)**

**Return the value for key if key is in the dictionary, else default.**

**items(...)**

**D.items() -> a set-like object providing a view on D's items**

**keys(...)**

**D.keys() -> a set-like object providing a view on D's keys**

**pop(...)**

**D.pop(k[,d]) -> v, remove specified key and return the corresponding value.**

**If the key is not found, return the default if given; otherwise,**

**raise a KeyError.**

**popitem(self, /)**

**Remove and return a (key, value) pair as a 2-tuple.**

**Pairs are returned in LIFO (last-in, first-out) order.**

**Raises KeyError if the dict is empty.**

**setdefault(self, key, default=None, /)**

**Insert key with a value of default if key is not in the dictionary.**

**Return the value for key if key is in the dictionary, else default.**

**update(...)**

**D.update([E, ]\*\*F) -> None. Update D from dict/iterable E and F.**

**If E is present and has a .keys() method, then does: for k in E: D[k] = E[k]**

**If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v**

**In either case, this is followed by: for k in F: D[k] = F[k]**

**values(...)**

**D.values() -> an object providing a view on D's values**

**SET()**

**add(...)**

**Add an element to a set.**

**This has no effect if the element is already present.**

**clear(...)**

**Remove all elements from this set.**

**copy(...)**

**| Return a shallow copy of a set.**

**difference(...)**

**Return the difference of two or more sets as a new set.**

**(i.e. all elements that are in this set but not the others.)**

**difference\_update(...)**

**Remove all elements of another set from this set.**

**discard(...)**

**Remove an element from a set if it is a member.**

**Unlike set.remove(), the discard() method does not raise**

**an exception when an element is missing from the set.**

**intersection(...)**

**Return the intersection of two sets as a new set.**

**(i.e. all elements that are in both sets.)**

**intersection\_update(...)**

**Update a set with the intersection of itself and another.**

**isdisjoint(...)**

**Return True if two sets have a null intersection.**

**issubset(...)**

**Report whether another set contains this set.**

**issuperset(...)**

**Report whether this set contains another set.**

**pop(...)**

**Remove and return an arbitrary set element.**

**Raises KeyError if the set is empty.**

**remove(...)**

**Remove an element from a set; it must be a member.**

**If the element is not a member, raise a KeyError.**

**symmetric\_difference(...)**

**Return the symmetric difference of two sets as a new set.**

**(i.e. all elements that are in exactly one of the sets.)**

**symmetric\_difference\_update(...)**

**Update a set with the symmetric difference of itself and another.**

**union(...)**

**Return the union of sets as a new set.**

**(i.e. all elements that are in either set.)**

**update(...)**

**Update a set with the union of itself and others.**