The four important collection type objects in python are

1. List

2. Set

3. Tuple

4. Dictionary

List :

The most commonly used data structures in python is Lit.

List is a collection of elements can be of different data types.

Elements in list separated with comma(,) operator.

List elements can be processed using their index starting from 0.

>>> l1=[10,20,30]

>>> l2=[10, 'g' , 34.56 , "Python"]

>>> print(l1)

[10, 20, 30]

>>> print(l2)

[10, 'g', 34.56, 'Python']

>>> print(l1[1] , l2[3])

20 Python

Note : List is ordered collection and allow duplicates

>>> list = [10, 10, 10]

>>> print(list)

[10, 10, 10]

Functions of List:

>>> list = [10,20]

>>> print(list)

[10, 20]

append() : Add an element in the list

>>> list.append(30)

>>> list.append(40)

>>> print(list)

[10, 20, 30, 40]

insert() : insert an element @ specified location

>>> list.insert(2,100)

>>> print(list)

[10, 20, 100, 30, 40]

pop() : return last element in the list

(or) return element if we specify index

>>> print(list)

[10, 20, 100, 30, 40]

>>> list.pop()

40

>>> list.pop(1)

20

>>> print(list)

[10, 100, 30]

count() : returns count of specified element (duplicates)

>>> list = [10,20,10,40,10,50]

>>> list.count(10)

3

remove() : removes specified element in the given list

>>> list.remove(10)

>>> print(list)

[20, 10, 40, 10, 50]

copy() : removes a copy of specified list

>>> li = list.copy()

>>> print(li)

[20, 10, 40, 10, 50]

extend() : add all the elements of specified list at the end of this list

>>> print(list)

[20, 10, 40, 10, 50]

>>> print(li)

[20, 10, 40, 10, 50]

>>> li.extend(list)

>>> print(li)

[20, 10, 40, 10, 50, 20, 10, 40, 10, 50]

clear() : removes all the elements from the specified list

>>> print(list)

[20, 10, 40, 10, 50]

>>> list.clear()

>>> print(list)

[]

del : a pre-defined keyword is used to delete the objects(here list)

>>> del li

>>> print(li)

NameError: name 'li' is not defined

Indexing and Slicing :

We can access the elements either by using its index.

We can access a set of elements at a time using slicing concept.

>>> list = [10,20,30,40,50]

>>> list[2]

30

>>> list[2:4]

[30, 40]

>>> list[:4]

[10, 20, 30, 40]

>>> list[2:]

[30, 40, 50]

>>> list[-1]

50

>>> list[-2]

40

>>> list[-3:4]

[30, 40]

>>> list[-3:]

[30, 40, 50]

reverse() : reverse the list

sort() : sort the elements

>>> list=[10,20,30,40,50]

>>> list.reverse()

>>> print(list)

[50, 40, 30, 20, 10]

>>> list.sort()

>>> print(list)

[10, 20, 30, 40, 50]

List is mutable, Hence if we modify the list address will not be changed

>>> list=[10,20,30]

>>> id(list)

71777098760

>>> list.append(40)

>>> id(list)

71777098760

If list element can be modified and if it is immutable(int, str....), the address of element will be changed:

>>> list = [10,20,30]

>>> id(list[1])

1629338672

>>> list[1] = list[1]+30

>>> print(list)

[10, 50, 30]

>>> id(list[1])

1629339632

If list element is mutable and if we modify, the address will not be changed

>>> l1=[10,20,30]

>>> l2=[40,50]

>>> l3=[l1,l2]

>>> print(l3)

[[10, 20, 30], [40, 50]]

>>> l3[1]

[40, 50]

>>> id(l3[1])

71777021448

>>> l3[1].append(60)

>>> l3[1]

[40, 50, 60]

>>> id(l3[1])

71777021448

Nested list : Defining a list inside another list.

>>> list=[10,20,[30,40]]

>>> print(list[1])

20

>>> print(list[2])

[30, 40]

>>> print(list[2][0])

30

>>> print(list[2][1])

40

>>> list = [10,[20,[30,40],50],60]

>>> list[0]

10

>>> list[1][0]

20

>>> list[2]

60

>>> list[1][1]

[30, 40]

>>> list[1][1][1]

40

Tuple collection

**Tuple:**

* Tuple is a list of elements.
* Tuple can store heterogeneous elements.
* It is Similar to List.
* Tuple is Immutable, hence we cannot modify elements.

>>> tuple=(10,34.56,'python')

>>> print(tuple)

(10, 34.56, 'python')

Duplicates allowed into tuple:

>>> tuple=(10,20,30,10)

>>> print(tuple)

(10, 20, 30, 10)

Tuple can be created from another tuple elements or by the combination of more than one tuple collection

>>> t1 = (10,20,30)

>>> t2 = (40,50,60)

>>> t3=t1+t2

>>> print(t3)

(10, 20, 30, 40, 50, 60)

>>> t1=t1+t2

>>> print(t1)

(10, 20, 30, 40, 50, 60)

Tuple is Immutable , hence we cannot modify the tuple.

>>> tuple = (10,20,30)

>>> tuple.count(10)

1

>>> tuple.index(10)

0

Tuple elements can be mutable like list-collection.

>>> tuple = (10,[20,30])

>>> tuple[1].append(40)

>>> tuple[1]

[20, 30, 40]

Hence we cannot modify tuple element if it is immutable.

>>> print(tuple)

(10, 20, 30, 10)

>>> tuple[1]

20

>>> tuple[1] += 20

TypeError: 'tuple' object does not support item assignment

Set collection

* Set object is a collection of elements in Python.
* A set is an unordered collection.

Creating set:

* Set elements separated with comma(,) and surrounded with curly braces{ }
* We can create set also 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.).

>>> set={10,20,30,40,50}

>>> print(set)

{40, 10, 50, 20, 30}

* Set doesn’t allow duplicate elements.
* Main advantage of Set is eliminating duplicate entries.
* Every element is unique (no duplicates)

>>> set={10,20,30,10}

>>> print(set)

{10, 20, 30}

Set is mutable, hence we can modify set by adding and deleting the elements:

>>> set = {10,20,30,40}

>>> print(set)

{40, 10, 20, 30}

>>> set.add(50)

>>> print(set)

{40, 10, 50, 20, 30}

>>> set.pop()

40

>>> print(set)

{10, 50, 20, 30}

* But a set cannot have a mutable element, like [list](https://www.programiz.com/python-programming/list).
* Every element is unique (no duplicates) and must be immutable (which cannot be changed).
* However, the set itself is mutable. We can add or remove items from it.

>>> list=[10,20,30]

>>> set={10,20,list}

TypeError: unhashable type: 'list'

Update() : Takes the set of elements and add into this set.

>>> s = {10,20,30}

>>> s.update({40,50})

>>> print(s)

{40, 10, 50, 20, 30}

Removing elements :

Remove() : removes specified element and returns error if not present

Discard() : removes specified element and remains unchanged if element is not present.

**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.

>>> a = { }

>>> print(type(a))

<class 'dict'>

>>> a = set()

>>> print(type(a))

<class 'set'>

* Sets are mutable. But since they are unordered, indexing has no meaning.
* We cannot access or change an element of set using indexing or slicing.
* Set does not support indexing.

>>> my\_set = {10,20,30}

>>> print(my\_set)

{10, 20, 30}

>>> print(my\_set[0])

TypeError: 'set' object does not support indexing

Set operations:

>>> a = {1,2,3,4,5}

>>> b = {3,4,5,6,7}

>>> a&b # returns values present in both a and b

{3, 4, 5}

>>> a|b # returns values present either in a or b

{1, 2, 3, 4, 5, 6, 7}

>>> a^b

{1, 2, 6, 7}

>>> a-b # returns all the elements after removing combined elements of a and b

{1, 2}

Python Set Operations

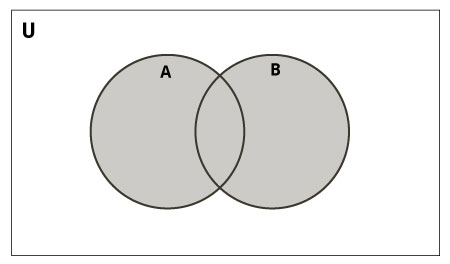
* Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference.
* We can do this with operators or methods.
* Let us consider the following two sets for the following operations.

>>> 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

print(A | B)

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

# use union function

>>> A.union(B)

{1, 2, 3, 4, 5, 6, 7, 8}

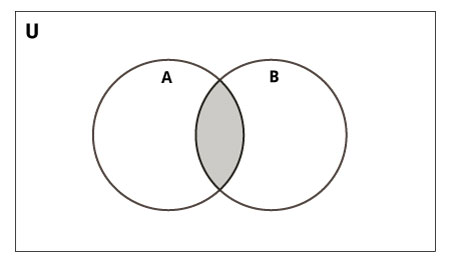
# use union function on B

>>> 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)

# use intersection function on A

>>> A.intersection(B)

{4, 5}

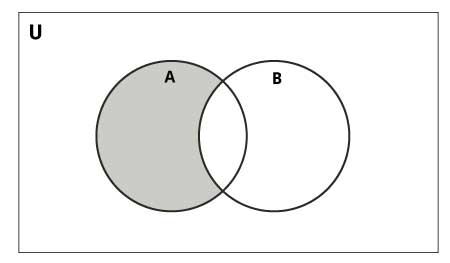
# use intersection function on B

>>> 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)

# use difference function on A

>>> A.difference(B)

{1, 2, 3}

# use - operator on B

>>> B - A

{8, 6, 7}

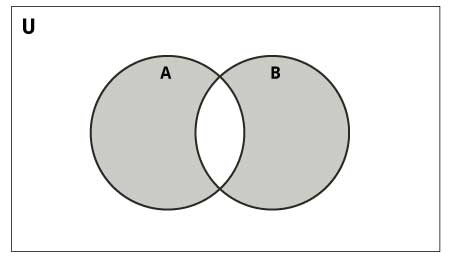
# use difference function on B

>>> 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)

# 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}

>>> del set

>>> s = set()

>>> type(s)

<class 'set'>

>>> a = {1,2,3,4,5}

>>> b = {3,4,5,6,7}

>>> a.union(b)

{1, 2, 3, 4, 5, 6, 7}

>>> a.difference(b)

{1, 2}

>>> a.intersection(b)

{3, 4, 5}

>>> a.symmetric\_difference(b)

{1, 2, 6, 7}

>>> a.difference\_update(b)

>>> print(a)

{1, 2}

>>> a = {1,2,3,4,5}

>>> b = {3,4,5,6,7}

>>> a.symmetric\_difference\_update(b)

>>> print(a)

{1, 2, 6, 7}

>>> print(b)

{3, 4, 5, 6, 7}

**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.

|  |  |
| --- | --- |
| **Python Set Methods** | |
| 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 KeyErrorif 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 |

**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

print('a' in my\_set)

# Output: True

# check if 'p' is present

print('p' not in my\_set)

# Output: False

**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, frozen sets 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])

>>> 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'

Dictionaries

* Another useful data type built into Python is the dictionary (see [Mapping Types — dict](https://docs.python.org/3/library/stdtypes.html#typesmapping)).
* Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.
* Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.
* You can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().
* It is best to think of a dictionary as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary).
* A pair of braces creates an empty dictionary: {}.
* Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.
* The main operations on a dictionary are storing a value with some key and extracting the value given the key.
* It is also possible to delete a key:value pair with del. If you store using a key that is already in use, the old value associated with that key is forgotten.
* It is an error to extract a value using a non-existent key.
* Performing list(d.keys()) on a dictionary returns a list of all the keys used in the dictionary, in arbitrary order (if you want it sorted, just usesorted(d.keys()) instead).
* To check whether a single key is in the dictionary, use the [in](https://docs.python.org/3/reference/expressions.html#in) keyword.

>>> d = {10:"One" , 20:"Two", 30:"Three"}

>>> print(d)

{10: 'One', 20: 'Two', 30: 'Three'}

>>> d.keys()

dict\_keys([10, 20, 30])

>>> d.items()

dict\_items([(10, 'One'), (20, 'Two'), (30, 'Three')])

>>> d.values()

dict\_values(['One', 'Two', 'Three'])

We cannot store duplicate keys, elemens will be replaced if we try:

>>> d = {10:20 , 10:30}

>>> print(d)

{10: 30}

We can store duplicate elements with unique keys

>>> d = {10:20 , 20:20}

>>> print(d)

{10: 20, 20: 20}

Keys must be immutable. Hence we cannot store list as key but tuple can be used.

>>> d = {10:"One", (20,30):"Two"}

>>> print(d.get(10)) # get returns the value associated with specified key.

One

>>> print(d.get("Two"))

None

Pop() : removes the element associated with specified key

>>> print(d.pop(10))

One

>>> print(d)

{(20, 30): 'Two'}

>>> d = {10,20,[30,40]}

TypeError: unhashable type: 'list'

Update() : we can update the list by adding one dictionary to another dictionary

>>> d = {10:"Two" , 20:"Two", 30:"Three"}

>>> d.update({40:"Four"})

>>> print(d)

{10: 'Two', 20: 'Two', 30: 'Three', 40: 'Four'}

Iterating elements:

>>> d = {10:"Two" , 20:"Two", 30:"Three"}

>>> for key in d:

print(key)

10

20

30

>>> for key in d:

print(d.get(key))

Two

Two

Three