10 Sets

"Chic and unique...."

Contents

What are Sets?

- Sets are like lists, with an exception that they do not contain duplicate entries.

```
a = set( )                    # empty set, use ( ) instead of { }
b = {20}                      # set with one item
c = {'Sanjay', 25, 34555.50}  # set with multiple items
d = {10, 10, 10, 10}          # only one 10 gets stored
```

- While storing an element in a set, its hash value is computed using a hashing technique to determine where it should be stored in the set.

- Since hash value of an element will always be same, no matter in which order we insert the elements in a set, they get stored in the same order.

```
s = {12, 23, 45, 16, 52}
t = {16, 52, 12, 23, 45}
u = {52, 12, 16, 45, 23}
print(s)                 # prints {12, 45, 16, 52, 23}
print(t)                 # prints {12, 45, 16, 52, 23}
print(u)                 # prints {12, 45, 16, 52, 23}
```

- It is possible to create a set of strings and tuples, but not a set of lists.

```
s1 = {'Morning', 'Evening'}             # works
s2 = {(12, 23), (15, 25), (17, 34)}     # works
s3 = {[12, 23], [15, 25], [17, 34]}     # error
```

  Since strings and tuples are immutable, their hash value remains same at all times. Hence a set of strings or tuples is permitted. However, a list may change, so its hash value may change, hence a set of lists is not permitted.

- Sets are commonly used for eliminating duplicate entries and membership testing.

Accessing Set Elements

- Entire set can be printed by just using the name of the set. Set is an unordered collection. Hence order of insertion is not same as the order of access.

```
s = {15, 25, 35, 45, 55}
print(s)                          # prints {35, 45, 15, 55, 25}
```

- Being an unordered collection, items in a set cannot be accessed using indices.

- Sets cannot be sliced using [ ].

Looping in Sets

- Like strings, lists and tuples, sets too can be iterated over using a **for** loop.

```
s = {12, 15, 13, 23, 22, 16, 17}
for ele in s :
    print(ele)
```

- Note that unlike a string, list or tuple, a **while** loop should not be used to access the set elements. This is because we cannot access a set element using an index, as in **s[i]**.

- Built-in function **enumerate( )** can be used with a set. The enumeration is done on access order, not insertion order.

Basic Set Operations

- Sets like lists are mutable. Their contents can be changed.

```
s = {'gate', 'fate', 'late'}
s.add('rate')                     # adds one more element to set s
```

- If we want an immutable set, we should use a **frozenset**.

```
s = frozenset({'gate', 'fate', 'late'})
s.add('rate')  # error
```

- Given below are the operations that work on lists and tuples. These operations are discussed in detail in Chapter 8. Try these operations on sets too.

Concatenation - doesn't work
Merging - doesn't work
Conversion - works
Aliasing - works
Cloning - works
Searching - works
Identity - works
Comparison - works
Emptiness - works

- Two sets cannot be concatenated using +.

- Two sets cannot be merged using the form **z = s + t**.

- While converting a set using **set( )**, repetitions are eliminated.

```
lst = [10, 20, 10, 30, 40, 50, 30]
s = set(lst)        # will create set containing 10, 20, 30, 40, 50
```

Using Built-in Functions on Sets

- Many built-in functions can be used with sets.

```
s = {10, 20, 30, 40, 50}
len(s)          # return number of items in set s
max(s)          # return maximum element in set s
min(s)          # return minimum element in set s
sorted(s)       # return sorted list (not sorted set)
sum(s)          # return sum of all elements in set s
any(t)          # return True if any element of s is True
all(t)          # return True if all elements of s are True
```

Note that **reversed( )** built-in function doesn't work on sets.

Set Methods

- Any set is an object of type **set**. Its methods can be accessed using the syntax **s.method( )**. Usage of commonly used set methods is shown below:

```
s = {12, 15, 13, 23, 22, 16, 17}
t = {'A', 'B', 'C'}
u = set ( )            # empty set
s.add('Hello')         # adds 'Hello' to s
s.update(t)            # adds elements of t to s
```

```
u = s.copy( )          # performs deep copy (cloning)
s.remove(15)           # deletes 15 from s
s.remove(101)          # would raise error, as 101 is not a member of s
s.discard(12)          # removes 12 from s
s.discard(101)         # won't raise an error, though 101 is not in s
s.clear( )             # removes all elements
```

- Following methods can be used on 2 sets to check the relationship between them:

```
s = {12, 15, 13, 23, 22, 16, 17}
t = {13, 15, 22}
print(s.issuperset(t))    # prints True
print(s.issubset(t))      # prints False
print(s.isdisjoint(t))    # prints False
```

Since all elements of **t** are present in **s**, **s** is a superset of **t** and **t** is subset of **s**. If intersection of two sets is null, the sets are called disjoint sets.

Mathematical Set Operations

- Following union, intersection and difference operations can be carried out on sets:

```
# sets
engineers = {'Vijay', 'Sanjay', 'Ajay', 'Sujay', 'Dinesh'}
managers = {'Aditya', 'Sanjay'}

# union - all people in both categories
print(engineers | managers)

# intersection - who are engineers and managers
print(engineers & managers)

# difference - engineers who are not managers
print(engineers - managers)

# difference - managers who are not engineers
print(managers - engineers)

# symmetric difference - managers who are not engineers
# and engineers who are not managers
print(managers ^ engineers)

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

```
b = {2, 4, 5}
print(a >= b)     # prints True as a is superset of b
print(a <= b)     # prints False as a is not a subset of b
```

Updating Set Operations

- Mathematical set operations can be extended to update an existing set.

```
a |= b          # update a with the result of a | b
a &= b          # update a with the result of a & b
a -= b          # update a with the result of a - b
a ^= b          # update a with the result of a ^ b
```

Set Varieties

- Unlike a list and tuple, a set cannot contain a set embedded in it.

```
s = {'gate', 'fate', {10, 20, 30}, 'late'}     # error, nested sets
```

- It is possible to unpack a set using the *operator.

```
x = {1, 2, 3, 4}
print(*x)         # outputs 1, 2, 3, 4
```

Programs

Problem 10.1

What will be the output of the following program?

```
a = {10, 20, 30, 40, 50, 60, 70}
b = {33, 44, 51, 10, 20,50, 30, 33}
print(a | b)
print(a & b)
print(a - b)
print(b - a)
print(a ^ b)
print(a >= b)
print(a <= b)
```

Output

```
{33, 70, 40, 10, 44, 50, 51, 20, 60, 30}
{10, 50, 20, 30}
{40, 60, 70}
{33, 51, 44}
{33, 70, 40, 44, 51, 60}
False
False
```

Problem 10.2

What will be the output of the following program?

```
a = {1, 2, 3, 4, 5, 6, 7}
b = {1, 2, 3, 4, 5, 6, 7}
c = {1, 2, 3, 4, 5, 6, 7}
d = {1, 2, 3, 4, 5, 6, 7}
e = {3, 4, 1, 0, 2, 5, 8, 9}
a |= e
print(a)
b &= e
print(b)
c -= e
print(c)
d ^= e
print(d)
```

Output

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
{1, 2, 3, 4, 5}
{6, 7}
{0, 6, 7, 8, 9}
```

Problem 10.3

Write a program to carry out the following operations on the given set

```
s = {10, 2, -3, 4, 5, 88}
```

- number of items in set s
- maximum element in set s
- minimum element in set s

- sum of all elements in set s
- obtain a new sorted set from s, set s remaining unchanged
- report whether 100 is an element of set s
- report whether -3 is an element of set s

Program

```
s = {10, 2, -3, 4, 5, 88}
print(len(s))
print(max(s))
print(min(s))
print(sum(s))
t = sorted(s)
print(t)
print(100 in s)
print(-3 not in s)
```

Output

```
6
88
-3
106
[-3, 2, 4, 5, 10, 88]
False
False
```

Problem 10.4

What will be the output of the following program?

Program

```
l = [10, 20, 30, 40, 50]
t = ('Sundeep', 25, 79.58)
s = 'set theory'
s1 = set(l)
s2 = set(t)
s3 = set(s)
print(s1)
print(s2)
print(s3)
```

Output

```
{40, 10, 50, 20, 30}
{25, 79.58, 'Sundeep'}
{'h', 's', 't', 'y', ' ', 'r', 'e', 'o'}
```

Exercises

[A] What will be the output of the following programs:

(a)
```
s = {1, 2, 3, 7, 6, 4}
s.discard(10)
s.remove(10)
print(s)
```

(b)
```
s1 = {10, 20, 30, 40, 50}
s2 = {10, 20, 30, 40, 50}
print(id(s1), id(s2))
```

(c)
```
s1 = {10, 20, 30, 40, 50}
s2 = {10, 20, 30, 40, 50}
s3 = {*s1, *s2}
print(s3)
```

(d)
```
s = set('KanLabs')
t = s[::-1]
print(t)
```

(e)
```
num = {10, 20, {30, 40}, 50}
print(num)
```

(f)
```
s = {'Tiger', 'Lion', 'Jackal'}
del(s)
print(s)
```

(g)
```
fruits = {'Kiwi', 'Jack Fruit', 'Lichi'}
fruits.clear( )
print(fruits)
```

(h)
```
s = {10, 25, 4, 12, 3, 8}
sorted(s)
print(s)
```

(i)
```
s = { }
t = {1, 4, 5, 2, 3}
```

```
print(type(s), type(t))
```

[B] Answer the following questions:

(a) A set contains names which begin either with A or with B. write a program to separate out the names into two sets, one containing names beginning with A and another containing names beginning with B.

(b) Create an empty set. Write a program that adds five new names to this set, modifies one existing name and deletes two names existing in it.

(c) What is the difference between the two set functions—**discard( )** and **remove( )**.

(d) Write a program to create a set containing 10 randomly generated numbers in the range 15 to 45. Count how many of these numbers are less than 30. Delete all numbers which are greater than 35.

(e) What do the following set operators do?

|, &, ^, ~

(f) What do the following set operators do?

|=, &=, ^=, -=

(g) How will you remove all duplicate elements present in a string, a list and a tuple?

(h) Which operator is used for determining whether a set is a subset of another set?

(i) What will be the output of the following program?

```
s = {'Mango', 'Banana', 'Guava', 'Kiwi'}
s.clear( )
print(s)
del(s)
print(s)
```

(j) Which of the following is the correct way to create an empty set?

```
s1 = set( )
s2 = { }
```

What are the types of **s1** and **s2**? How will you confirm the type?