```
        if i.count(j)!=1:
            flag=False
    if flag:
        print(i)
```

7.5. Множини

Множина - це структура даних, що містить невпорядкований набір унікальних елементів. Множини в мові Python досить подібні до множин у математиці. Використання множин є доцільним у тому випадку, коли присутність елемента в наборі важливіша порядку слідування елементів та кількості їх повторень.

Множина може містити елемененти різних типів, проте ці елементи можуть бути лише незмінюваних типів даних: числа, рядки, кортежі. Вимога незмінності елементів множини накладається особливостями подання множини в пам'яті комп'ютера.

Сама множина є змінюваним типом даних, тому до множин можна додавати нові та видаляти існуючі елементи. Як і у випадку математичних множин, у мові Python передбачено виконання операцій над множинами: об'єднання, перетину, різниці, симетричної різниці.

На відміну від списків, де елементи зберігаються у вигляді впорядкованої послідовності, у множинах порядок зберігання елементів невизначений. Це дозволяє виконувати операції типу "перевірити приналежність елемента множині" швидше, ніж просто перебираючи всі елементи множини.

Множини записуються, як перелік елементів, розділених комою та взятих в фігурні дужки: `{1, 2, 3, 'Hello'}`.

7.5.1. Задання множини

Задання порожньої множини виконується з використанням функції `set()`:

```
>>> a = set()
```

Використання «{}» призведе до створення порожнього словника:

```
>>> s={}
```

```
>>> type(s)
<class 'dict'>
```

Проте можна задати множину, перерахувавши її елементи, взяті в фігурні дужки «{}»:

```
>>> s1 = {1, 2, 3}
>>> s2 = {'a', 'b', 'c', 5}
```

Створення множини з інших структур даних

Множину можна отримати з елементів об'єкта, що може ітеруватися (діапазон, список, рядок, словник, кортеж, файл і т.д.), використавши функцію **`set([iterable])`**. Проте варто пам'ятати, що до множини будуть включені лише унікальні елементи.

```
>>> a = set('hello')
>>> a
{'e', 'o', 'l', 'h'}
>>> set(range(5)
{0, 1, 2, 3, 4}
```

7.5.2. Виконання дій над елементами множини

Як зазначалося раніше, можна перевірити приналежність деякого елемента до множини, використовуючи оператор in (`значення in ім'я_множини`).

```
>>> a = {1, 2, 3}
>>> 2 in a
True
>>> 2 not in a
False
```

Відповідно для перебору всіх елементів множини необхідно скористатися циклом for:

```
s = {2, 5, 7, 11, 4}
for num in s:
    print(num)
```

7.5.3. Порівняння множин

Множини можна порівнювати між собою. Порівняння множин зводиться до перевірки, чи є множини рівними, або чи є певна множина підмножиною іншої.

```
>>> a={1,2,3}
>>> b={1,2,3}
>>> d={1,2,3,4}
>>> c={4,5,6}
```

set == other. Перевірка, чи множини set та other рівні. Повертає True, якщо всі елементи множини set належать множині other, і всі елементи множини other належать множині set, інакше – False.

```
>>> a == b
True
>>> a == d
False
```

set != other. Перевірка, чи є множини set та other не рівними. Повертає True, якщо приаймні один елемент множини set не належать множині other, або приаймні один елемент множини other не належать множині set, інакше – False.

```
>>> a != b
False
>>> a != d
True
```

set <= other. Перевірка, чи є множина set підмножиною множини other. Повертає True, якщо всі елементи множини set належать множині other, інакше – False.

```
>>> a<=d
True
>>> a<=b
True
```

set < other. Повертає True, якщо всі елементи множини set належать множині other, але не всі елементи множини other належать множині set, інакше – False.

```
>>> a<d
True
>>> a<b
False
```

set.isdisjoint(other). Повертає `True`, якщо множини `set` і `other` не мають спільних елементів, інакше – `False`.

```
>>> a.isdisjoint(c)
True
>>> c.isdisjoint(d)
False
```

set.issubset(other). Перевірка, чи є множина `set` підмножиною множини `other`. Аналогічно до `set <= other`.

set.issuperset(other). Перевірка чи є множина `other` підмножиною множини `set`. Аналогічно до `set >= other`.

7.5.4. Методи множин

set.add(x). Додає елемент `x` до множини `set`.

```
>>> s = {1, 2, 3}
>>> s.add(4)
>>> print(s)
{1, 2, 3, 4}
```

set.remove(x). Вилучає елемент `x` із множини `set`. Якщо такого елемента в множині немає, то виникає виняток `KeyError`.

set.discard(x). Вилучає елемент `x` із множини `set`. Якщо такого елемента в множині немає, то нічого не відбувається

set.pop(). Вилучає «перший» елемент з множини `set` та повертає його значення, як результат виконання функції. Так як множина – це невпорядкований набір, то не можна бути впевненим який з елементів буде взятий як перший. Якщо множина порожня, то виникає виняток `KeyError`.

set.clear(). Очищує множину `set` (вилучає всі елементи з множини).

Операції з множинами

```
>>> d={1,2,3,4}
>>> c={4,5,6}
```

set.union(*other) або **set | other | ...**. Повертає об'єднання множин `set` і `other`. Множина-результат буде містити як елементи множини `set`, так і елементи множини `other`.

```
>>> d | c
{1, 2, 3, 4, 5, 6}
```

set.intersection(*other) або **set & other & ...**. Повертає перетин множин `set` і `other`. Множина-результат буде містити елементи, які належать як множині `set`, так і множині `other`.

```
>>> d & c
{4}
```

set.difference(*other) або **set - other - ...**. Повертає різницю множин `set` і `other`. Множина-результат буде містити елементи множини `set`, які не належать множині `other`.

```
>>> d - c
{1, 2, 3}
>>> c - d
{5, 6}
```

set.symmetric_difference(*other) або **set ^ other**. Повертає симетричну різницю множин `set` і `other`. Множина-результат буде містити елементи, які належать множинам `set` та `other`, але не належать обом множинам. Аналогічно до `(set | other)-( set & other)`.

```
>>> d^c
{1, 2, 3, 5, 6}
```

Як можна бачити, вказані методи не призводять до зміни множини, а за ними відбувається формування нової множини. Проте для множин передбачені методи, шо будуть призводити до зміни початкової множини.

set.update(*other) або **set |= other.** Додає до множини `set` всі елементи множини `other`. Множина `set` буде містити об'єднання множин `set` і `other`.

set.intersection_update(*other) або **set &= other.** Вилучає з множини set всі елементи, які не входять до множинаи other. Множина set буде містити перетин множин set і other.

set.difference_update(*other) або **set -= other.** Вилучає з множини set всі елементи, які входять до множини other. Множина set буде містити різницю множин set і other.

set.symmetric_difference_update(other) або **set ^= other.** Множина set буде містити симетричну різницю множин set і other.

7.5.5. Приклади розв'язування задач

Приклад. Дано два рядки. Написати програму за якою буде створено дві множини (перша з символів першого рядка, друга з символів другого рядка) та виведено перетин, об'єднання, різницю цих множин.

```
st1=input('Введіть 1-й рядок')
st2=input('Введіть 2-й рядок')
s1=set(st1)
s2=set(st2)
o=s1|s2
p=s1&s2
r1=s1-s2
r2=s2-s1
print('Множина 1:',s1)
print('Множина 2:',s2)
print("Обє'днання 1:",o)
print('Перетин 1:',p)
print('Різниця між 1-ю і 2-ю множинами:',r1)
print('Різниця між 2-ю і 1-ю множинами:',r2)
```

Приклад. Дано рядок, який складається із слів розділених пропусками та розділовими знаками (,.!?). Написати програму за якою буде виведено всі слова що не містять повторюваних символів.

```
s=input('Введіть рядок: ')
a=list(s.split(' '))
a=[i.strip(',.!?') for i in a if i!='']
for i in a:
    if len(i)==len(set(i)):
        print(i)
```

Останій цикл можна переписати в вигляді генератору спсику:

```
[print(i) for i in a if len(i)==len(set(i))]
```

8. Функції

У практиці програмування часто виникає ситуація, коли одну й ту саму групу операторів, які реалізують певну частину загальної задачі, треба повторити без змін в різних місцях програми. Для розв'язання цієї задачі використовується концепція підпрограм, яка отримала широке розповсюдження практично у всіх мовах програмування [7]. Так, наприклад, у мові Pascal підпрограми поділяються на процедури та функції, в мовах C та Python розглядаються лише функції. Також підпрограми (функції) використовуються для спрощення сприйняття коду за рахунок розбиття програми на менші, логічно завершені підпрограми.

Функції в Python практично нічим не відрізняються від функцій в інших мовах програмування. ***Функцією*** називають іменований фрагмент програмного коду, до якого можна звернутися з іншого місця програми скільки завгодно разів [7] за її іменем. Єдиним виключенням, в Python, є lambda-функції, у яких немає імені. Тому в загальному можна зазначити, що функція - це така частина коду, яка ізольована від решти програми і виконуються тільки тоді, коли викликається.

Раніше йшлося про вбудовані функції мови Python: `sqrt()`, `len()`, `print()` та ін. Як можна було бачити, функції можуть приймати аргументи (нуль, один або кілька), і можуть повертати деякий результат (значення). Наприклад, функція `sqrt()` приймає один аргумент і повертає значення (корінь числа). Функція `print()` приймає змінне число аргументів і нічого не повертає.