```
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.

При описі власних функцій потрібно уникати так званого побічного ефекту, тобто виконання функцією дій, відмінних від її основного призначення. Наприклад, у випадку обчислювального характеру функції не варто розміщувати в функції оператори виведення певних даних.

8.1. Опис та виклик функцій

Функція складається з заголовку та тіла функції. Заголовок функції відповідає за її опис, тобто іменування та вказання необхідних параметрів. Тіло ж функції відповідає за самі дії, які мають виконатися при виклику функції. Заголовок та тіло функції оформлюються, як основна та вкладена інструкції.

Заголовок функції починається зі службового слова `def`, після якого вказується ім'я функції та список її формальних параметрів у круглих дужках. Завершується заголовок функції символом двокрапка «:». З наступного рядка слідує тіло функції, яке може містити довільну кількість операторів, що записуються з однаковим відступом на початку рядків по відношенню до заголовку функції.

Формат опису функції:

```
def ім'я_функції(список формальних параметрів):
    оператори тіла функції
```

Ім'я_функції – ідентифікатор, що формується за правилами створення ідентифікаторів і в подальшому буде використовуватися для виклику функції.

Формальні параметри – це назви змінних, що вказуються при описі функції і через які будуть передаватися дані з основної програми в функцію. Значення для формальних параметрів будуть вказуватися при виклику функції.

Список формальних параметрів записується у вигляді перерахованих через кому імен змінних. Як було зазначено раніше, функція може і не мати формальних параметрів.

Найпростіша функція, за якою виводиться привітання, буде мати наступний вигляд:

```
def func(name):
    print('Hello',name)
```

У випадку, якщо тіло функції має єдиний оператор, як у нашому випадку, то функцію можна записати одним рядком:

```
def funcs(name): print('Hello', name)
```

Як можна бачити, ця функція має єдиний параметр `name`, який використовується як частина повідомлення для виведення на екран.

Для виклику функції вказується її ім'я з указаним в дужках списком фактичних параметрів.

Формат оператора виклику функції:

```
ім'я_функції([список фактичних параметрів]);
```

Виклик раніше описаної функції матиме вигляд:

```
>>> func('World')
Hello World
```

Фактичні параметри – це назви змінних чи конкретні значення, що вказуються при виклику функції.

Список фактичних параметрів записується у вигляді перерахованих через кому фактичних параметрів.

Формальні параметри інколи називають параметрами функції, фактичні параметри інколи називають аргументами функції. Відповідні фактичні та формальні параметри не обов'язково повинні мати однакові імена.

В загальному кількість фактичних параметрів повинна бути такою самою, як і кількість формальних параметрів. Це необхідно тому, що при виклику функції встановлюється взаємно однозначна відповідність між фактичними та формальними параметрами. Інколи ще говорять, що співставлення параметрів є позиційним, тобто значення фактичних параметрів присвоюються формальним параметрам відповідно до їх позиції. Проте в мові Python є особливості співставлення цих параметрів, які будуть розглянуті пізніше.

Наявність параметрів у функціях надає можливість програмісту робити функції більш гнучкими та універсальними. Параметри виступають вхідними чи допоміжними даними, які будуть використані в обчисленнях, передбачених функцією.

Для того, щоб функцією було повернуте певне значення, її тіло повинно містити інструкцію `return`. Інструкція `return` являє собою

службове слово `return`, після якого вказується значення чи змінна, значення якої потрібно повернути за функцією.

Наприклад, функція, за якою буде повернуто одиницю:

```
def one():
    return 1
```

Якщо за функцією не передбачено повернення ніяких значень, функція може викликатися у головній програмі чи іншій функції як окремий оператор: `func('World')`.

Якщо за функцією передбачене повернення певного значення, то функція може викликатися у головній програмі чи іншій функції як операнд виразу: `two= one()+one()`.

Проте варто зауважити, що у випадку відсутності в тілі функції інструкції `return`, тобто не передбачення за функцією повернення ніякого значення, за функцією все рівно буде повернуте значення None.

```
>>> def func(): print('Hello!')
>>> print(func())
Hello!
None
```

Інколи виникає необхідність повернення за функцією не одного значення, а двох чи більше значень. Для цього потрібно в інструкції `return` вказати список або кортеж з бажаної кількості значень:

```
def f1(a, b): return [a**2, b**2]
def f2(a, b): return (a**2, b**2)
```

Тоді виклик функції можна записати:

```
n1, m1 = f1(2, 3)
n2, m2 = f2(2, 3)
s1 = f1(2, 3)
s2 = f2(2, 3)
```

В результаті виклику і виконання функцій отримаємо: n1 та n2 будуть містити значення 4, m1 та m2 будуть містити значення 9, s1 буде містити список [4, 9], s2 буде містити кортеж (4, 9).

В загальному, тіло функції може містити безліч інструкцій `return`, повернення значення за функцією буде визначатися першим викликом інструкції `return`.

Приклад. Написати функцію, за якою буде обчислюватися сума двох аргументів:

```
>>> def summa(a, b): return a+b
>>> summa(2, 3)
5
>>> x=5;
>>> y=6;
>>> summa(x, y)
11
```

Зважаючи на динамічну типізацію змінних в мові Python, аргументами однієї і тієї ж функції можуть бути значення чи змінні різного типу, наприклад:

```
>>> summa('Hello ','world.')
'Hello world.'
>>> summa([2,3,4],[4,5,6])
[2, 3, 4, 4, 5, 6]
```

Проте при такому багатогранному використанні функцій варто пам’ятати про строгу типізацію мови Python і неможливість проведення операцій у виразах з даними різних несумісних типів. Так, наступний виклик функції summa('Hello ',5) призведе до виникнення винятку TypeError (unsupported operand type(s) for +: 'int' and 'str').

8.2. Розширене використання параметрів та аргументів

В мові Python передбачена досить гнучка можливість використання параметрів, а саме: можливість вказання для параметрів значень за замовчуванням; використання параметрів, до яких можна звернутися за їх назвою (ключових аргументів); створення функцій, що будуть приймати змінну кількість аргументів; вказання обов’язковості ключових аргументів.

8.2.1. Значення параметрів за замовчуванням

При описі функцій інколи виникають випадки, що деякий параметр при виклику функції в більшості випадків буде мати одне й те саме значення, але незважаючи на це, все ж можуть бути випадки виклику цієї ж функції, з

іншим значенням даного параметру. Наприклад, функція `print()` має параметр `sep`, який досить часто не вказується при її виклику, а значенням для нього в такому випадку встановлюється символ пропуску. Таке значення для параметру називається значенням за замовчуванням, а сам параметр стає необов'язковим.

Для вказання того, що параметр матиме значення за замовчуванням, необхідно при описі функції після імені цього параметру поставити знак присвоєння (=) і вказати відповідне значення. Значення за замовчуванням має бути незмінюваним.

Наприклад:

```
>>> def summa(a, b=2): return a+b
>>> summa(4, 6)
10
>>> summa(4)
6
```

В першому випадку при виклику функції вказані обидва аргументи (фактичні параметри), тому параметру `a` буде надане значення `4`, параметру `b` – `6` (сума рівна 10). В другому випадку виклик функції містить лише один аргумент, тому параметру `a` буде надане значення `4` (це перший параметр, а відповідно йому надається значення першого аргументу), другому ж параметру буде надане значення `2`, яке є його значенням за замовчування.

Параметрів зі значенням за замовчуванням у списку параметрів може бути будь-яка кількість. При виклику функції значення вказаних аргументів будуть надаватися параметрам функції послідовно. Це пов'язано з тим, що за замовчуванням аргументи є позиційними, тобто значення аргументів присвоюються параметрам відповідно до їх позиції.

```
>>> def func(a, b=2, c=10):
     print('a=',a,'b=',b,'c=',c)
>>> func(3, 4, 5)
a= 3 b= 4 c= 5
>>> func(3, 4)
a= 3 b= 4 c= 10
>>> func(3)
a= 3 b= 2 c= 10
```

Параметри зі значенням за замовчуванням у списку параметрів не можуть передувати параметрам без значення за замовчуванням. Тобто значеннями за замовчуванням можуть бути забезпечені тільки параметри, які знаходяться в кінці списку параметрів. Наприклад, припустимо, що є функція, описана наступним чином: `def summa(a=2, b): return a+b`. При її виклику `summa(3)`, незважаючи на те, що параметр `a` має значення за замовчуванням, йому буде надане значення першого аргументу (тобто, `3`), а для параметру `b` аргументу не вистачить, відповідно функція не зможе бути викликана.

8.2.2. Ключові аргументи

Окрім позиційних аргументів, коли значення параметрів встановлюються у відповідності зі слідуванням аргументів, у мові Python передбачене використання так званих ключових аргументів. Ключовий аргумент – це аргумент, який записується з указанням імені параметра, для якого він призначений ("ім'я_параметра = значення").

Наприклад, є деяка функція з параметрами, що мають значення за замовчуванням. А при виклику цієї функції необхідно вказати значення тільки для деяких параметрів. В такому випадку якраз і стане у нагоді використання ключових аргументів. При виклику функції можна буде вказати лише значення для тих параметрів, яким не підходять їх значення за замовчуванням.

Для задання ключового аргументу необхідно вказати ім'я параметра та після знаку присвоєння необхідне значення.

```
>>> def func(a, b=2, c=10):
    print('a=',a,'b=',b,'c=',c)
>>> func(3, c=15)
a= 3 b= 2 c= 15
>>> func(3, c=15, b=8)
a= 3 b= 8 c= 15
```

Аргумент може бути ключовим не лише для параметра зі значенням за замовчуванням. Проте в такому випадку такий ключовий аргумент має бути обов'язково присутній при виклику функції.

```
>>> func(c=22, a=23)
a= 23 b= 2 c= 22
```

Можна виділити декілька переваг використання ключових аргументів: по-перше, використання функції стає легшим, оскільки немає необхідності відстежувати порядок аргументів; по-друге, можна задавати значення тільки деяким параметрам, за умови, що решта параметрів мають значення за замовчуванням.

Проте в версії 3.8 з'явилася можливість обмеження, щодо використання ключових аргументів. Якщо між параметрами функції розмістити символ «/» то параметри які йдуть до нього будуть лише позиційними, а параметри які йдуть за ним можуть бути ключовими.

```
>>> def func(a, b=2, /, c=10):
     print('a=',a,'b=',b,'c=',c)
>>> func(3, 4, 5)
a= 3 b= 4 c= 5
>>> func(3, 4, c=5)
a= 3 b= 4 c= 5
>>> func(3, b=4, c=5)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    func(3, b=4, c=5)
TypeError: func() got some positional-only arguments
     passed as keyword arguments: 'b'
```

8.2.3. Змінна кількість аргументів

Іноді буває необхідно визначити функцію, що може бути викликана з будь-якою кількістю аргументів, тобто здатну приймати змінну кількість аргументів. Наприклад, функція `print()` може приймати декілька (одне, два, три і т.д.) значень, що мають бути виведені, чи функція `min()` може приймати декілька (два, три і т.д.) значень для визначення мінімального з них. Така можливість реалізується за рахунок того, що під час виклику функції змінна кількість аргументів буде зібрана (упакована) в одну змінну типу кортеж або словник та передана в функцію.

Змінна кількість аргументів може застосовуватися і для позиційних, і для ключових аргументів. Окрім того, при описі функції можуть бути присутні параметри як для одиничних аргументів, так і для змінної кількості аргументів, головне щоб опис параметрів для одиничних аргументів передували параметрам для змінної кількості аргументів.

Для вказання того, що функція зможе приймати змінну кількість позиційних аргументів, при її описі необхідно вказати параметр, перед яким поставити знак «*». Наприклад, для функції:

```
def summa(first=0, *param):
    print('first=', first, '; param=', param, sep='')
    s=first
    for number in param:
        s+=number
    return s
```

можливі різні варіанти її виклику:

```
>>> print('summa=', summa())
first=0; param=()
summa= 0
>>> print('summa=', summa(5,3,4))
first=5; param=(3, 4)
summa= 12
>>> print('summa=', summa(5,3,4,5,6,7,8))
first=5; param=(3, 4, 5, 6, 7, 8)
summa= 38
```

Отже, можна бачити, що всі позиційні аргументи, починаючи з другого, збираються в кортеж під ім'ям `param`.

Для вказання того, що функція зможе приймати змінну кількість ключових аргументів, при її описі необхідно вказати параметр, перед яким поставити два знаки «**». Наприклад, для функції:

```
def summa(first=0, **param):
    print('first=', first, '; param=', param, sep='')
    s=first
    for key in param:
        s+=param[key]
```

```
    return s
```

можливі різні варіанти її виклику:

```
>>> print('summa=', summa(2, s1=3, s2=4))
first=2; param={'s1': 3, 's2': 4}
summa= 9
>>> print('summa=', summa(s1=3, s2=4))
first=0; param={'s1': 3, 's2': 4}
summa= 7
```

Отже, можна бачити, що всі ключові аргументи, починаючи з другого, зібраються в словник під ім'ям `param`.

Розглянуті випадки можуть бути використані одночасно:

```
def summa(first=0, *param1, **param2):
    print('first=', first)
    print('param1=', param1)
    print('param2=', param2)
    s=first
    for number in param1:
        s+=number
    for key in param2:
        s+=param2[key]
    return s
>>> print('summa=', summa(3,4,5,6))
first= 3
param1= (4, 5, 6)
param2= {}
summa= 18
>>> print('summa=', summa(s1=3, s2=4))
first= 0
param1= ()
param2= {'s1': 3, 's2': 4}
summa= 7
>>> print('summa=', summa(3,4,5,6,s1=3, s2=4))
first= 3
param1= (4, 5, 6)
```

```
param2= {'s1': 3, 's2': 4}
summa= 25
```

8.2.4. Обов'язкові ключові аргументи

Інколи є необхідність опису функцій, в яких певні параметри мають бути доступні лише за ключовими аргументами, що забезпечить більшу зрозумілість її параметрів. Наприклад, функція `print()` має параметри `sep` та `end`, вказати значення для яких можна лише з використанням ключових аргументів.

Для визначення параметра, який зможе приймати значення тільки від ключового аргументу, його необхідно оголосити після параметра з зірочкою (параметра, що буде приймати змінну кількість аргументів).

```
def summa(first=0, *param, mult):
    print('first=', first, '; param=', param, sep='')
    print(mult=', mult)
    s=first
    for number in param:
        s+=number
    s*=mult
    return s
>>> print('result=', summa(1,2,3, mult=5))
first=1; param=(2, 3)
mult= 5
result= 30
>>> print('result=', summa(1,2,3,5))
```

У разі виконання оператора `print('result=', summa(1,2,3,5))`, отримаємо помилку, оскільки параметра `mult` не буде надане значення, так як всі аргументи, починаючи з другого, будуть зібрані в кортежі `param`.

Якщо за функцією не передбачається наявність змінної кількості аргументів, але є необхідність в параметрі лише за ключовими аргументами, то заголовок такої функції можна визначити так:

```
def summa(first=0, *, mnog):
```