Формат ввода

Вводятся строки вида:
Описание - чьё оно

Формат вывода

Для каждого героя с новой строки вывести все описания, которые относятся к нему, без повторений, в любом порядке в виде:
Герой: описание 1; описание 2; ...

Ввод

blue nose – master
Cherry grey wig - master Cherry
an old battered jacket - Geppetto
a hammer and a plane - Geppetto
yellow wig – Geppetto
blue hair - Fairy blue nose - master Cherry

Вывод

master Cherry: blue nose; grey wig Geppetto: yellow wig; an old battered jacket; a hammer and a plane Fairy: blue hair

Ввод

black, disheveled beard - Mangiafuoco
paper jacket - Pinocchio
long beard – Mangiafuoco
mouth huge as a stove - Mangiafuoco
long nose - Pinocchio
a thick whip of snakes - Mangiafuoco
smile from ear to ear - Arlecchino
eyes like flashlights - Mangiafuoco
tears in the eyes - Pierrot
long beard – Mangiafuoco
bread crumb cap - Pinocchio

Вывод

Mangiafuoco: mouth huge as a stove; eyes like flashlights; long beard; a thick whip of snakes; black, disheveled beard Pinocchio: paper jacket; bread crumb cap; long nose Arlecchino: smile from ear to ear Pierrot: tears in the eyes

```
import sys


s = {}
for i in sys.stdin.readlines():
    name = i[i.find('-') + 2:].strip()
    about = i[:i.find('-') - 1].strip()
    if name not in s:
        s[name] = []
    if about not in s[name]:
        s[name].append(about)

for i in s:
    print(i + ':', '; '.join(s[i]))
```

Функции **power_of_theater(text, divisor_1, divisor_2, *args, **kwargs)** передаются аргументы:

- текст;
- первый разделитель, по которому текст разбивается и превращается в список;
- второй разделитель, по которому разбиваются все элементы первого списка;
- произвольное количество кортежей, в которых записаны координаты элемента в двумерном списке и имя функции, с помощью которой этот элемент нужно преобразовать;
- произвольное количество именованных аргументов – функций для преобразования.

Если в кортежах встречается функция, которая не определена в именованных пераметрах, то с этим элементом никаких преобразований не производится.

Функция возвращает преобразованный двумерный список.

```
print(*power_of_theater( 'At the sudden appearance of the theater owner, everything was numb, no one dared to sigh loudly, you could hear a fly flying.', ', ', ' ', (1, 0, 'is_up'), (1, 2, 'turn'), (0, 2, 'slice'), slice=lambda x: x[::2], turn=lambda x: x[::-1], up=lambda x: x.upper(), is_up=lambda x: str(x.isupper()) ), sep='\n')
```

```
['At', 'the', 'sde', 'appearance', 'of', 'the', 'theater', 'owner']
['False', 'was', 'bmun']
['no', 'one', 'dared', 'to', 'sigh', 'loudly']
['you', 'could', 'hear', 'a', 'fly', 'flying.']
```

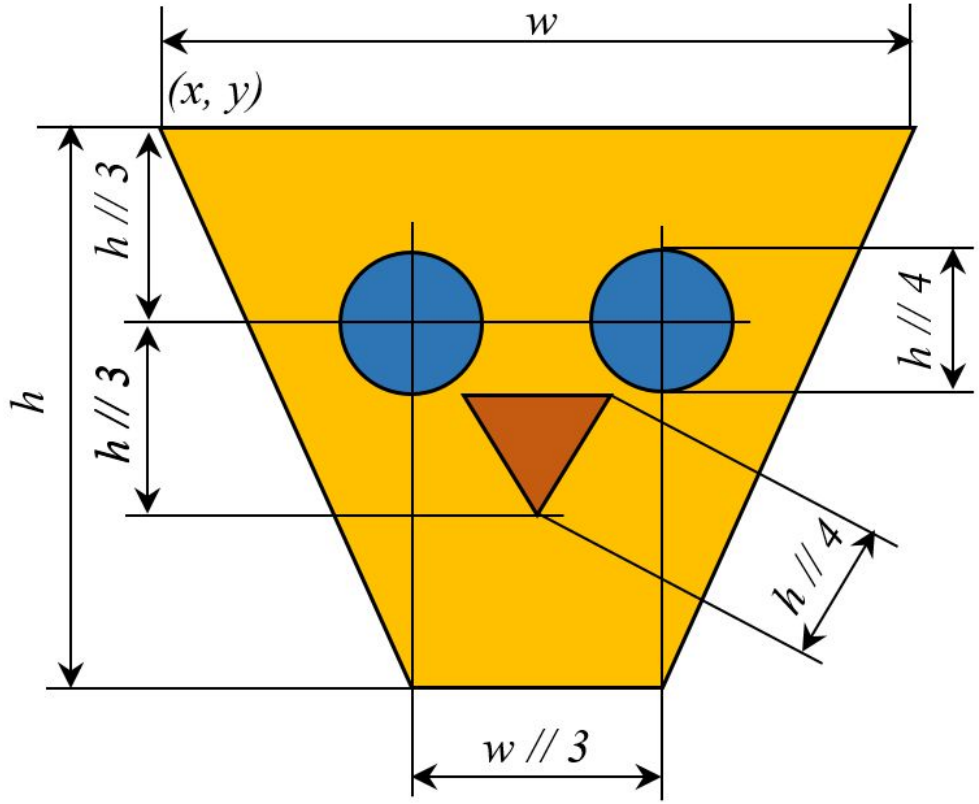
Ввод	Вывод
print(*power_of_theater('It was undoubtedly an exciting performance.\nBut the audience in the hall lost patience.\nThey wanted to see the Comedy continue.\nAnd they began to shout.', '\n', ' ', (0, 3, 'up'), (1, 0, 'mult'), (2, 1, 'turn'), (3, 2, 'abracodabra'), mult=lambda x: x * 2, turn=lambda x: x[::-1], up=lambda x:	['It', 'was', 'undoubtedly', 'AN', 'exciting', 'performance.'] ['ButBut', 'the', 'audience', 'in', 'the', 'hall', 'lost', 'patience.'] ['They', 'detnaw', 'to', 'see', 'the', 'Comedy', 'continue.'] ['And', 'they', 'began', 'to', 'shout.']

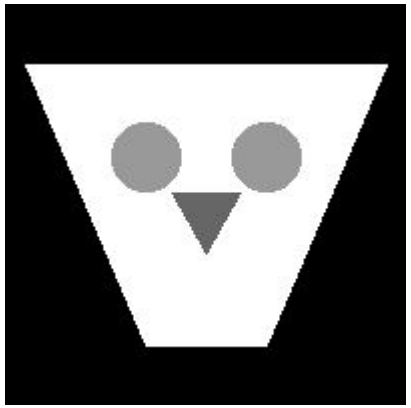
```
def power_of_theater(text, divisor_1, divisor_2, *args, **kwargs):
    new_text1 = text.split(divisor_1)
    s = []
    for i in range(len(new_text1)):
        s.append(new_text1[i].split(divisor_2))
    for i in args:
        if i[2] in kwargs:
            s[i[0]][i[1]] = kwargs[i[2]](s[i[0]][i[1]])
    return s
```

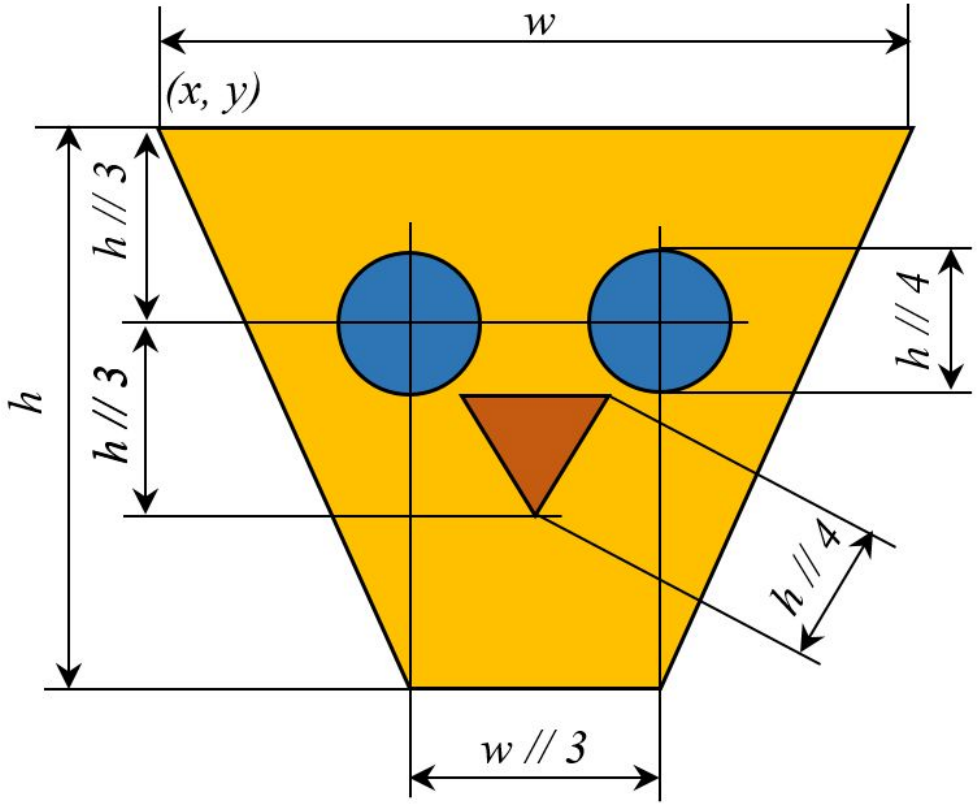
Напишите класс **MaskImageDraw(ImageDraw.ImageDraw)**, в котором есть метод **mask(xy, fill)** для рисования маски, где

- **xy** — кортеж из 4 значений: *(x, y)* — координаты верхнего левого угла рисунка, *w* и *h* — соответственно ширина и высота;
- **fill** — кортеж из 3-х цветов в 16-ричном формате соответственно маски, глаз и носа.

Маска представляет собой трапецию с глазами в виде окружностей и носом в виде равностороннего треугольника, размеры и расположение которых указаны на рисунке.
Метод рисует маску и ничего не возвращает.

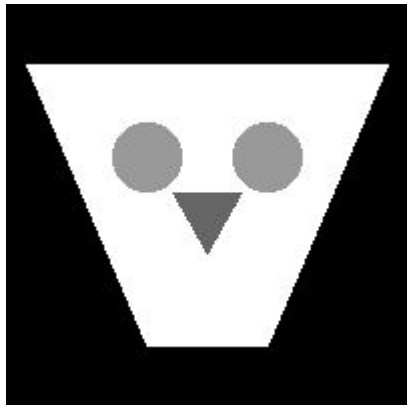


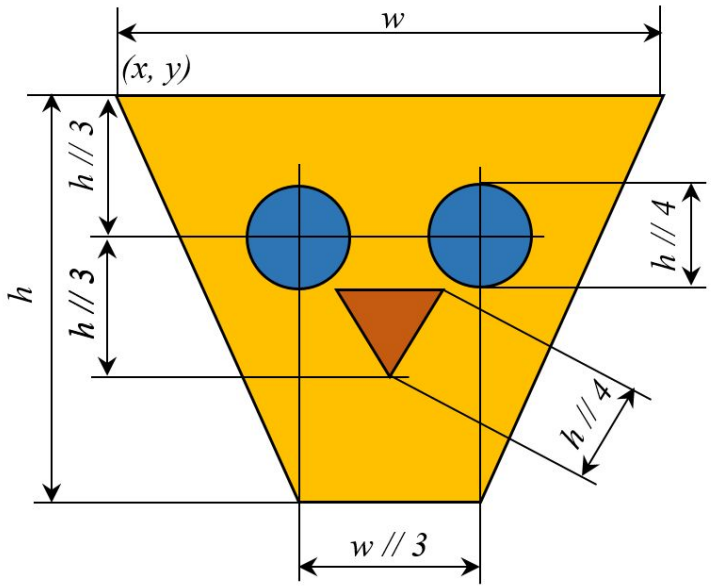





```
from PIL import Image

img = Image.new('RGB', (200, 200), '#000000')
drw = MaskImageDraw(img)
drw.mask((10, 30, 180, 140), ('#ffffff', '#999999', '#666666'))
img.save('result.png')
```






```
from PIL import ImageDraw, Image
import math


class MaskImageDraw(ImageDraw.ImageDraw):
    def mask(self, xy, fill):
        x1, y1, w, h = xy[0], xy[1], xy[2], xy[3]
        m, ea, n = fill[0], fill[1], fill[2]
        self.polygon(((x1, y1), (x1 + w, y1), (x1 + (w / 3) * 2, y1 + h),
                      (x1 + (w / 3), y1 + h), (x1, y1)), m)
        x_n, y_n = x1 + w // 2, y1 + h / 3 * 2
        self.polygon(((x_n, y_n),
                      (x_n + h // 8, y_n - (h // 4 * math.sqrt(3)) / 2),
                      (x_n - h // 8, y_n - (h // 4 * math.sqrt(3)) / 2)),
                      n)
        x_ts1, x_ts2, y_ts1_2 = x1 + w // 3, x1 + w // 3 * 2, y1 + h // 3
        self.ellipse((x_ts1 - h // 8, y_ts1_2 - h // 8, x_ts1 + h // 8,
                      y_ts1_2 + h // 8), ea)
        self.ellipse((x_ts2 - h // 8, y_ts1_2 - h // 8, x_ts2 + h // 8,
                      y_ts1_2 + h // 8), ea)
```

Напишите класс **Кукольник (Puppeteer)**, экземпляр **p** которого инициализируется с аргументами: *имя* (строка), *длина бороды*, *сколько раз чихает*, когда сильно растроган (целые числа). Класс реализует (инкапсулирует) функциональность:

- **str(p)** — возвращает строковое представление в виде **Puppeteer <имя>, <длина бороды>, <количество чиханий>**;
- **p.wrap_his_beard(circumference)** — обматывает бороду вокруг ствола дерева. Возвращает количество полных оборотов, вычисленное как *длина бороды // окружность ствола* (аргумент метода);
- **p.sneeze()** — чихает. Возвращает строку **Sneeze**, повторённую столько раз, сколько записано у него в соответствующем атрибуте;
- **p.frustrate(arg)** — расстраивается. Если аргумент метода кратен двум, то свойство *чихать* увеличивается на 1, если не кратен, то уменьшается на 1, если есть, что уменьшать;
- кукольников можно сравнивать. Сначала сравниваются по длине бороды, потом по чиханию, потом по длине имени и, наконец, имена по алфавиту.

Пример 1

Ввод	Вывод

`from solution import Puppeteer`
 	
`p = Puppeteer('Mangiafuoco', 10, 5)`
`print(p)`
`print(p.wrap_his_beard(3))`
`print(p.sneeze())`
`print(p.frustrate(5))`
`print(p)`
	`Puppeteer Mangiafuoco, 10, 5`
	`3`
	`SneezeSneezeSneezeSneezeSneez`
	`None`
	`Puppeteer Mangiafuoco, 10, 4`

Пример 2

Ввод	Вывод

`from solution import Puppeteer`
 	
`p1 = Puppeteer('Mangiafuoco', 10, 5)`
`p2 = Puppeteer('Carabas', 10, 5)`
`print(p1, p2, sep='\n')`
`print(p1 > p2)`
`print(p1.wrap_his_beard(2))`
`p2.frustrate(4)`
`print(p2 >= p1)`
`print(p2.sneeze())`
	`Puppeteer Mangiafuoco, 10, 5`
	`Puppeteer Carabas, 10, 5`
	`True`
	`5`
	`True`
	`SneezeSneezeSneezeSneezeSneezeSneeze`

```
class Puppeteer:
    def __init__(self, name, ln, count):
        self.name = name
        self.ln = ln
        self.count = count

    def __str__(self):
        return 'Puppeteer {}, {}, {}'.format(self.name, self.ln,
self.count)

    def wrap_his_beard(self, circumference):
        return self.ln // circumference

    def sneeze(self):
        return 'Sneeze' * self.count

    def frustrate(self, arg):
        if arg % 2 == 0:
            self.count += 1
        elif self.count > 0:
            self.count -= 1
```

```
    def __gt__(self, other):
        if self.ln > other.ln:
            return True
        elif self.ln == other.ln:
            if self.count > other.count:
                return True
            elif self.count == other.count:
                if len(self.name) > len(other.name):
                    return True
                elif len(self.name) == len(other.name):
                    if self.name > other.name:
                        return True
        return False

    def __eq__(self, other):
        return (self.ln == other.ln and self.count == other.count and
                self.name == other.name)

    def __lt__(self, other):
        if self.ln < other.ln:
            return True
        elif self.ln == other.ln:
            if self.count < other.count:
                return True
            elif self.count == other.count:
                if len(self.name) < len(other.name):
                    return True
                elif len(self.name) == len(other.name):
                    if self.name < other.name:
                        return True
        return False
```

```
    def __ge__(self, other):
        return self > other or self == other

    def __le__(self, other):
        return self < other or self == other
```