Вложенные циклы

На данном занятии мы рассмотрим вложенные циклы, позволяющие запустить цикл внутри циклического оператора. Приведем несколько примеров вложенности разных циклов, а также применение операторов break и continue со вложенными циклами.

Вложенные циклы. Принцип работы.

Часто бывают ситуации, когда один и тот же набор действий необходимо выполнить несколько раз для каждого повторяющегося действия. Например, мы уже несколько раз сталкивались с задачами, когда программа получает от пользователя данные до сигнала остановки, — для этого используется цикл. А теперь представьте, что после ввода данных или числа с ними надо сделать какие-либо действия, которые тоже требуют цикла (например, вычислить факториал), тогда нам нужен еще один цикл, внутри первого.

Вложенные циклы

Циклы называются вложенными (т. е. один цикл находится внутри другого), если внутри одного цикла во время каждой итерации необходимо выполнить другой цикл. Так для каждого витка внешнего цикла выполняются все витки внутреннего цикла. Основное требование для таких циклов: чтобы **все** действия вложенного цикла располагались внутри внешнего.

При использовании вложенных циклов стоит помнить, что изменения, внесенные внутренним циклом в какие-либо данные, могут повлиять на внешний.

Давайте рассмотрим следующую задачу: необходимо вывести в строку таблицу умножения для заданного числа. Задача решается так:

```
k = int(input())
for i in range(1, 10):
    print(i, '*', k, '=', k * i, sep='', end='\t')
```

А если нам нужно вывести таблицу умножения для всех чисел от 1 до k?

Очевидно, что в этом случае предыдущую программу нужно повторить k раз, где вместо k будут использоваться числа от 1 до k включительно.

Эту задачу можно записать двумя циклами, где для каждого значения внешнего цикла будут выполняться все значения внутреннего цикла.

Программа будет выглядеть так:

```
k = int(input())
for j in range(1, k + 1):
    for i in range(1, 10):
        print(i, '*', j, '=', j * i, sep='', end='\t')
    print()
```

Проанализируем работу данной программы. Выполнение программы начинается с внешнего цикла. Итератор j внешнего цикла for меняет свое значение от начального (1) до конечного (k). Обратите внимание: чтобы включить число k в рассматриваемый диапазон, в заголовке цикла указывается промежуток от 1 до k + 1. Затем циклически выполняется следующее:

1. Проверяется условие j < k + 1.
2. Если оно соблюдается, выполняется оператор в теле цикла, т. е. выполняется внутренний цикл.

Итератор i внутреннего цикла for будет изменять свои значения от начального (1) до конечного (10), не включая 10

Затем циклически выполняется следующее:

- Проверяется условие i < 10
- Если оно удовлетворяется, выполняется оператор в теле цикла, т. е. оператор print(i, '*', j, '=', j * i, sep='', end='\t'), выводящий на экран строку таблицы умножения в соответствии с текущими значениями переменных i и j

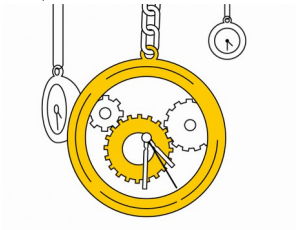
- Затем значение итератора i внутреннего цикла увеличивается на единицу, и оператор внутреннего цикла for проверяет условие i < 10. Если условие соблюдается, выполняется тело внутреннего цикла при неизменном значении итератора внешнего цикла до тех пор, пока выполняется условие i < 10
- Если условие i < 10 не удовлетворяется, т. е. как только i станет равен или больше 10, оператор тела цикла не выполняется, внутренний цикл завершается и управление в программе передается за пределы оператора for внутреннего цикла, т. е. выполняется перевод строки, вызванный использованием функции print() (строка 5), а затем возвращается к оператору for внешнего цикла

3. Значение итератора внешнего цикла j увеличивается на единицу, и проверяется условие j < k + 1. Если условие не соблюдается, т. е. как только j станет больше k, оператор тела цикла не выполняется, внешний цикл завершается и управление в программе передается за пределы оператора for внешнего цикла, т. е. в данном случае программа завершает работу.

Таким образом, на примере печати таблицы умножения показано, что при вложении циклов внутренний цикл выполняется полностью от начального до конечного значения параметра при неизменном значении параметра внешнего цикла. Затем значение параметра внешнего цикла изменяется на единицу, и опять от начала и до конца выполняется вложенный цикл. И так до тех пор, пока значение параметра внешнего цикла не станет больше конечного значения, определенного в операторе for внешнего цикла.

Графическое представление вложенных циклов

Работу циклов можно сравнить с вращением связанных шестеренок разного размера:



Внешний цикл — это как бы большая шестеренка, за один свой оборот (виток цикла) внешний цикл заставляет вращаться вложенный цикл (меньшую шестеренку) несколько раз.

Обратите внимание: такая иллюстрация точна в случае, если число повторов вложенного цикла не зависит от того, какой именно (1-й, n-й или иной) виток делает внешний цикл, а так бывает не всегда.

Оператор break и continue во вложенных циклах

Рассмотрим другую задачу: представьте, что необходимо распечатать все строки таблицы умножения для чисел от 1 до 10, кроме строки для числа k.

Тогда нам нужно будет пропустить выполнение внутреннего цикла, когда придет k-я строка.

Это можно сделать через оператор continue, который просто прервет выполнение данного витка цикла и перейдет к следующей итерации цикла:

```
k = int(input())
for j in range(1, 10):
    if j == k:
        continue
    for i in range(1, 10):
        print(i, '*', j, '=', j * i, sep='', end='\t')
    print()
```

Важно! Обратите внимание: если оператор break или continue расположен внутри вложенного цикла, он действует именно на вложенный цикл, а не на внешний. Нельзя выскочить из вложенного цикла сразу на самый верхний уровень.

А теперь попробуйте вывести всю таблицу умножения, кроме столбца k.

Вот еще одна программа, которая использует вложенные циклы и оператор break. Она учит пользователя вводить числа палиндромы — программа выполняется до тех пор, пока не будет введено число палиндром:

```
print('Тренажер по вводу палиндрома:')
while True:
    print('Введите число палиндром:')
    number = n = int(input())
    reverse = 0
    while n > 0:
        reverse = reverse * 10 + n % 10
        n //= 10
    if number == reverse:
        print('Вы ввели палиндром! Программа остановлена.')
        break
    print('Введенное число не палиндром, попробуйте еще раз.')
```

Пример 1. Таблица деления

Выведите таблицу деления заданных размеров.

Формат ввода

На первой строке вводится число колонок в таблице.

На второй строке вводится число строк в таблице.

Формат вывода

Выводится указанное число строк. В каждой строке выводятся разделённые символами пустого пространства частные: номер колонки, делённый на номер строки. Нумерация колонок и строк начинается с 1.

Пример

Ввод	Вывод
3	1.0 2.0 3.0
2	0.5 1.0 1.5

Примечания: При работе с вещественными числами может возникнуть проблема точности вычислений, хотя для этой задачи она и не столь важна.

Однако тем, кого заинтересовал этот вопрос, мы раскроем тайну: для данной задачи контроль точности вычислений установлен в размере 10−7.

```
x = int(input())
y = int(input())
for i in range(1, y + 1):
    for j in range(1, x + 1):
        print(j / i, sep='', end='\t')
    print()

```

Пример 2. Рисуем прямоугольник

Вася решил познать азы ASCII-живописи и, как все начинающие художники, начал с рисования простых фигур.

Помогите Васе написать программу построения прямоугольника n x m, состоящего из символов symb.

Фигура должна быть пустой, а не заполненной. То есть она должна состоять только из контура.

Формат ввода

Два числа, каждое в отдельной строке — высота и ширина прямоугольника. На третьей строке символ, используемый для рисования контуров.

Пример 1

Ввод	Вывод
3 3 &	&&& & & &&&

Пример 2

Ввод	Вывод
15 10 .	....

```python
x = int(input())
y = int(input())
z = input()
for i in range(1, x + 1):
    for j in range(1, y + 1):
        if i == x or i == 1 or j == 1 or j == y:
            print(z, sep=' ', end='')
        else:
            print(" ", end='')
    print()

```

Пример 3. Обратный отсчёт: серия пусков

Производим серию последовательных пусков космических аппаратов (в действительности так обычно не делают).

Для каждого космического аппарата нужно вести свой обратный отсчёт, причём отсчёт начинается сразу с 0 секунд для первого пуска и удлиняется на 1 секунду для каждого следующего пуска (допустим, чтобы запускаемые аппараты не мешали другу).

Формат ввода

Вводится одно натуральное число — количество запускаемых аппаратов.

Формат вывода

Для каждого из запускаемых аппаратов проводится обратный отсчёт, который начинается с 0 секунд для первого аппарата и длится на 1 секунду дольше для каждого следующего аппарата. В ходе отсчёта выводится фраза «Осталось секунд: 2» (подставить нужное количество секунд) для каждой секунды отсчёта. После этого выводится слово «Пуск» и номер аппарата (начиная с 1).

Пример

Ввод	Вывод
3	Осталось секунд: 0 Пуск 1 Осталось секунд: 1 Осталось секунд: 0 Пуск 2 Осталось секунд: 2 Осталось секунд: 1 Осталось секунд: 0 Пуск 3

```
x = int(input())
for i in range(x):
    z = i
    for j in range(i + 1):
        print('Осталось секунд:', z)
        z = z - 1
    print("Пуск", i + 1)
```

Пример 4. Логистический максимин

Ваша компания занимается грузоперевозками в Швейцарских Альпах. Вам нужно доставить груз из пункта A в пункт Z на большом грузовике. Из A в Z ведёт несколько дорог, каждая из которых проходит через несколько туннелей известной высоты. Выясните максимальную высоту, которую может иметь ваш грузовик.

Формат ввода

На первой строке вводится количество дорог. Затем для каждой дороги вводится (на отдельных строках) количество туннелей и высота каждого туннеля (точнее, максимально допустимая высота грузовика) в сантиметрах.

Формат вывода

Два целых числа: номер дороги (начиная нумерацию с единицы), по которой нужно проехать, чтобы высота грузовика была наибольшей, и сама эта высота.

Гарантируется, что ответ однозначный.

Пример

Ввод	Вывод
2 3 470 430 465 2 451 450	2 450

```
x = int(input())
min = 10000
max = 0
for i in range(1, x + 1):
    min = 10000
    y = int(input())
    for j in range(1, y + 1):
        z = int(input())
        if z < min:
            min = z
    if min >= max:
        max = min
        n = i
print(n, max)
```

Пример 5. Простые числа на миллион долларов

Точная оценка количества простых чисел, меньших данного числа, связана с вопросом об истинности математического утверждения, известного как гипотеза Римана. Поскольку эта гипотеза имеет большое значение ещё и для многих других математических вопросов, доказательство этой гипотезы является одной из семи (и одной из шести нерешённых на данный момент) «задач миллениума», за решение которых Математический институт Клэя предлагает приз в миллион долларов.

Напишите программу, которая выводит все простые числа, меньшие данного натурального числа.

Формат ввода

Одно натуральное число.

Формат вывода

Все простые числа, меньшие введённого числа.

Пример

Ввод	Вывод
20	2 3 5 7 11 13 17 19

```
x = int(input())
for i in range(2, x):
    k = 0
    for j in range(2, (x + 1) // 2):
        if i % j == 0 and i != j:
            k = k + 1
    if k == 0:
        print(i)
```

Пример 6. Начинающий фермер

Для восстановления сельского хозяйства государство выделяет целевые субсидии на покупку скота начинающим фермерам. Выделяемая сумма определяется для каждого региона отдельно, при этом устанавливается точное количество голов скота, которое надо приобрести. Цены следующие: бык – 20 тыс. рублей, корова – 10 тыс. рублей, а теленок – 5 тыс. рублей. Выделяемую сумму необходимо потратить полностью, иначе финансы сгорят.

Выведите все возможные варианты стада, которое может купить начинающий фермер на эту сумму. Обратите внимание, что для развития хозяйства необходимо, чтобы в стаде был как минимум один бык.

Гарантируется, что на выделенную сумму можно купить хотя бы один вариант стада, удовлетворяющий всем условиям.

Формат ввода

В первой строке, указывается размер выделяемой субсидии в тыс. рублей. На второй строке – количество голов скота, которое надо купить.

Формат вывода

Строки, описывающие состав стада – количество быков, коров и телят, которые могут быть куплены

Пример 1

Ввод	Вывод
560 100	1 9 90 2 6 92 3 3 94 4 0 96

Пример 2

Ввод	Вывод
520 100	1 1 98

```
x = int(input())
y = int(input())
for k1 in range(1, y + 1):
    for k2 in range(y + 1):
        for k3 in range(y + 1):
            if k1 * 20 + k2 * 10 + k3 * 5 == x and k1 + k2 + k3 == y:
                print(k1, k2, k3)
```

Задания на практику

1. Создайте программу которая использует вложенный цикл for для поиска простых чисел от 2 до 100.
2. Напишите программу для вывода узора по образцу, используя вложенный цикл.

Образец:

```
*
**
***
****
*****
****
***
**
*
```

3. Дано натуральное число n<=9. Выведите лесенку из n ступенек, i-я ступенька состоит из чисел от 1 до i без пробелов.

Входные данные

Вводится натуральное число.

Выходные данные

Выведите ответ на задачу.

Пример:

№	Входные данные	Выходные данные
1	3	1 12 123

Множества

На данном занятии мы обсудим множества Python. Этот тип данных аналогичен математическим множествам, он поддерживает быстрые операции проверки наличия элемента в множестве, добавления и удаления элементов, операции объединения, пересечения и вычитания множеств.

Объекты типа set

Мы написали уже много программ, работающих с данными, количество которых неизвестно на момент написания программы. Теперь было бы здорово уметь хранить в памяти неизвестное на момент написания программы количество данных. В этом нам помогут так называемые **коллекции** — специальные типы данных, которые умеют хранить несколько значений под одним именем. Первая из коллекций, с которой мы познакомимся, называется **множество**.

Множество

Множество — составной тип данных, представляющий собой несколько значений (элементов множества) под одним именем. Этот тип называется *set*, не создавайте, пожалуйста, переменные с таким именем! Чтобы задать множество, нужно в фигурных скобках перечислить его элементы.

Здесь создается множество из четырех элементов (названий млекопитающих), которое затем выводится на экран:

```
mammals = {'cat', 'dog', 'fox', 'elephant'}
print(mammals)
```

Введите этот код в Python и запустите программу несколько раз. Скорее всего, вы увидите разный порядок перечисления млекопитающих, так происходит потому, что элементы в множестве Python не упорядочены. Это позволяет быстро выполнять операции над множествами, о которых мы скоро поговорим чуть позже.

Создание множества

Для создания пустых множеств обязательно вызывать функцию *set*:

```
empty = set()
```

Обратите внимание: элементами множества могут быть строки или числа. Возникает вопрос: а может ли множество содержать и строки, и числа? Давайте попробуем:

```
mammals_and_numbers = {'cat', 5, 'dog', 3, 'fox', 12, 'elephant', 4}
print(mammals_and_numbers)
```

Как видим, множество может содержать и строки, и числа, а Python опять выводит элементы множества в случайном порядке. Заметьте, если поставить в программе оператор вывода множества на экран несколько раз, не изменяя само множество, порядок вывода элементов не изменится.

Может ли элемент входить в множество несколько раз? Это было бы странно, так как совершенно непонятно, как отличить один элемент от другого. Нет смысла хранить несколько одинаковых объектов, удобно иметь контейнер, сохраняющий только уникальный объекты. Поэтому множество содержит каждый элемент только один раз. Следующий фрагмент кода это демонстрирует:

```
birds = {'raven', 'sparrow', 'sparrow', 'dove', 'hawk', 'falcon'}
print(birds)
```

Важно! Итак, у множеств есть три ключевые особенности:

- Порядок элементов в множестве не определен
- Элементы множеств — строки и/или числа
- Множество не может содержать одинаковых элементов

Выполнение этих трех свойств позволяет организовать элементы множества в структуру со сложными взаимосвязями, благодаря которым можно быстро проверять наличие элементов в множестве, объединять множества и т. д. Но пока давайте обсудим ограничения.

Операции над множеством

Простейшая операция — **вычисление числа элементов** множества. Для этого служит функция *len*. Мы уже встречались с этой функцией раньше, когда определяли длину строки:

```
my_set = {'a', 'b', 'c'}
n = len(my_set)  # => 3
```

Далее можно **вывести элементы** множества с помощью функции *print*:

```
my_set = {'a', 'b', 'c'}
print(my_set) # => {'b', 'c', 'a'}
```

В вашем случае порядок может отличаться, так как правило упорядочивания элементов в множестве выбирается случайным образом при запуске интерпретатора Python.

Очень часто необходимо **обойти все элементы** множества в цикле. Для этого используется цикл *for* и оператор *in*, с помощью которых можно перебрать не только все элементы диапазона (как мы это делали раньше, используя *range*), но и элементы множества:

```
my_set = {'a', 'b', 'c'}
for elem in my_set:
    print(elem)
```

такой код выводит:

```
b
a
c
```

Однако, как и в прошлый раз, в вашем случае порядок может отличаться: заранее он неизвестен. Код для работы с множествами нужно писать таким образом, чтобы он правильно работал при любом порядке обхода. Для этого надо знать два правила:

- Если мы не изменяли множество, порядок обхода элементов тоже не изменится
- После изменения множества порядок элементов может измениться произвольным образом

Чтобы **проверить наличие элемента** в множестве, можно воспользоваться уже знакомым оператором *in*:

```
if elem in my_set:
    print('Элемент есть в множестве')
else:
    print('Элемента нет в множестве')
```

Выражение elem in my_set возвращает True, если элемент есть в множестве, и False, если его нет. Интересно, что эта операция для множеств в Python выполняется за время, не зависящее от мощности множества (количества его элементов).

Добавление элемента в множество делается при помощи *add*:

```
new_elem = 'e'
my_set.add(new_elem)
```

add — что-то вроде функции, «приклеенной» к конкретному множеству. Такие «приклеенные функции» называются **методами**.

Таким образом, если в коде присутствует имя множества, затем точка и еще одно название со скобками, второе название — имя метода. Если элемент, равный *new_elem*, уже существует в множестве, оно не изменится, поскольку не может содержать одинаковых элементов. Ошибки при этом не произойдет. Небольшой пример:

```
my_set = set()
my_set.add('a')
my_set.add('b')
my_set.add('a')
print(my_set)
```

Данный код три раза вызовет метод *add*, «приклеенный» к множеству *my_set*, а затем выведет либо {'a', 'b'}, либо {'b', 'a'}.

С **удалением элемента** сложнее. Для этого есть сразу три метода: *discard* (удалить заданный элемент, если он есть в множестве, и ничего не делать, если его нет), *remove* (удалить заданный элемент, если он есть, и породить ошибку KeyError, если нет) и *pop*. Метод pop удаляет некоторый элемент из множества и возвращает его как результат. Порядок удаления при этом неизвестен.

```
my_set = {'a', 'b', 'c'}

my_set.discard('a')        # Удалён
my_set.discard('hello')    # Не удалён, ошибки нет
my_set.remove('b')         # Удалён
print(my_set)              # В множестве остался один элемент 'c'
my_set.remove('world')     # Не удалён, ошибка KeyError
```

На первый взгляд, странно, что есть метод *remove*, который увеличивает количество падений вашей программы. Однако если вы на 100 % уверены, что элемент должен быть в множестве, то лучше получить ошибку во время отладки и исправить ее, чем тратить время на поиски при неправильной работе программы.

Метод *pop* удаляет из множества случайный элемент и возвращает его значение:

```
my_set = {'a', 'b', 'c'}
print('до удаления:', my_set)
elem = my_set.pop()
print('удалённый элемент:', elem)
print('после удаления:', my_set)
```

Результат работы случаен, например, такой код может вывести следующее:

до удаления: {'b', 'a', 'c'}
удалённый элемент: b
после удаления: {'a', 'c'}

Если попытаться применить *pop* к пустому множеству, произойдет ошибка KeyError.

Очистить множество от всех элементов можно методом *clear*:

```
my_set.clear()
```

Операции над двумя множествами

Есть четыре операции, которые из двух множеств делают новое множество: объединение, пересечение, разность и симметричная разность.

Объединение двух множеств включает в себя все элементы, которые есть хотя бы в одном из них. Для этой операции существует метод *union*:

```
union = my_set1.union(my_set2)
```

Или можно использовать оператор /:

```
union = my_set1 | my_set2
```

Пересечение двух множеств включает в себя все элементы, которые есть в обоих множествах:

```
intersection = my_set1.intersection(my_set2)
```

Или аналог:

```
intersection = my_set1 & my_set2
```

Разность двух множеств включает в себя все элементы, которые есть в первом множестве, но которых нет во втором:

```
diff = my_set1.difference(my_set2)
```

Или аналог:

```
diff = my_set1 - my_set2
```

Симметричная разность двух множеств включает в себя все элементы, которые есть только в одном из этих множеств:

```
symm_diff = my_set1.symmetric_difference(my_set2)
```

Или аналогичный вариант:

```
symm_diff = my_set1 ^ my_set2
```

Люди часто путают обозначения | и &, поэтому рекомендуется вместо них писать s1.union(s2) и s1.intersection(s2). Операции − и ^ перепутать сложнее, их можно записывать прямо так.

```
s1 = {'a', 'b', 'c'}
s2 = {'a', 'c', 'd'}
union = s1.union(s2)              # {'a', 'b', 'c', 'd'}
intersection = s1.intersection(s2)  # {'a', 'c'}
diff = s1 - s2                  # {'b'}
symm_diff = s1 ^ s2               # {'b', 'd'}
```

Сравнение множеств

Все операторы сравнения множеств, а именно: ==, <, >, <=, >=, возвращают True, если сравнение истинно, и False — в противном случае.

Равенство и неравенство множеств

Множества считаются равными, если они содержат одинаковые наборы элементов. Равенство множеств, как в случае с числами и строками, обозначается оператором ==.

Неравенство множеств обозначается оператором *!=*. Он работает противоположно оператору ==.

```
if set1 == set2:
    print('Множества равны')
else:
    print('Множества не равны')
```

Обратите внимание на то, что у двух равных множеств могут быть разные порядки обхода, например, из-за того, что элементы в каждое из них добавлялись в разном порядке.

Теперь перейдем к операторам <=, >=. Они означают «является подмножеством» и «является надмножеством».

Подмножество и надмножество

Подмножество — некоторая выборка элементов множества, которая может быть как меньше множества, так и совпадать с ним, на что указывают символы «<» и «=» в операторе <=. Наоборот, надмножество включает все элементы некоторого множества и, возможно, какие-то еще.

```
s1 = {'a', 'b', 'c'}
print(s1 <= s1)  # True

s2 = {'a', 'b'}
print(s2 <= s1)  # True
s3 = {'a'}
print(s3 <= s1)  # True
s4 = {'a', 'z'}
print(s4 <= s1)  # False
```

Операция s1 < s2 означает «*s1* является подмножеством *s2*, но целиком не совпадает с ним». Операция s1 > s2 означает «*s1* является надмножеством *s2*, но целиком не совпадает с ним».

Операции с множествами

С множествами в питоне можно выполнять обычные для математики операции над множествами.

A | B A.union(B)	Возвращает множество, являющееся объединением множеств A и B.
A |= B A.update(B)	Добавляет в множество A все элементы из множества B.
A & B A.intersection(B)	Возвращает множество, являющееся пересечением множеств A и B.
A &= B A.intersection_update(B)	Оставляет в множестве A только те элементы, которые есть в множестве B.
A - B A.difference(B)	Возвращает разность множеств A и B (элементы, входящие в A, но не входящие в B).
A -= B A.difference_update(B)	Удаляет из множества A все элементы, входящие в B.
A ^ B A.symmetric_difference(B)	Возвращает симметрическую разность множеств A и B (элементы, входящие в A или в B, но не в оба из них одновременно).
A ^= B A.symmetric_difference_update(B)	Записывает в A симметрическую разность множеств A и B.
A <= B A.issubset(B)	Возвращает true, если A является подмножеством B.
A >= B A.issuperset(B)	Возвращает true, если B является подмножеством A.
A < B	Эквивалентно A <= B and A != B
A > B	Эквивалентно A >= B and A != B

Пример 1 Иностранные языки

Каждый ученик в классе имеет возможность изучать в группах английский, немецкий или французский языки, а также любую их комбинацию: английский и немецкий, немецкий и французский, английский и французский, английский, немецкий и французский.

У классного руководителя были списки учеников, изучающих каждый из языков. Для проведения классного мероприятия он напечатал списки, разрезал их пофамильно и разложил полученные листочки по кучкам: каждая кучка – это отдельный язык. Вдруг из окна подул ветер и перемешал все листочки. Помогите классному руководителю выяснить, сколько учеников в классе изучают ровно два языка.

Формат ввода

В первых трех строках указывается количество учеников, изучающих английский, немецкий и французский языки (M, N и K). Затем идут M+N+K строк с фамилиями учеников, расположенные в произвольном порядке. Это означает, что перемешались не только сами фамилии, но и группы по изучению языков. Гарантируется, что среди учеников нет однофамильцев.

Формат вывода

Количество учеников, которые изучают ровно два языка. Если таких не окажется, в строке вывода нужно написать NO.

Пример 1

Ввод	Вывод
2 2 2 Иванов Петров Сидоров Иванов Петров Иванов	1

Пример 2

Ввод	Вывод
2 2 2 Зайцев Петров Петров Зайцев Иванов Иванов	3

```
x = int(input())
y = int(input())
z = int(input())
m1 = set()
m2 = set()
k = 0
for i in range(x + y + z):
    f = input()
    if f in m2:
        k = k + 1
    elif f in m1:
        m2.add(f)
    else:
        m1.add(f)
y = x + y + z - 2 * k - len(m1)
if y > 0:
    print(y)
else:
    print('NO')
```

Пример 2. Книги на лето

Алексей получил в конце учебного года список литературы на лето. Теперь ему надо выяснить, какие книги из этого списка у него есть, а каких нет. К счастью, у Алексея на компьютере есть текстовый документ, в котором записаны все книги из его домашней библиотеки в случайном порядке. Определите, какие книги из списка на лето есть у Алексея, а каких нет.

Формат ввода

В первой строке записано число M — число книг в домашней библиотеке. Во второй строке записано число N - число книг в списке на лето. В домашней библиотеке и списке книг есть хотя бы по одной книге ($M \geq 1$ и $N \geq 1$). Далее идут M строчек с названиями книг из домашней библиотеки и N строчек названий из списка на лето. Гарантируется, что все слова в названиях книг разделены одним пробелом, а после последнего слова сразу идёт перевод строки (т. е. нет «невидимых» пробелов).

Формат вывода

Выходные данные: N строчек, в каждой из которых написано слово YES, если книга найдена в библиотеке, и NO, если нет.

Пример 1

Ввод	Вывод
4 2 Хоббит Алиса в стране чудес Том Сойер Остров сокровищ Том Сойер Властелин Колец	YES NO

Пример 2

Ввод	Вывод
4 4 Хоббит Алиса в стране чудес Том Сойер Остров сокровищ Буратино Хоббит Остров сокровищ Война и мир	NO YES YES NO

```
x = int(input())
y = int(input())
m1 = set()
m2 = set()
k = 0
for i in range(x):
    f = input()
    m1.add(f)
for i in range(y):
    f = input()
    m2.add(f)
    if len(m1 & m2) == 1:
        print("YES")
    else:
        print("NO")
    m2 = set()
```

Пример 3. Новые блюда

Главный повар летнего лагеря хочет приготовить в последний день смены блюда, которые ни разу не готовил в течение смены. В его распоряжении есть список блюд, которые можно приготовить в столовой и списки блюд, которые были приготовлены в каждый из дней смены.

Формат ввода

Число блюд (M), которые может приготовить столовая. M строк с названиями блюд. Далее число дней N, для которых есть списки блюд. Далее N блоков строчек для списков блюд на каждый из дней. В первой строчке блока записано число блюд в данный день, затем перечисляются эти блюда.

Формат вывода

Список блюд, которые ещё ни разу не готовили. Каждое блюдо выводится на отдельной строчке. Порядок вывода — произвольный.

Пример

Ввод	Вывод
5 Овсянка Рис Суп МаннаяКаша Рыба 2 3 Рис Суп Рыба 2 Рис Рыба	МаннаяКаша Овсянка

```
x = int(input())
m1 = set()
m2 = set()
for i in range(x):
    f = input()
    m1.add(f)
y = int(input())
for i in range(y):
    k = int(input())
    for j in range(k):
        f = input()
        m2.add(f)
b = m1 - m2
for i in range(len(b)):
    c = b.pop()
    print(c)
```

Пример 4. Города

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Аня и Наташа играют в города. Они очень любят эту игру, знают много городов и к концу игры забывают, какие уже называли. На вас возложена почётная задача вести запись игры и напоминать девочкам, если какой-то город уже был назван.

Формат ввода

В первой строке записано число названных городов N. Затем идут N строк с названиями городов и ещё одна строка с новым только что названым городом.

Формат вывода

Слово OK, если такого города ещё не было названо, и TRY ANOTHER, если город уже был назван.

Пример 1

Ввод	Вывод
3 Москва Нью-Йорк Лондон Париж	OK

Пример 2

Ввод	Вывод
8 Лондон Париж Москва Вашингтон Берлин Вена Мадрид Рим Хельсинки	OK

Пример 3

Ввод	Вывод
8 Лондон Париж Москва Вашингтон Берлин Вена Мадрид Рим Мадрид	TRY ANOTHER

```
x = int(input())
m1 = set()
for i in range(x):
    f = input()
    m1.add(f)
y = input()
if y in m1:
    print("TRY ANOTHER")
else:
    print("OK")
```

Пример 5. Однофамильцы

Начальник кадровой службы хочет узнать, сколько мужчин-однофамильцев работает в организации. У него есть список фамилий, и на основании этого списка нужно вычислить количество фамилий, которые совпадают с другими.

Формат ввода

В первой строке указывается количество мужчин - сотрудников организации (N). Затем идут N строк с фамилиями этих сотрудников в произвольном порядке.

Формат вывода

Количество однофамильцев в организации.

Пример 1.

Ввод	Вывод
6 Иванов Петров Сидоров Петров Иванов Петров	5

Пример 2

Ввод	Вывод
3 Иванов Петров Сидоров	0

```
x = int(input())
m1 = set()
m2 = set()
k = 0
s = ""
for i in range(x):
    f = input()
    s = s + f
    m1.add(f)
for i in m1:
    if s.count(i) > 1:
        k = k + s.count(i)
print(k)

```

Пример 6. Иностранные Языки

Каждый ученик в классе изучает либо английский, либо немецкий, либо оба этих языка. У классного руководителя были списки учеников, изучающих каждый из языков. Для проведения классного мероприятия он напечатал списки и разрезал их пофамильно и разложил полученные листочки две отдельные кучки. Вдруг из окна подул ветер и перемешал листочки в обеих кучах. Помогите классному руководителю выяснить, сколько учеников в классе изучают только один язык.

Формат ввода

В первых двух строках указывается количество учеников, изучающих английский и немецкий языки (M и N). Затем идут M+N строк с фамилиями учеников в произвольном порядке. Гарантируется, что среди учеников нет однофамильцев.

Формат вывода

Количество учеников, которые изучают только один язык. Если таких не окажется, в строке вывода нужно написать NO.

Пример 1

Ввод	Вывод
3 2 Иванов Петров Васечкин Иванов Михайлов	3

Пример 2

Ввод	Вывод
3 3 Иванов Петров Васечкин Иванов Петров Васечкин	NO

```
x = int(input())
y = int(input())
m1 = set()
m2 = set()
k = 0
for i in range(x + y):
    f = input()
    if f in m1:
        m1.remove(f)
        m2.add(f)
    else:
        m1.add(f)
if len(m1) <= 0:
    print('NO')
else:
    print(len(m1))
```

Задания на лабораторную работу

Задание 1.

Напишите программу, которая считает знаки пунктуации в символьной строке. К знакам пунктуации относятся символы из набора ".,;:!?".

Входные данные

Программа получает на вход символьную строку.

Выходные данные

Программа должна вывести общее количество знаков пунктуации во входной строке.

Примеры

входные данные

Hi, guys!

выходные данные

2

Задание 2.

Напишите программу, которая находит все различные цифры в символьной строке.

Входные данные

На вход программе подаётся символьная строка.

Выходные данные

Программа должна вывести в одной строке все различные цифры, которые встречаются в исходной строке, в порядке возрастания. Если в строке нет цифр, нужно вывести слово 'NO'.

Примеры

входные данные

ab1n32kz2

выходные данные

123

входные данные

asdasd

выходные данные

NO

Задание 3.

Напишите программу, которая выводит все цифры, встречающиеся в символьной строке больше одного раза.

Входные данные

Входная строка может содержать содержит цифры, пробелы и латинские буквы.

Выходные данные

Программа должна вывести в одну строчку в порядке возрастания все цифры, встречающиеся во входной строке больше одного раза. Если таких цифр нет, нужно вывести слово 'NO'.

Примеры

входные данные

asd12gh23

выходные данные

2

входные данные

t1y2u3i4o5

выходные данные

NO

Задание 4.

Напишите программу, которая определяет правильность записи целого числа в восьмеричной системе счисления.

Входные данные

На вход программы поступает символьная строка.

Выходные данные

Программа должна вывести ответ 'YES', если строка представляет собой правильную запись целого числа в восьмеричной системе счисления, и 'NO', если запись ошибочна.

Задание 5.

Напишите программу, которая удаляет из строки все повторяющиеся символы.

Входные данные

На вход программы подаётся строка, содержащая символы таблицы ASCII.

Выходные данные

Программа должна вывести исходную строку, из которой удалены все повторяющиеся символы.

Примеры

входные данные

abc13a1b2z3c

выходные данные

abc132z

входные данные

QWasd123

выходные данные

QWasd123