

Однострочники Python для ускорения написания кода TIPS & TRICKS

Python часто выбирают из-за его простоты и читабельности. Но знаете ли вы, что код на Python можно существенно сокращать, не жертвуя функциональностью? Однострочники могут сэкономить вам много времени, сократить расходование памяти и произвести впечатление на ваших друзей.

Предупреждение от автора. Пожалуйста, не используйте однострочники на собеседованиях и в продакшен-коде. Эта статья носит скорее развлекательный характер: мы чисто из интереса посмотрим, как можно заменять блоки кода одной строкой. Но и польза от подобных знаний, безусловно, тоже есть.

Что такое однострочники?

Однострочник – это сжатый блок кода, вмещенный в одну строку. По-английски – one-liner. По сути это лаконичные, полезные программы, занимающие всего строку кода.

Зачем нужны однострочники?

Если вы еще не фанат однострочников, вероятно, вам интересно, зачем вообще они нужны, в чем их смысл. Вот несколько аргументов в пользу их изучения и применения:

Научившись писать однострочники, вы попутно куда лучше разберетесь в основах языка Python.

Однострочники позволяют писать код быстрее, а это может пригодиться на соревнованиях по программированию.

Вы научитесь писать код более «питонично». Люди, перешедшие на Python с других языков программирования, часто пишут код, не используя нативные функции этого языка. Скажем, не пользуются представлениями списков, множественным присваиванием, срезами и т. п. вещами.

Ловко применив однострочник, вы сможете произвести впечатление на друзей и коллег.

Но с применением однострочников связаны и определенные сложности.

Подумайте о программировании как о шахматах. Вы знаете основы (что такое переменные, циклы, условия, структуры данных, классы).

Однострочники можно сравнить с изучением мастерских ходов и созданием собственных стратегий.

[python_ad_block]

Поначалу вам может быть трудно, но как только вы поднатореете в написании однострочников, вы сможете достигать своих целей быстрее.

Примеры однострочников на Python

1. If-else

До:

```
if 3 < 2:
```

```
var=21
else:
    var=42
После:

var = 21 if 3<2 else 42
2. Elif
```

До:

```
>>> x = 42
>>> if x > 42:
>>>     print("no")
>>> elif x == 42:
>>>     print("yes")
>>> else:
>>>     print("maybe")
yes
```

После:

```
>>> print("no") if x > 42 else print("yes") if x == 42 else
print("maybe")
yes
3. If без else
```

До:

```
condition = True
if condition:
    print('hi')
```

После:

```
if condition: print('hello')
print('hello') if condition else None
4. Функция
```

До:

```
def f(x):
    return "hello " + x
```

После:

```
f = lambda x: "hello " + x
f = exec("def f(x):\n    return 'hello ' + x")
5. Цикл (list comprehension)
```

До:

```
squares = []
for i in range(10):
```

```
squares.append(i**2)
```

После:

```
squares=[i**2 for i in range(10)]
```

6. Цикл с условием `if`

До:

```
squares = []  
for i in range(10):  
    if i%2==0:  
        squares.append(i**2)
```

После:

```
squares = [i**2 for i in range(10) if i%2==0]
```

7. Цикл с `if else`

До:

```
squares = []  
for i in range(10):  
    if i%2==0:  
        squares.append(i**2)  
    else:  
        squares.append(False)
```

После:

```
squares = [i**2 if i%2==0 else False for i in range(10)]
```

8. Цикл `while` с `if else`

До:

```
c=0  
while c < 10:  
    if c!=5:  
        print(c)  
    else:  
        print("FIVE")  
    c+=1
```

После:

```
while c < 10: c+=1; print(c) if c!=5 else print("FIVE")
```

9. Меняем местами переменные

До:

```
>>> def swap(x,y):  
    x = x ^ y  
    y = x ^ y  
    x = x ^ y
```

```
        return x, y
>>> swap(10,20)
(20,10)
```

После:

```
>>> x, y = 10, 20
>>> x, y = y, x
(20, 10)
```

10. Множественное присваивание

До:

```
a="ONE"
b=2
c=3.001
```

После:

```
a, b, c = "One", 2, 3.001
```

11. Запись строки в файл

До:

```
text = "Helllloooooo"
fileName = "hello.txt"
f=open(fileName, "a")
f.write(text)
f.close()
```

После:

```
text = "Helllloooooo"
fileName = "hello.txt"
print(text, file=open(fileName, 'a'))
```

12. Быстрая сортировка

До:

Source - <https://stackabuse.com/quicksort-in-python/>

```
def partition(array, start, end):
    pivot = array[start]
    low = start + 1
    high = end
    while True:
        while low <= high and array[high] >= pivot:
            high = high - 1
        while low <= high and array[low] <= pivot:
            low = low + 1
        if low <= high:
            array[low], array[high] = array[high], array[low]
        else:
            break
```

```

    array[start], array[high] = array[high], array[start]
    return high
def quick_sort(array, start, end):
    if start >= end:
        return
    p = partition(array, start, end)
    quick_sort(array, start, p-1)
    quick_sort(array, p+1, end)
array = [29,99,27,41,66,28,44,78,87,19,31,76,58,88,83,97,12,21,44]
quick_sort(array, 0, len(array) - 1)
print(array)

```

После:

```

array = [29,99,27,41,66,28,44,78,87,19,31,76,58,88,83,97,12,21,44]
q = lambda l: q([x for x in l[1:] if x <= l[0]]) + [l[0]] + q([x for x
in l if x > l[0]]) if l else []
print(q(array))

```

13. Последовательность Фибоначчи

До:

```

def fib(x):
    if x <= 2:
        return 1
    return fib(x - 1) + fib(x - 2)

```

После:

```

fib=lambda x: x if x<=1 else fib(x-1) + fib(x-2)

```

14. HTTP-сервер

До:

```

import http.server
import socketserver
PORT = 8000
Handler = http.server.SimpleHTTPRequestHandler
with socketserver.TCPServer("", PORT), Handler) as httpd:
    print("serving at port", PORT)
    httpd.serve_forever()

```

После:

```

python -m http.server 8000

```

15. Вложенные циклы for

До:

```

iter1 = [1, 2, 3, 4]
iter2 = ['a', 'b', 'c']
for x in iter1:
    for y in iter2:

```

```
print(x, y)
```

После:

```
[print(x, y) for x in iter1 for y in iter2]
```

16. Вывод без перехода на новую строку

До:

```
for i in range(1,5):  
    print(i, end=" ")
```

После:

```
print(*range(1,5))
```

17. Класс

До:

```
class School():  
    fun = {}
```

После:

```
School = type('School', (object,), {'fun':{}})
```

18. Оператор walrus:= (Python 3.8)

До:

```
command = input("> ")  
while command != "quit":  
    print("You entered:", command)
```

После:

```
while (command := input("> ")) != "quit": print("You entered:",  
command)
```

От редакции Pythonist. Если вас заинтересовала тема однострочников, можем порекомендовать книгу «Python One-Liners» Кристиана Майера. Ее краткий обзор вы найдете в статье «Самые новые книги по Python для начинающих питонистов».

На easy На вход программе подаются два целых числа a и b

b. Напишите программу, которая выводит:

сумму чисел a и b ; разность чисел a и b ; произведение чисел a и b ; частное чисел a и b ; целую часть от деления числа a на b ; остаток от деления числа a на b ; корень квадратный из суммы их 10-х степеней: 10^{a+b}

- 10^{a+b}

. Формат входных данных На вход программе подаются два целых числа a и b ($b \neq 0$) $b(b \neq 0)$, каждое на отдельной строке.

```
# put your python code here
```

```
a = int(input())
b = int(input())
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b)
print(a % b)
print((a**10+b**10)**0.5)
```

Индекс массы тела

Напишите программу для вычисления и оценки индекса массы тела (ИМТ) человека. ИМТ показывает весит человек больше или меньше нормы для своего роста. ИМТ человека рассчитывают по формуле:

```
ИМТ
=
масса
(
кг
)
рост
(
м
)
х
рост
(
м
)
ИМТ=
рост(м)×рост(м)
масса (кг)

,
```

где масса измеряется в килограммах, а рост – в метрах.

Масса человека считается оптимальной, если его ИМТ находится между

18.5

и

25

25. Если ИМТ меньше

18.5

18.5, то считается, что человек весит ниже нормы. Если значение ИМТ больше

25

25, то считается, что человек весит больше нормы.

Программа должна вывести "Оптимальная масса", если ИМТ находится между 18.5 и 25 (включительно). "Недостаточная масса", если ИМТ меньше 18.5 и "Избыточная масса", если значение ИМТ больше 25.

```
ves = float(input())
rost = float(input())
imt = ves/rost**2
if imt >= 18.5 and imt <= 25:
    print("Оптимальная масса")
elif imt < 18.5:
    print("Недостаточная масса")
elif imt > 25:
    print("Избыточная масса")
```

Стоимость строки. Дана строка текста. Напишите программу для подсчета стоимости строки, исходя из того, что один любой символ (в том числе пробел) стоит 60 копеек. Ответ дайте в рублях и копейках в соответствии с примерами.

Формат входных данных. На вход программе подается строка текста.

Формат выходных данных. Программа должна вывести стоимость строки.

```
s=0
for i in input():
    s=s+1*0.60
print("%.f"%(s//1), "р. ", "%.f"%((s % 1)*100), "коп.")
```

Количество слов. Дана строка, состоящая из слов, разделенных пробелами. Напишите программу, которая подсчитывает количество слов в этой строке.

```
s=0
txt = input()
for i in range(len(txt)):
    if i < len(txt) and txt[i] == " " and txt[i+1] != " ":
        s=s+1
    elif i == len(txt) and s[i]==" ":
        s=s+1
print(s+1)
```

Зодиак Китайский гороскоп назначает животным годы в 12-летнем цикле. Один 12-летний цикл показан в таблице ниже. Таким образом, 2012 год будет очередным годом дракона.

Год Животное 2000 2000 Дракон 2001 2001 Змея 2002 2002 Лошадь 2003 2003 Овца 2004 2004 Обезьяна 2005 2005 Петух 2006 2006 Собака 2007 2007 Свинья 2008 2008 Крыса 2009 2009 Бык 2010 2010 Тигр 2011 2011 Заяц

Напишите программу, которая считывает год и отображает название связанного с ним животного. Ваша программа должна корректно работать с любым годом, не только теми, что перечислены в таблице.

```
y = int(input())
for j in range(12):

r=["Обезьяна", "Петух", "Собака", "Свинья", "Крыса", "Бык", "Тигр", "Заяц", "Д
ракон"
    , "Змея", "Лошадь", "Овца"]
    for i in range(j,100000,12):
        if y == i:
            print(r[j])
```

Переворот числа Дано пятизначное или шестизначное натуральное число. Напишите программу, которая изменит порядок его последних пяти цифр на обратный.

Координатные четверти Дан набор точек на координатной плоскости. Необходимо подсчитать и вывести количество точек, лежащих в каждой координатной четверти.

```
# put your python code here
x=[]
y=[]
a,b,c,z=0,0,0,0
for i in range(int(input())):
    f = input().split()
    x.append(int(f[0]))
    y.append(int(f[1]))
    f=''
for i in range(len(x)):
    if x[i] != 0 and y[i] != 0:
        if x[i] > 0 and y[i] > 0:
            a=a+1 #pervaya
        elif x[i] < 0 and y[i] > 0:
            b=b+1 # vtaraya
        elif x[i] < 0 and y[i] < 0:
            c=c+1 #tretaya
        elif x[i] > 0 and y[i] < 0 :
            z=z+1 #tshetvertaya
print("Первая четверть:",a)
print("Вторая четверть:",b)
print("Третья четверть:",c)
print("Четвертая четверть:",z)
```

Больше предыдущего На вход программе подается строка текста из натуральных чисел. Из неё формируется список чисел. Напишите программу подсчета количества чисел, которые больше предшествующего им в этом списке числа.

Формат входных данных

```
s=input().split()
j=[]
d=0
for i in range(len(s)):
    if i >= 1:
        if int(s[i]) > int(j[i-1]):
            #print(s[i],j[i-1])
            d=d+1
    j.append(s[i])
print(d)
```

Назад, вперёд и наоборот На вход программе подается строка текста из натуральных чисел. Из элементов строки формируется список чисел. Напишите программу, которая меняет местами соседние элементы списка (a[0] с a[1], a[2] с a[3] и т.д.). Если в списке нечетное количество элементов, то последний остается на своем месте.

```
# put your python code here
s=input().split()
j=[]
for i in range(len(s)):
    if i == 0 or i%2 ==0:
        j.append(int(s[i]))
    if i >= 1 and i%2 != 0:
        r=int(j[i-1])
        j[i-1] = s[i]
        j.append(int(r))
print(*j)
```

Сдвиг в развитии На вход программе подается строка текста из натуральных чисел. Из элементов строки формируется список чисел. Напишите программу циклического сдвига элементов списка направо, когда последний элемент становится первым, а остальные сдвигаются на одну позицию вперед, в сторону увеличения индексов.

Формат входных данных На вход программе подается строка текста из разделенных пробелами натуральных чисел.

Формат выходных данных Программа должна вывести элементы измененного списка с циклическим сдвигом, разделяя его элементы одним пробелом.

```
# put your python code here
s =input().split()
d=s[0]
s[0] = s[(len(s)-1)]
```

```
s.insert(1,d)
f=len(s)
s=s[-f:-1]
print(*s)
```

Различные элементы На вход программе подается строка текста, содержащая натуральные числа, расположенные по неубыванию. Из строки формируется список чисел. Напишите программу для подсчета количества разных элементов в списке.

```
s=input().split()
d=[]
for i in s:
    if i not in d:
        d.append(i)
print(len(d))
```

Произведение чисел

Напишите программу для определения, является ли число произведением двух чисел из данного набора. Программа должна выводить результат в виде ответа «ДА» или «НЕТ».

Формат входных данных

В первой строке подаётся число

0

(

0

<

0

<

1000

)

$n(0 < n < 1000)$ – количество чисел в наборе. В последующих

0

n строках вводятся целые числа, составляющие набор (могут повторяться). Затем следует целое число, которое является или не является произведением двух каких-то чисел из набора.

Формат выходных данных

Программа должна вывести «ДА» или «НЕТ» в соответствии с условием задачи

```
n = int(input())
d=[]
q=0
flag=False
for i in range(n):
    num_=int(input())
    d.append(num_)

com_n=int(input())
```

```

for i in range(len(d)):
    if d[i] == 0 and com_n == 0:
        flag = True

    if d[i] != 0:
        if com_n/d[i] in d:
            q=q+1
            if q >1:
                flag=True
if flag == True:
    print("ДА")
else:
    print("НЕТ")

```

Камень, ножницы, бумага Тимур и Руслан пытаются разделить фронт работы по курсу "Python для профессионалов". Для этого они решили сыграть в камень, ножницы и бумагу. Помогите ребятам бросить честный жребий и определить, кто будет делать очередной модуль нового курса.

```

a=input()
b=input()
timur=a
ruslan=b
flag = False
if timur == "ножницы" and ruslan== "бумага":
    flag = True
    print("Тимур")
if timur == "бумага" and ruslan == "камень":
    print("Тимур")
    flag = True
if timur == "камень" and ruslan == "ножницы":
    print("Тимур")
    flag = True
if a != b and flag == False:
    print("Руслан")
if a==b:
    print("ничья")

```

Камень, ножницы, бумага, ящерица, Спок Проиграв 10 10 раз Тимуру, Руслан понял, что так дело дальше не пойдет, и решил усложнить игру. Теперь Тимур и Руслан играют в игру Камень, ножницы, бумага, ящерица, Спок. Помогите ребятам вновь бросить честный жребий и определить, кто будет делать следующий модуль в новом курсе.

```

a=input()
b=input()
timur=a
ruslan=b
flag = False
if timur == "ножницы" and (ruslan== "бумага" or ruslan== "ящерица") :

```

```

    flag = True
    print("Тимур")
if timur == "бумага" and (ruslan== "Спок" or ruslan== "камень"):
    print("Тимур")
    flag = True
if timur == "камень" and (ruslan== "ножницы" or ruslan== "ящерица"):
    print("Тимур")
    flag = True
if timur == "ящерица" and (ruslan== "Спок" or ruslan== "бумага"):
    print("Тимур")
    flag = True
if timur == "Спок" and (ruslan== "ножницы" or ruslan== "камень"):
    print("Тимур")
    flag = True

if a != b and flag == False:
    print("Руслан")
if a==b:
    print("ничья")

```

Орел и решка Дана строка текста, состоящая из букв русского алфавита "О" и "Р". Буква "О" соответствует выпадению Орла, а буква "Р" - выпадению Решки. Напишите программу, которая подсчитывает наибольшее количество подряд выпавших Решек.

Формат входных данных На вход программе подается строка текста, состоящая из букв русского алфавита "О" и "Р".

Формат выходных данных Программа должна вывести наибольшее количество подряд выпавших Решек.

Примечание. Если выпавших Решек нет, то необходимо вывести число 0

```

s="000000PPRORORPPPPPP"
r=[]
d=0
t=0
for i in s:
    t=t+1
    if i != "P":
        if d>1:
            r.append(d)
            d=0
    if i=="P" :
        d=d+1
    if t == len(s):
        r.append(d)
print(max(r))

[3, 7]

```

```

s="000000PPP0P0PPPPPP"
d=s.split("0")

['', '', '', '', '', '', 'PPP', 'P', 'PPPPPP']

s=input()
d=s.split("0")
t,r=[],0
for i in d:
    r=0
    for j in range(len(i)):
        if i[j] == 'P':
            r=r+1
    t.append(r)
print(r)

OPP0P0P00PPP0
0

# put your python code here
s=input()
r=[]
d=0
t=0
for i in s:
    t=t+1
    if i != "P":
        if d>1:
            r.append(d)
            d=0
    if i=="P" :
        d=d+1
    if t == len(s):
        r.append(d)
print(max(r))

```

Тема урока: тип данных bool Логический тип данных Логические операторы Булевы значения как числа Функции bool(), type(), isinstance() Аннотация. Урок посвящен логическому типу данных bool.

Типы данных в Python В предыдущем курсе мы изучали примитивные типы данных, такие как int, float, str, list и т.д. В их числе мы упоминали и о логическом типе данных, однако вскользь. Давайте закроем пробелы текущим уроком, который посвящен целиком и полностью логическому типу данных, который в Python представлен типом bool 🤖.

Встроенные типы данных Логический тип данных Логический тип данных (булев тип, Boolean) — примитивный тип данных в информатике, принимающий два возможных значения, иногда называемых истиной (True) и ложью (False). Присутствует в подавляющем большинстве языков программирования как самостоятельная сущность или реализуется через численный тип данных. В некоторых языках программирования за значение "истина" принимается 1, за значение "ложь" — 0

Название типа Boolean получило в честь английского математика и логика Джорджа Буля, среди прочего занимавшегося вопросами математической логики в середине XIX века.

George Boole.jpg Джордж Буль Логический тип данных в Python Логические значения True (истина) и False (ложь) представляют тип данных bool. У этого типа только два возможных значения и два соответствующих литерала: True и False.

Мы активно использовали логический тип данных, когда работали с флагами:

flag = False или когда использовали условный оператор if-else:

```
a = 100 b = 17
```

```
if b > a: print('b больше a') else: print('b не больше a')
```

Результатом логического выражения `b > a` является булево значение, в данном примере False, так как значение в переменной `b` меньше значения в переменной `a`.

Логические выражения можно использовать не только в условном операторе.

Следующий программный код:

```
print(17 > 7) print(17 == 7) print(17 < 7)
```

выведет:

True False False Логический тип данных – основа информатики.

Логические операторы в Python Для создания произвольно сложных логических выражений (условий) мы используем три логические операции:

и (and); или (or); не (not). Логические операции используют операнды со значениями True и False и возвращают результат также с логическими значениями. Определённые для объектов типа bool операторы (and, or, not) известны как логические операторы и имеют общеизвестные определения:

`a and b` даёт True, если оба операнда True, и False, если хотя бы один из них False; `a or b` даёт False, если оба операнда False, и True, если хотя бы один из них True; `not a` даёт True, если `a` имеет значение False, и False, если `a` имеет значение True. Следующий программный код:

```
a = True b = False
```

```
print('a and b is', a and b) print('a or b is', a or b) print('not a is', not a)
```

выведет:

`a and b` is False `a or b` is True `not a` is False Запомните: приоритет оператора `not` выше, чем у оператора `and`, приоритет которого, в свою очередь, выше, чем у оператора `or`.

Булевы значения как числа Логические значения в Python можно трактовать как числа. Значению True соответствует число 1, в то время как значению False соответствует 0

1. Таким образом, мы можем сравнить логические значения с числами:

Следующий программный код:

```
print(True == 1) print(False == 0)
```

выведет:

True True Мы можем также применять арифметические операции к логическим значениям.

Следующий программный код:

```
print(True + True + True - False) print(True + (False / True))
```

 выведет:

3 1.0 Возможность трактовать булевы выражения как числа на практике используется не так часто. Однако есть один прием, который может оказаться полезным. Поскольку True равно 1, а False равно 0, сложение логических значений вместе – это быстрый способ подсчета количества значений True. Это может пригодиться, когда требуется подсчитать количество элементов, удовлетворяющих условию.

Следующий программный код:

```
numbers = [1, 2, 3, 4, 5, 8, 10, 12, 15, 17] res = 0
```

```
for num in numbers: res += (num % 2 == 0)
```

print(res) выведет количество четных элементов списка numbers, то есть число 5.

Примечания Примечание 1. Вместо избыточного кода:

if flag == True: программисты обычно пишут код:

if flag: Аналогично, вместо кода

if flag == False: программисты обычно пишут код:

if not flag: Примечание 2. Операторы and и or ленивые:

при вычислении логического выражения `x and y`, если `x == False`, то результат всего выражения `x and y` будет False, так что `y` не вычисляется; при вычислении логического выражения `x or y`, если `x == True`, то результат всего выражения `x or y` будет True, и `y` не вычисляется. Примечание 3. Математическая теория булевой логики определяет, что никакие другие операторы, кроме `not`, `and` и `or`, не нужны. Все остальные операторы на двух входах могут быть указаны в терминах этих трех операторов. Все операторы на трех или более входах могут быть указаны в терминах операторов двух входов.

Фактически, даже наличие пары `or` и `and` избыточно. Оператор `and` может быть определен в терминах `not` и `or`, а оператор `or` может быть определен в терминах `not` и `and`. Однако, `and` и `or` настолько полезны, что во всех языках программирования есть и то, и другое.

Примечание 4. Встроенные типы данных на английском языке.

Примечание 5. Приведем таблицы истинности для логических операторов `and`, `or`, `not`:

Примечание 6. Посмотреть фильм о Джордже Буле на русском языке можно по ссылке.

Тема урока: тип данных `NoneType` Тип данных `NoneType` Литерал `None` Сравнение `None` с другими типами данных Аннотация. Урок посвящен типу данных `NoneType`.

Пустое значение Во многих языках программирования (Java, C++, C#, JavaScript и т.д.) существует ключевое слово `null`, которое можно присвоить переменным. Концепция ключевого слова `null` заключается в том, что оно дает переменной нейтральное или "нулевое" поведение.

В языке Python, слово null заменено на None, поскольку слово null звучит не очень дружелюбно, а None относится именно к требуемой функциональности – это ничего и не имеет поведения.

Литерал None Литерал None в Python позволяет представить null переменную, то есть переменную, которая не содержит какого-либо значения. Другими словами, None – это специальная константа, означающая пустоту. Если более точно, то None – это объект специального типа данных `NoneType`.

Следующий программный код:

```
var = None print(type(var))
```

 выведет:

```
<class 'NoneType'>
```

 Мы можем присвоить значение None любой переменной, однако мы не можем самостоятельно создать другой `NoneType` объект.

Все переменные, которым присвоено значение None, ссылаются на один и тот же объект типа `NoneType`. Создание собственных экземпляров типа `NoneType` недопустимо. Объекты, существующие в единственном экземпляре, называются синглтонами.

Проверка на None Для того, чтобы проверить значение переменной на None, мы используем либо оператор `is`, либо оператор проверки на равенство `==`.

Следующий программный код:

```
var = None if var is None: # используем оператор is print('None') else: print('Not None')
```

 выведет:

None Следующий программный код:

```
var = None if var == None: # используем оператор == print('None') else: print('Not None')
```

 выведет:

None Для сравнения переменной с None всегда используйте оператор `is`. Для встроенных типов поведение `is` и `==` абсолютно одинаково, однако с пользовательскими типами могут возникнуть проблемы, так как в Python есть возможность переопределения операторов сравнения в пользовательских типах.

Сравнение None с другими типами данных Сравнение None с любым объектом, отличным от None, дает значение `False`.

Следующий программный код:

```
print(None == None)
```

 выведет:

True Следующий программный код:

```
print(None == 17) print(None == 3.14) print(None == True) print(None == [1, 2, 3]) print(None == 'Beegeek')
```

 выведет:

False False False False False Важно понимать, что следующий программный код:

```
print(None == 0) print(None == False) print(None == '')
```

 выведет:

False False False Значение None не отождествляется с значениями 0, False, ''.

Сравнивать None с другими типами данных можно только на равенство.

Следующий программный код:

`print(None > 0)` `print(None <= False)` приводит к ошибке:

`TypeError: '>' not supported between instances of 'NoneType' and 'int' ('bool')` Примечания

Примечание 1. Обратите внимание, что функции, не возвращающие значений, на самом деле, в Python возвращают значение None.

`def print_message(): print('Я - Тимур,') print('король матана.')` Мы можем вызвать функцию `print_message()` так:

`print_message()` или так:

`res = print_message()` В переменной `res` хранится значение None.

Примечание 2. Концепция null значений появилась при создании языка ALGOL W великим Чарльзом Хоаром, который позднее назвал собственную идею ошибкой на миллиард долларов. Подробнее можно почитать тут.

На конференции в EPFL 20 июня 2011 г. Чарльз Энтони Ричард Хоар Чарльз Хоар - автор одного из самых быстрых алгоритмов сортировки, основанной на сравнениях: быстрая сортировка (QuickSort).

тема урока: вложенные списки

Вложенные списки

Объявление и индексация

Функции `len()`, `max()`, `min()`

Списочные методы

Аннотация. Урок посвящен вложенным спискам, то есть спискам, входящим в качестве элементов в другие списки.

Введение

Как мы уже знаем, список представляет собой упорядоченную последовательность элементов, индексы которых пронумерованы от 0

0. Элементами списка могут быть любые типы данных – числа, строки, булевы значения и т.д. Например,

```
numbers = [10, 3]
constants = [3.1415, 2.71828, 1.1415]
countries = ['Russia', 'Armenia', 'Argentina']
flags = [True, False]
```

Список `numbers` состоит из

2

2-х элементов, и каждый из них – целое число:

```
numbers[0] == 10;
numbers[1] == 3.
```

Список `constants` состоит из

3

3-х элементов, каждый из которых – вещественное число:

```
constants[0] == 3.1415;  
constants[1] == 2.71828;  
constants[2] == 1.1415.
```

Список countries состоит из

3

3-х элементов, каждый из которых – строка:

```
countries[0] == 'Russia';  
countries[1] == 'Armenia';  
countries[2] == 'Argentina'.
```

Список flags состоит из

2

2-х элементов, и каждый из них – булево значение:

```
flags[0] == True;  
flags[1] == False.
```

Мы также говорили, что элементы списка не обязательно должны иметь одинаковый тип данных. Список может содержать значения разных типов данных:

```
info = ['Timur', 1992, 72.5]
```

Список info содержит строковое значение, целое число и число с плавающей точкой:

```
info[0] == 'Timur';  
info[1] == 1992;  
info[2] == 72.5.
```

Обычно элементы списка содержат данные одного типа, и на практике редко приходится создавать списки, содержащие элементы разных типов данных.

Вложенные списки

Оказывается, элементами списков могут быть другие списки и в реальной разработке такая конструкция оказывается очень полезной. Такие списки называются вложенными списками.

Создание вложенного списка

Работа с вложенными списками принципиально ничем не отличается от работы со списками, например, чисел или строк. Чтобы создать вложенный список, мы также перечисляем элементы через запятую в квадратных скобках:

```
my_list = [[0], [1, 2], [3, 4, 5]]
```

Переменная my_list ссылается на список, состоящий из других списков (с вложенными списками).

Поскольку глубина вложенности списка my_list равна двум, то такой список обычно называют двумерным списком. На практике, как правило, мы

работаем с двумерными списками, реже – с трехмерными.

Рассмотрим программный код:

```
my_list = [[0], [1, 2], [3, 4, 5]]
```

```
print(my_list)
print(my_list[0])
print(my_list[1])
print(my_list[2])
print(len(my_list))
```

Результатом работы такого кода будет:

```
[[0], [1, 2], [3, 4, 5]]
[0]
[1, 2]
[3, 4, 5]
3
```

Давайте взглянем на каждую строку приведенного кода поближе.

строка

1

1 создает список и присваивает его переменной my_list. Список имеет три элемента, и каждый элемент тоже является списком:

элементом my_list[0] является список [0];

элементом my_list[1] является список [1, 2];

элементом my_list[2] является список [3, 4, 5].

строка

3

3 распечатывает весь список my_list;

строка

4

4 распечатывает элемент my_list[0];

строка

5

5 распечатывает элемент my_list[1];

строка

6

6 распечатывает элемент my_list[2];

строка

7

7 распечатает количество элементов списка my_list, то есть число

3

3.

Индексация

При работе с одномерными списками мы использовали индексацию, то есть обращение к конкретному элементу по его индексу. Аналогично можно индексировать и вложенные списки:

```
my_list = ['Python', [10, 20, 30], ['Beegreek', 'Stepik']]
```

```
print(my_list[0])
print(my_list[1])
print(my_list[2])
```

Результатом работы такого кода будет:

```
Python
[10, 20, 30]
['Beegeek', 'Stepik']
```

Так как элементы списка `my_list` – строка и списки, их также можно индексировать.

Рассмотрим программный код:

```
my_list = ['Python', [10, 20, 30], ['Beegeek', 'Stepik!']]
```

```
print(my_list[0][2])      # индексирование строки 'Python'
print(my_list[1][1])      # индексирование списка [10, 20, 30]
print(my_list[2][-1])     # индексирование списка ['Beegeek',
'Stepik!']
print(my_list[2][-1][-1]) # индексирование строки 'Stepik!'
```

Результатом работы такого кода будет:

```
t
20
Stepik!
!
```

Попытка обратиться к элементу списка по несуществующему индексу:

```
print(my_list[3]) # у списка my_list индексы от 0 до 2
```

вызовет ошибку:

IndexError: index out of range

Функции `len()`, `max()`, `min()`

В прошлом курсе мы рассматривали встроенные функции `max()`, `min()`, `len()`, полезные и при работе с вложенными списками (обработке вложенных списков).

Функция `len()`

Рассмотрим программный код:

```
my_list = [[0], [1, 2], [3, 4, 5], [], [10, 20, 30]]
```

```
print(len(my_list))
```

Результатом работы такого кода будет:

```
5
```

Обратите внимание, встроенная функция `len()` возвращает количество элементов (число

```
5
```

5) списка `my_list`, а не общее количество элементов во всех списках (число 9).

Если требуется посчитать общее количество элементов во вложенном списке `my_list`, мы можем использовать цикл `for` в связке с функцией `len()`:

```
total = 0
my_list = [[0], [1, 2], [3, 4, 5], [], [10, 20, 30]]

for li in my_list:
    total += len(li)

print(total)
```

Результатом работы такого кода будет:

9

Переменная `li` последовательно принимает все значения элементов списка `my_list`, то есть является сама по себе списком, поэтому мы можем вызывать функцию `len()` с переданным аргументом `li`.

Функции `min()` и `max()`

Функции `min()` и `max()` могут работать и со списками. Если этим функциям передается несколько списков, то целиком возвращается один из переданных списков. При этом сравнение происходит поэлементно: сначала сравниваются первые элементы списков. Если они не равны, то функция `min()` вернет тот список, первый элемент которого меньше, `max()` – наоборот. Если первые элементы равны, то будут сравниваться вторые и т. д.

Рассмотрим программный код:

```
list1 = [1, 7, 12, 0, 9, 100]
list2 = [1, 7, 90]
list3 = [1, 10]
```

```
print(min(list1, list2, list3))
print(max(list1, list2, list3))
```

Результатом работы такого кода будет:

```
[1, 7, 12, 0, 9, 100]
[1, 10]
```

Функции `min()` и `max()` также можно использовать при работе с вложенными списками. Рассмотрим программный код:

```
list1 = [[1, 7, 12, 0, 9, 100], [1, 7, 90], [1, 10]]
list2 = [['a', 'b'], ['a'], ['d', 'p', 'q']]
```

```
print(min(list1))
print(max(list1))
print(min(list2))
print(max(list2))
```

Результатом работы такого кода будет:

```
[1, 7, 12, 0, 9, 100]
[1, 10]
['a']
['d', 'p', 'q']
```

Обратите внимание – элементы вложенных списков в этой ситуации должны быть сравнимы.

Таким образом, следующий код:

```
my_list = [[1, 7, 12, 0, 9, 100], ['a', 'b']]
```

```
print(min(my_list))
print(max(my_list))
```

приведет к возникновению ошибки:

```
TypeError: '<' not supported between instances of 'str' and 'int'
```

Примечания

Примечание 1. Независимо от вложенности списков, нам нужно помнить по возможности все списочные методы:

метод `append()` добавляет новый элемент в конец списка;

метод `extend()` расширяет один список другим списком;

метод `insert()` вставляет значение в список в заданной позиции;

метод `index()` возвращает индекс первого элемента, значение которого равняется переданному в метод значению;

метод `remove()` удаляет первый элемент, значение которого равняется переданному в метод значению;

метод `pop()` удаляет элемент по указанному индексу и возвращает его;

метод `count()` возвращает количество элементов в списке, значения которых равны переданному в метод значению;

метод `reverse()` инвертирует порядок следования значений в списке, то есть меняет его на противоположный;

метод `copy()` создает поверхностную копию списка.;

метод `clear()` удаляет все элементы из списка;

оператор `del` позволяет удалять элементы списка по определенному индексу.

Примечание 2. Методы строк, работающие со списками:

метод `split()` разбивает строку на слова, используя в качестве разделителя последовательность пробельных символов, символ табуляции (`\t`) или символ новой строки (`\n`).

метод `join()` собирает строку из элементов списка, используя в качестве разделителя строку, к которой применяется метод.

Примечание 3. Язык Python не ограничивает нас в уровнях вложенности: элементами списка могут быть списки, их элементами могут быть другие списки, элементами которых в свою очередь могут быть другие списки...

Тема урока: вложенные списки

Создание вложенных списков

Считывание вложенных списков

Перебор элементов вложенных списков

Обработка вложенных списков

Вывод вложенных списков

Аннотация. Урок посвящен работе с вложенными списками.

Создание вложенных списков

Для создания вложенного списка можно использовать литеральную форму записи – перечисление элементов через запятую в квадратных скобках:

```
my_list = [[0], [1, 2], [3, 4, 5]]
```

Иногда нужно создать вложенный список, заполненный по определенному правилу – шаблону. Например, список длиной `n`, содержащий списки длиной `m`, каждый из которых заполнен нулями.

Рассмотрим несколько способов решения задачи.

Способ 1. Создадим пустой список, потом `n` раз добавим в него новый элемент – список длины `m`, составленный из нулей:

```
n, m = int(input()), int(input())    # считываем значения n и m
my_list = []
```

```
for _ in range(n):
    my_list.append([0] * m)
```

```
print(my_list)
```

Если ввести значения `n = 3`, `m = 5`, то результатом работы такого кода будет:

```
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

Если передать значения `n = 5`, `m = 3`, то результатом работы такого кода будет:

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Способ 2. Сначала создадим список из `n` элементов (для начала просто из

n нулей). Затем сделаем каждый элемент списка ссылкой на другой список из m элементов, заполненный нулями:

```
n, m = int(input()), int(input()) # считываем значения n и m
my_list = [0] * n

for i in range(n):
    my_list[i] = [0] * m
```

```
print(my_list)
```

Способ 3. Можно использовать генератор списка: создадим список из n элементов, каждый из которых будет списком, состоящих из m нулей:

```
n, m = int(input()), int(input()) # считываем значения n и m
my_list = [[0] * m for _ in range(n)]
```

```
print(my_list)
```

В этом случае каждый элемент создается независимо от остальных (заново конструируется вложенный список [0] * m для заполнения очередного элемента списка).

Обратите внимание, что очевидное решение, использующее операцию умножения списка на число (операция повторения), оказывается неверным:

```
n, m = int(input()), int(input()) # считываем значения n и m
my_list = [[0] * m ] * n
```

```
print(my_list)
```

В этом легко убедиться, если присвоить элементу my_list[0][0] любое значение, например, 17, а затем вывести список на печать:

```
n, m = int(input()), int(input())

my_list = [[0] * m ] * n
my_list[0][0] = 17
```

```
print(my_list)
```

Если ввести значения n = 5, m = 3, то результатом работы такого кода будет:

```
[[17, 0, 0], [17, 0, 0], [17, 0, 0], [17, 0, 0], [17, 0, 0]]
```

То есть, изменив значение элемента списка my_list[0][0], мы также изменили значения элементов my_list[1][0], my_list[2][0], my_list[3][0], my_list[4][0].

Причина такого поведения кроется в самой природе списков (тип list). В Python списки – ссылочный тип данных. Конструкция [0] * m возвращает ссылку на список из m нулей. Повторение этого элемента создает список из n ссылок на один и тот же список.

Вложенный список нельзя создать при помощи операции повторения (умножения списка на число). Для корректного создания вложенного списка мы используем способы

```
1
-
3
1-3, отдавая предпочтение способу
3
3.
```

Считывание вложенных списков

Если элементы списка вводятся через клавиатуру (каждая строка на отдельной строке, всего n строк, числа в строке разделяются пробелами), для ввода списка можно использовать следующий код:

```
n = 4                                # количество строк
(элементов)
my_list = []

for _ in range(n):
    elem = [int(i) for i in input().split()] # создаем список из
элементов строки
    my_list.append(elem)
```

В этом примере мы используем списочный метод `append()`, передавая ему в качестве аргумента другой список. Так у нас получается список списков.

В результате, если на вход программе подаются строки:

```
2 4
6 7 8 9
1 3
5 6 5 4 3 1
```

то в переменной `my_list` будет храниться список:

```
[[2, 4], [6, 7, 8, 9], [1, 3], [5, 6, 5, 4, 3, 1]]
```

Не забывайте, что метод `split()` возвращает список строк, а не чисел. Поэтому мы предварительно сконвертировали строку в число, с помощью вызова функции `int()`.

Также следует помнить отличие работы списочных методов `append()` и `extend()`.

Следующий код:

```
n = 4
my_list = []

for _ in range(n):
    elem = [int(i) for i in input().split()]
```

```
my_list.extend(elem)
```

создает одномерный (!) список, а не вложенный. В переменной `my_list` будет храниться список:

```
[2, 4, 6, 7, 8, 9, 1, 3, 5, 6, 5, 4, 3, 1]
```

Перебор и вывод элементов вложенного списка

Как мы уже знаем, для доступа к элементу списка указывают индекс этого элемента в квадратных скобках. В случае двумерных вложенных списков надо указать два индекса (каждый в отдельных квадратных скобках), в случае трехмерного списка — три индекса и т. д.

Рассмотрим программный код:

```
my_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
print(my_list[0][0])
```

```
print(my_list[1][2])
```

```
print(my_list[2][1])
```

Результатом работы такого кода будет:

```
1
```

```
6
```

```
8
```

Когда нужно перебрать все элементы вложенного списка (например, чтобы вывести их на экран), обычно используются вложенные циклы.

Рассмотрим программный код:

```
my_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
for i in range(len(my_list)):
```

```
    for j in range(len(my_list[i])):
```

```
        print(my_list[i][j], end=' ') # используем необязательный
```

```
параметр end
```

```
        print()
```

```
# перенос на новую строку
```

Результатом работы такого кода будет:

```
1 2 3
```

```
4 5 6
```

```
7 8 9
```

Вызов функции `print()` с пустыми параметрами нужен для того, чтобы переносить вывод на новую строку, после того как будет распечатан очередной элемент (список) вложенного списка.

В предыдущем примере мы перебирали индексы элементов, а можно сразу перебирать сами элементы вложенного списка:

```
my_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
for row in my_list:
```

```
for elem in row:
    print(elem, end=' ')
print()
```

Результатом работы такого кода будет:

```
1 2 3
4 5 6
7 8 9
```

Перебор элементов вложенного списка по индексам дает нам больше гибкости для вывода данных. Например, поменяв порядок переменных *i* и *j*, мы получаем иной тип вывода:

```
my_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
for i in range(len(my_list)):
    for j in range(len(my_list[i])):
        print(my_list[j][i], end=' ') # выводим my_list[j][i] вместо
my_list[i][j]
    print()
```

Результатом работы такого кода будет:

```
1 4 7
2 5 8
3 6 9
```

Обработка вложенных списков

Для обработки элементов вложенного списка, так же как и для вывода его элементов на экран, как правило, используются вложенные циклы.

Используем вложенный цикл для подсчета суммы всех чисел в списке:

```
my_list = [[1, 9, 8, 7, 4], [7, 3, 4], [2, 1]]
```

```
total = 0
for i in range(len(my_list)):
    for j in range(len(my_list[i])):
        total += my_list[i][j]
```

```
print(total)
```

Или то же самое с циклом не по индексу, а по значениям:

```
my_list = [[1, 9, 8, 7, 4], [7, 3, 4], [2, 1]]
```

```
total = 0
for row in my_list:
    for elem in row:
        total += elem
```

```
print(total)
```

Таким образом, можно обработать элементы вложенного списка практически в любом языке программирования. В Python, однако, можно упростить код,

если использовать встроенную функцию `sum()`, которая принимает список чисел и возвращает его сумму. Подсчет суммы с помощью функции `sum()` выглядит так:

```
my_list = [[1, 9, 8, 7, 4], [7, 3, 4], [2, 1]]
```

```
total = 0
for row in my_list: # в один цикл
    total += sum(row)
print(total)
```

Названия переменных `row` (строка) и `elem` (элемент) удобно использовать при переборе вложенного списка по значениям. Названия переменных `i` и `j` используются при переборе вложенного списка по индексам.

File "<tokenize>", line 140

Результатом работы такого кода будет:

IndentationError: unindent does not match any outer indentation level

Список по образцу 1

На вход программе подается число

0

n. Напишите программу, которая создает и выводит построчно список, состоящий из

0

n списков `[[1, 2, ..., n], [1, 2, ..., n], ..., [1, 2, ..., n]]`.

Формат входных данных

На вход программе подается натуральное число

0

n.

Формат выходных данных

Программа должна вывести построчно указанный список.

Тестовые данные

```
# put your python code here
```

```
n=int(input())
```

```
y=[]
```

```
list=[]
```

```
for i in range(1,n+1):
```

```
    y.append(i)
```

```
    list.append(y)
```

```
for i in list:
```

```
    print(i,end="\n")
```

Список по образцу 2

На вход программе подается число



n. Напишите программу, которая создает и выводит построчно вложенный список, состоящий из



n списков `[[1], [1, 2], [1, 2, 3], ..., [1, 2, ..., n]]`.

Формат входных данных

На вход программе подается натуральное число



n.

Формат выходных данных

Программа должна вывести построчно указанн

put your python code here

```
n = int(input())
r = []
for i in range(1, n+1):
    r.append(list(range(1,i+1)))
print(*r, sep="\n")
```

Треугольник Паскаля 1

Треугольник Паскаля — бесконечная таблица биномиальных коэффициентов, имеющая треугольную форму. В этом треугольнике на вершине и по бокам стоят единицы. Каждое число равно сумме двух расположенных над ним чисел.

```
0:      1
1:     1 1
2:    1 2 1
3:   1 3 3 1
4:  1 4 6 4 1
      . . . . .
```

На вход программе подается число



n. Напишите программу, которая возвращает указанную строку треугольника Паскаля в виде списка (нумерация строк начинается с нуля).

Формат входных данных

На вход программе подается число



```
(

≥
0
)
n (n ≥ 0).
```

Формат выходных данных

Программа должна вывести указанную строку треугольника Паскаля в виде списка.

Примечание 1. Решение удобно оформить в виде функции `pascal()`, которая принимает номер строки и возвращает соответствующую строку треугольника Паскаля.

```
n = int(input())+1
d=[]
for row in range(n):
    row=[0]* (row+1)
    d.append(row)
for row in range(n):
    for elem in range(len(d[row])):
        d[row][0]=1

        if len(d[row]) > 1:
            d[row][(len(d[row])-1)]=1
        if d[row][elem]==0:
            d[row][elem]=d[row-1][elem-1]+d[row-1][elem]
print(d[n-1])
```

Треугольник Паскаля 2

На вход программе подается натуральное число

?

n. Напишите программу, которая выводит первые

?

n строк треугольника Паскаля.

Формат входных данных

На вход программе подается число

?

(

?

≥

1

)

n ($n \geq 1$).

Формат выходных данных

Программа должна вывести первые

?

n строк треугольника Паскаля, каждую на отдельной строке в соответствии с образцом.

put your python code here

```
n = int(input())
```

```
d=[]
```

```

for row in range(n):
    row=[0]* (row+1)
    d.append(row)
for row in range(n):
    for elem in range(len(d[row])):
        d[row][0]=1

        if len(d[row]) > 1:
            d[row][(len(d[row])-1)]=1
        if d[row][elem]==0:
            d[row][elem]=d[row-1][elem-1]+d[row-1][elem]
    print(*d[elem])

```

Упаковка дубликатов

На вход программе подается строка текста, содержащая символы. Напишите программу, которая упаковывает последовательности одинаковых символов заданной строки в подсписки.

Формат входных данных

На вход программе подается строка текста, содержащая символы, отделенные символом пробела.

Формат выходных данных

Программа должна вывести указанный вложенный список.

```

# put your python code here
s=input().split()
i=1
while i <len(s):
    if s[i] == s[i-1] or s[i] in s[i-1]:
        s[i-1]=list(s[i-1])
        s[i-1].append(s[i])
        s.remove(s[i])
        i=i-1
    else:
        s[i-1]=list(s[i-1])
        s[i]=list(s[i])
        i=i+1
print(s)

```

Разбиение на чанки

На вход программе подаются две строки: на одной – символы, на другой – число

🔗

n. Из первой строки формируется список.

Реализуйте функцию chunked(), которая принимает на вход список и число, задающее размер чанка (куска), а возвращает список из чанков (кусков) указанной длины.

Формат входных данных

На вход программе подается строка текста, содержащая символы, отделенные символом пробела и число

0

n на отдельной строке.

Формат выходных данных

Программа должна вывести указанный вложенный список.

Примечание. Не забудьте вызвать функцию `chunked()`, чтобы вывести результат ☺.

Подписки списка

Подсписок — часть другого списка. Подсписок может содержать один элемент, несколько или даже ни одного. Например, [1], [2], [3] и [4] — подписки списка [1, 2, 3, 4]. Список [2, 3] — подсписок списка [1, 2, 3, 4], но список [2, 4] не подсписок списка [1, 2, 3, 4], так как элементы

2

2 и

4

4 во втором списке не смежные (т. к. они разрываются элементом

3

3). Пустой список — подсписок любого списка. Сам список — подсписок самого себя, то есть список [1, 2, 3, 4] подсписок списка [1, 2, 3, 4].

На вход программе подается строка текста, содержащая символы. Из данной строки формируется список. Напишите программу, которая выводит список, содержащий все возможные подписки списка, включая пустой список.

Тема урока: матрицы Работа с матрицами Квадратные и прямоугольные матрицы Функции `ljust()` и `rjust()` Главная и побочная диагонали Аннотация. Урок посвящен работе с матрицами — прямоугольными таблицами.

Матрицы В прошлых уроках мы изучили вложенные списки, то есть списки, входящие в качестве элементов в другие списки. Частный случай вложенных списков — матрицы. Это прямоугольные таблицы, заполненные какими-то значениями, обычно числами.

1.png

Матрицы часто применяются в математике, так как многие задачи с их помощью гораздо проще сформулировать, записать и решить.

Для работы с матрицами нужно уметь получать элемент i -й строки j -го столбца. Для этого обычно заводят список строк матрицы, где каждая строка — список элементов. Получается вложенный список или список списков. Теперь, чтобы получить определенный элемент, достаточно из списка строк матрицы выбрать i -ю и взять j -й элемент этой строки.

Давайте заведем матрицу размера 3×4 (3 строки и 4 столбца), содержащую числа, и получим элемент на позиции (2,

3) (2, 3), то есть элемент второй строки в третьем столбце.

```
matrix = [[2, -5, -11, 0], [-9, 4, 6, 13], [4, 7, 12, -2]]
```

```
print(matrix[1][2])
```

 # вывод элемента на позиции (2, 3) В переменной matrix хранится вся матрица, при этом matrix[1] — список значений во второй строке, matrix[1][2] — элемент в третьем столбце этой строки.

2.png

В математике нумерация строк и столбцов начинается с единицы, а не с нуля. По договоренности сначала всегда указывается строка, а затем — столбец. Элемент на i -ой строке, j -м столбце матрицы a обозначается так — a_{ij} .

Перебор элементов матрицы Чтобы перебрать элементы матрицы, необходимо использовать вложенные циклы. Например, выведем на экран все элементы матрицы, перебирая их по строкам:

```
rows, cols = 3, 4 # rows - количество строк, cols - количество столбцов
```

```
matrix = [[2, 3, 1, 0], [9, 4, 6, 8], [4, 7, 2, 7]]
```

```
for r in range(rows): for c in range(cols): print(matrix[r][c], end=' ') print()
```

 Результатом работы такого кода будет:

2 3 1 0 9 4 6 8 4 7 2 7 Для перебора элементов матрицы по столбцам можно использовать следующий код:

```
rows, cols = 3, 4 # rows - количество строк, cols - количество столбцов
```

```
matrix = [[2, 3, 1, 0], [9, 4, 6, 8], [4, 7, 2, 7]]
```

```
for c in range(cols): for r in range(rows): print(matrix[r][c], end=' ') print()
```

 Результатом работы такого кода будет:

2 9 4 3 4 7 1 6 2 0 8 7 Функции ljust() и rjust() Рассмотрим программный код:

```
rows, cols = 3, 4 # rows - количество строк, cols - количество столбцов
```

```
matrix = [[277, -930, 11, 0], [9, 43, 6, 87], [4456, 8, 290, 7]]
```

```
for r in range(rows): for c in range(cols): print(matrix[r][c], end=' ') print()
```

 Результатом работы такого кода будет:

277 -930 11 0 9 43 6 87 4456 8 290 7 Выведенная матрица не сильно похожа на упорядоченный прямоугольник. Элементы матрицы имеют разное количество разрядов и результат вывода получается смазанным. Для решения проблемы удобно использовать строковые методы ljust() и rjust().

Метод ljust() Строковый метод ljust() выравнивает текст по ширине, добавляя пробелы в конец текста.

Результатом выполнения следующего кода:

```
print('a'.ljust(3)) print('ab'.ljust(3)) print('abc'.ljust(3))
```

 будет:

a_ ab_ abc Исходная строка не обрезается, даже если в ней больше символов, чем нужно.

Результатом выполнения следующего кода:

```
print('abcdefg'.ljust(3))
```

 будет:

abcdefg Строковый метод `ljust()` использует вместо пробела другой символ, если передать ему второй аргумент, необязательный.

Результатом выполнения следующего кода:

```
print('a'.ljust(5, '*')) print('ab'.ljust(5, '$')) print('abc'.ljust(5, '#'))
```

 будет:

a**** ab\$\$\$ abc### Метод `rjust()` Строковый метод `rjust()` выравнивает текст по ширине, добавляя пробелы в начало текста.

Результатом выполнения следующего кода:

```
print('a'.rjust(3)) print('ab'.rjust(3)) print('abc'.rjust(3))
```

 будет:

_a _ab abc Исходная строка не обрезается, даже если в ней больше символов, чем нужно.

Результатом выполнения следующего кода:

```
print('abcdefg'.rjust(3))
```

 будет:

abcdefg Строковый метод `rjust()` использует вместо пробела другой символ, если передать ему второй аргумент, необязательный.

Результатом выполнения следующего кода:

```
print('a'.rjust(5, '*')) print('ab'.rjust(5, '$')) print('abc'.rjust(5, '#'))
```

 будет:

****a \$\$\$ab

##abc Применив метод `ljust()` для выравнивания столбцов, при выводе таблицы мы получим следующий код:

```
rows, cols = 3, 4 # rows - количество строк, cols - количество столбцов
```

```
matrix = [[277, -930, 11, 0], [9, 43, 6, 87], [4456, 8, 290, 7]]
```

```
for r in range(rows): for c in range(cols): print(str(matrix[r][c]).ljust(6), end='') print()
```

Результатом выполнения такого кода будет:

277 -930 11 0

9 43 6 87

4456 8 290 7

Квадратные матрицы Матрица с одинаковым количеством строк и столбцов называется квадратной. У квадратной матрицы есть две диагонали:

главная: проходит из верхнего левого в правый нижний угол матрицы; побочная: проходит из нижнего левого в правый верхний угол матрицы.

3.png

Элементы с равными индексами $i == j$ находятся на главной диагонали. Такие элементы обозначаются `matrix[i][i]`.

Элементы с индексами i и j , связанными соотношением $i + j + 1 = n$ (или $j = n - i - 1$), где n — размерность матрицы, находятся на побочной диагонали.

Таким образом, чтобы установить элементы главной или побочной диагонали, достаточно одного цикла.

Результатом выполнения следующего кода:

```
n = 8 matrix = [[0]*n for _ in range(n)] # создаем квадратную матрицу размером 8×8
```

```
for i in range(n): # заполняем главную диагональ единицами, а побочную двойками matrix[i][i] = 1 matrix[i][n-i-1] = 2
```

```
for r in range(n): # выводим матрицу for c in range(n): print(matrix[r][c], end=' ') print() будет:
```

```
1 0 0 0 0 0 2 0 1 0 0 0 0 2 0 0 0 1 0 0 2 0 0 0 0 0 1 2 0 0 0 0 0 0 2 1 0 0 0 0 0 2 0 0 1 0 0 0 2 0 0 0
0 1 0 2 0 0 0 0 0 0 1 Индексы i и j элементов на главной диагонали связаны соотношением i = j. Индексы i и j элементов на побочной диагонали связаны соотношением i + j + 1 = n (или j = n - i - 1), где n — размерность матрицы.
```

Заметим также, что:

если элемент находится выше главной диагонали, то $i < j$, если ниже — $i > j$. если элемент находится выше побочной диагонали, то $i + j + 1 < n$, если ниже — $i + j + 1 > n$. Примечания
Примечание 1. Чтобы понять, в какой области лежит элемент можно воспользоваться следующей картинкой.

4.png

Примечание 2. Используйте функцию `print_matrix()` для вывода квадратной матрицы размерности n :

```
def print_matrix(matrix, n, width=1):
    for r in range(n):
        for c in range(n):
            print(str(matrix[r][c]).ljust(width), end=' ')
        print()
```

Примечание 3. Для считывания матрицы из n строк, заполненной числами, удобно использовать следующий код:

```
n = int(input())
matrix = []
for i in range(n):
    temp = [int(num) for num in input().split()]
    matrix.append(temp)
```

Вывести матрицу 1

На вход программе подаются два натуральных числа

❖

n и

❖

m, каждое на отдельной строке — количество строк и столбцов в матрице. Далее вводятся сами элементы матрицы — слова, каждое на отдельной строке; подряд идут элементы сначала первой строки, затем второй, и т.д.

Напишите программу, которая сначала считывает элементы матрицы один за другим, затем выводит их в виде матрицы.

Формат входных данных

На вход программе подаются два числа

❖

n и

❖

m — количество строк и столбцов в матрице, далее идут

❖

x

❖

n×m слов, каждое на отдельной строке.

Формат выходных данных

Программа должна вывести считанную матрицу, разделяя ее элементы одним пробелом.

```
d=[]
row=int(input())
colon=int(input())
for i in range(row):
    d.append([0]*colon)
for i in range(row):
    for j in range(colon):
        d[i][j]=input()
for row in d:
    print(*row)
```

Вывести матрицу 2 На вход программе подаются два натуральных числа ❖ n и ❖ m, каждое на отдельной строке — количество строк и столбцов в матрице. Далее вводятся сами элементы матрицы — слова, каждое на отдельной строке; подряд идут элементы сначала первой строки, затем второй, и т.д.

Напишите программу, которая считывает элементы матрицы один за другим, выводит их в виде матрицы, выводит пустую строку, и снова ту же матрицу, но уже поменяв местами строки со столбцами: первая строка выводится как первый столбец, и так далее.

Формат входных данных На вход программе подаются два числа ❖ n и ❖ m — количество строк и столбцов в матрице, далее идут ❖ × ❖ n×m слов, каждое на отдельной строке.

Формат выходных данных Программа должна вывести считанную матрицу, за ней пустую строку и ту же матрицу, но поменяв местами строки со столбцами. Элементы матрицы разделять одним пробелом.

```
# put your python code here
d=[]
row=int(input())
colon=int(input())
for i in range(row):
    d.append([0]*colon)
for i in range(row):
    for j in range(colon):
        d[i][j]=input()
for row in d:
    print(*row)
print()
for i in range(len(d[0])):
    for j in range(len(d)):
        print(d[j][i],end=" ")
    print()
```

След матрицы

Следом квадратной матрицы называется сумма элементов главной диагонали. Напишите программу, которая выводит след заданной квадратной матрицы.

Формат входных данных

На вход программе подаётся натуральное число

0

n – количество строк и столбцов в матрице, затем элементы матрицы (целые числа) построчно через пробел.

Формат выходных данных

Программа должна вывести одно число – след заданной матрицы.

```
# put your python code here
s=int(input())
d=[]
summa=0
for i in range(s):
    d.append(input().split())
    summa=int(d[i][i])+summa
print(summa)
```

Больше среднего

Напишите программу, которая выводит количество элементов квадратной матрицы в каждой строке, больших среднего арифметического элементов данной строки.

Формат входных данных

На вход программе подаётся натуральное число



n — количество строк и столбцов в матрице, затем элементы матрицы (целые числа) построчно через пробел.

Формат выходных данных


Программа должна вывести



n чисел — для каждой строки количество элементов матрицы, больших среднего арифметического элементов данной строки.

```
n=int(input())
d=[]
for i in range(n):
    d.append(list(map(int,input().split())))
for i in d:
    s=0
    sum_=sum(i)/n
    for j in i:
        if j > sum_:
            s=s+1
    print(s)
```

Максимальный в области 1 Напишите программу, которая выводит максимальный элемент в заштрихованной области квадратной матрицы.

Формат входных данных На вход программе подаётся натуральное число  n — количество строк и столбцов в матрице, затем элементы матрицы (целые числа) построчно через пробел.

Формат выходных данных Программа должна вывести одно число — максимальный элемент в заштрихованной области квадратной матрицы.

1.png

```
# put your python code here
n=int(input())
d=[]
for i in range(n):
    d.append(list(map(int,input().split())))
m=[]
for i in range(n):
    m.append(max(d[i][0:i+1]))
print(max(m))
```

Максимальный в области 2

Напишите программу, которая выводит максимальный элемент в заштрихованной области квадратной матрицы.

Формат входных данных

На вход программе подаётся натуральное число



n — количество строк и столбцов в матрице, затем элементы матрицы (целые числа) построчно через пробел.

Формат выходных данных

Программа должна вывести одно число — максимальный элемент в заштрихованной области квадратной матрицы.


Примечание. Элементы диагоналей также учитываются.

1.png

```
n=int(input())
a=[]
s=[]
d=[]
for i in range(n):
    s.append(list(map(int,input().split())))
for i in range(n):
    for j in range(n):
        if i >= j and i <= n-1-j:
            d.append(s[i][j])
        elif i <= j and i >= n-1-j:
            d.append(s[i][j])
print(max(d))
```

Суммы четвертей Квадратная матрица разбивается на четыре четверти, ограниченные главной и побочной диагоналями: верхнюю, нижнюю, левую и правую.

Напишите программу, которая вычисляет сумму элементов: верхней четверти; правой четверти; нижней четверти; левой четверти.

Формат входных данных На вход программе подаётся натуральное число  n — количество строк и столбцов в матрице, затем элементы матрицы (целые числа) построчно через пробел.

Формат выходных данных Программа должна вывести текст в соответствии с условием задачи.

Примечание. Элементы диагоналей не учитываются.

1.png

Таблица умножения

На вход программе подаются два натуральных числа



n и m — количество строк и столбцов в матрице. Создайте матрицу `mult` размером $n \times m$ и заполните её таблицей умножения по формуле `mult[i][j] = i * j`.

Формат входных данных

На вход программе на разных строках подаются два числа

n и m

m — количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести таблицу умножения отводя на вывод каждого числа ровно

3

3 символа (для этого используйте строковый метод `ljust()`).

put your python code here

```
mult=[]
n=int(input())
m=int(input())
for i in range(n):
    s=[]
    for j in range(m):
        s.append(i*j)
    mult.append(s)
for i in range(n):
    for j in range(m):
        print (str(mult[i][j]).ljust(3),end=" ")
    print()
```

Максимум в таблице На вход программе подаются два натуральных числа n и m — количество строк и столбцов в матрице, затем n строк по m целых чисел в каждой, отделенных символом пробела.

Напишите программу, которая находит индексы (строку и столбец) первого вхождения максимального элемента.

Формат входных данных На вход программе на разных строках подаются два числа n и m — количество строк и столбцов в матрице, затем сами элементы матрицы построчно через пробел.

Формат выходных данных Программа должна вывести два числа: номер строки и номер столбца, в которых стоит наибольший элемент таблицы. Если таких элементов несколько,

то выводится тот, у которого меньше номер строки, а если номера строк равны то тот, у которого меньше номер столбца.

Примечание. Считайте, что нумерация строк и столбцов начинается с нуля.

```
# put your python code here
n=int(input())
m=int(input())
#d=[[4, 3, 4 ,4, 1 ,2, 2, 3],[2 ,3 ,0 ,3 ,3,4,4,5]]
d=[]
for i in range(n):
    d.append(list(map(int,input().split())))
#print(d)
max_=d[0][0]
for i in d:
    if max(i) > max_:
        max_=max(i)
#print(max_)
flag=True
for i in range(n):
    if flag == False:
        break
    for j in range(m):
        if d[i][j] == max_:
            g=i
            w=j
            flag=False
            print(g,w)
            break
```

Обмен столбцов

Напишите программу, которая меняет местами столбцы в матрице.

Формат входных данных

На вход программе на разных строках подаются два натуральных числа

0

n и

0

m – количество строк и столбцов в матрице, затем элементы матрицы
построчно через пробел, затем числа

0

i и

0

j – номера столбцов, подлежащих обмену.

Формат выходных данных

Программа должна вывести указанную таблицу с замененными столбцами.

```
n= int(input())
m=int(input())
```

```

d=[]
#d = [[11, 12 ,13 ,14]
#      ,[21, 22, 23, 24]
#      ,[31 ,32, 33,34]]
for i in range(n):
    d.append(list(map(int,input().split())))
f=[]
for i in range (n):
    f.append([0]*m)
s=list(map(int,input().split()))
#print(s)
for i in range(n):
    for j in range(m):
        f[i][j]=d[i][j]
        f[i][s[0]]=d[i][s[1]]
        f[i][s[1]]=d[i][s[0]]
for j in f:
    print(*j)

```

Симметричная матрица

Напишите программу, которая проверяет симметричность квадратной матрицы относительно главной диагонали.

Формат входных данных

На вход программе подаётся натуральное число



n – количество строк и столбцов в матрице, затем элементы матрицы построчно через пробел.

Формат выходных данных

Программа должна вывести YES, если матрица симметрична относительно главной диагонали, и слово NO в противном случае.

Тестовые данные 

```

# put your python code here
n=int(input())
count=0
d=[]
for i in range(n):
    d.append(list(map(int,input().split())))
for i in range(n):
    for j in range(n):
        if d[i][j]==d[j][i]:
            count=count+1
if count == n*n:
    print("YES")
else:
    print("NO")

```

Обмен диагоналей

Дана квадратная матрица чисел. Напишите программу, которая меняет местами элементы, стоящие на главной и побочной диагонали, при этом каждый элемент должен остаться в том же столбце (то есть в каждом столбце нужно поменять местами элемент на главной диагонали и на побочной диагонали).

Формат входных данных

На вход программе подаётся натуральное число

🔗

n — количество строк и столбцов в матрице, затем элементы матрицы построчно через пробел.

Формат выходных данных

Программа должна вывести матрицу с элементами главной и побочной диагонали, поменявшимися своими местами.

Тестовые данные 📄

```
n=int(input())
d=[]
for i in range(n):
    d.append(list(map(int,input().split())))
for i in range(n):
    d[i][i],d[n-1-i][i] = d[n-1-i][i],d[i][i]
for s in d:
    print(*s)
```

Зеркальное отображение Дана квадратная матрица чисел. Напишите программу, которая зеркально отображает её элементы относительно горизонтальной оси симметрии.

Формат входных данных На вход программе подаётся натуральное число 🔗 n —

количество строк и столбцов в матрице, затем элементы матрицы построчно через пробел.

Формат выходных данных Программа должна вывести матрицу, в которой зеркально отображены элементы относительно горизонтальной оси симметрии.

Тестовые данные 📄

```
# put your python code here
n=int(input())
d=[]
for i in range (n):
    d.append(list(map(int,input().split())))
s=d[::-1]
for i in s:
    print(*i)
```

Поворот матрицы

Напишите программу, которая поворачивает квадратную матрицу чисел на

9
0
°
90
°

по часовой стрелке.

Формат входных данных

На вход программе подаётся натуральное число

0

n – количество строк и столбцов в матрице, затем элементы матрицы построчно через пробел.

Формат выходных данных

Программа должна вывести результат на экран, числа должны быть разделены одним пробелом.

Тестовые данные

put your python code here

```
n=int(input())
f=[]
d=[]
for i in range (n):
    f.append(list(map(int,input().split())))
f=f[::-1]
for i in range(n):
    d.append([0]*n)
for i in range(n):
    for j in range(n):
        d[i][j]=f[j][i]
for i in d:
    print(*i)
```

Ходы коня

На шахматной доске

8
x
8

8×8 стоит конь. Напишите программу, которая отмечает положение коня на доске и все клетки, которые бьёт конь. Клетку, где стоит конь, отметьте английской буквой N, а клетки, которые бьёт конь, отметьте символами *, остальные клетки заполните точками.

Формат входных данных

На вход программе подаются координаты коня на шахматной доске в шахматной нотации (то есть в виде e4, где сначала записывается номер столбца (буква от a до h, слева направо), затем номеру строки (цифра от

1

```
1 до
8
8, снизу вверх)).
```

Формат выходных данных

Программа должна вывести на экран изображение доски, разделяя элементы пробелами.

```
# put your python code here
col_,str_=input()
col_=ord(col_)-97
str_=int(str_)
d=[]
for i in range(8):
    d.append([0]*8)
for i in range(8):
    for j in range(8):
        y = ((8-str_)- j) * (col_ - i)
        d[8-str_][col_]="N"
        if y==2 or y == -2:
            d[j][i]="*"
        else:
            d[j][i]="."
for i in d:
    print(*i)
```

Магический квадрат Магическим квадратом порядка n называется квадратная таблица размера $n \times n$, составленная из всех чисел $1, 2, 3, \dots, n^2$ (то есть все числа разные) так, что суммы по каждому столбцу, каждой строке и каждой из двух диагоналей равны между собой. Напишите программу, которая проверяет, является ли заданная квадратная матрица магическим квадратом.

Формат входных данных На вход программе подаётся натуральное число n — количество строк и столбцов в матрице, затем элементы матрицы: n строк, по n чисел в каждой, разделённые пробелами.

Формат выходных данных Программа должна вывести слово YES, если матрица является магическим квадратом, и слово NO в противном случае.

```
# put your python code here
n=int(input())
f=[]
f_=[]
for i in range(n):
    f.append(list(map(int,input().split())))
sum_str=[]
colon=0
sum_glav=0
sum_anoth=0
sum_colon=[]
```

```

flag=True
for i in f:
    sum_str.append(sum(i))
for i in range(n):
    colon=0
    for j in range(n):
        f_.append(f[j][i])
        colon=colon+f[j][i]
    sum_colon.append(colon)
test_sum_str=sum(sum_str)
for i in sum_str:
    if i*n != test_sum_str :
        flag=False
test_sum_colon=sum(sum_colon)
for i in sum_colon:
    if i*n != test_sum_colon :
        flag=False

for i in range(n):
    for j in range(n):
        if i==j :
            sum_glav=f[i][j]+sum_glav
        if i+j+1 == n:
            sum_anoth=sum_anoth+f[i][j]
for i in range(len(f_)):

    if f_.count(f_[i]) > 1:
        flag=False
for i in [1,2,3,4,5,6,7,8,9]:
    if i not in f_:
        flag=False

#print(sum_str,sum_colon,sum_glav,sum_anoth)
for i , j in zip(sum_str ,sum_colon):
    if i != j:
        flag=False
if sum_glav != sum_anoth:
    flag=False
if flag==True:
    print("YES")
else:
    print("NO")

```

Шахматная доска

На вход программе подаются два натуральных числа

0

n и

0

m. Напишите программу для создания матрицы размером

0

×

0

$n \times m$, заполнив её символами . и * в шахматном порядке. В левом верхнем углу должна стоять точка. Выведите полученную матрицу на экран, разделяя элементы пробелами.

Формат входных данных

На вход программе на одной строке через пробел подаются два натуральных числа

0

n и

0

m – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести матрицу, описанную в условии задачи.

```
nm= input().split()
n=int(nm[0])
m=int(nm[1])
d=[]
for a in range(n):
    d.append(["*"]*m)
for i in range(n):
    for j in range(m):
        if (i+j) % 2 == 0:
            d[i][j]="."
for i in d:
    print(*i,sep=" ")
```

Побочная диагональ

На вход программе подается натуральное число

0

n. Напишите программу, которая создает матрицу размером

0

×

0

$n \times n$ и заполняет её по следующему правилу:

числа на побочной диагонали равны

1

1;

числа, стоящие выше этой диагонали, равны

0

0;

числа, стоящие ниже этой диагонали, равны

2

2.

Полученную матрицу выведите на экран. Числа в строке разделяйте одним пробелом.

Формат входных данных

На вход программе подается натуральное число

0

n – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести матрицу в соответствии с условием задачи.

Примечание. Побочная диагональ – это диагональ, идущая из правого верхнего в левый нижний угол матрицы.

put your python code here

```
n=int(input())
d=[]
for i in range(n):
    d.append([1]*n)
for i in range(n):
    for j in range(n):
        if i+j+1 < n:
            d[i][j]=0
        elif i+j+1>n:
            d[i][j]=2
for i in d:
    print(*i)
```

Заполнение 1

На вход программе подаются два натуральных числа

0

n и

0

m . Напишите программу, которая создает матрицу размером

0

x

0

$n \times m$ и заполняет её числами от

1

1 до

0

.

0

$n \cdot m$ в соответствии с образцом.

Формат входных данных

На вход программе на одной строке подаются два натуральных числа

0

n и

0

m – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести матрицу в соответствии с образцом.

Примечание. Для вывода элементов матрицы как в примерах отводите ровно

3

3 символа на каждый элемент. Для этого используйте строковый метод `ljust()`. Можно обойтись и без `ljust()`, система примет и такое решение 😊

put your python code here

```
d=[]
s=0
n,m=input().split()
n,m=int(n),int(m)
for i in range(n):
    d.append([0]*m)
len_=len(str(n*m))+1
for i in range (n):
    for j in range(m):
        s=s+1
        d[i][j]=str(s)
for i in range(n):
    for j in range(m):
        print(d[i][j].ljust(len_),end = "")
    print()
```

Заполнение 2

На вход программе подаются два натуральных числа

0

n и

0

m. Напишите программу, которая создает матрицу размером

0

x

0

n×m, заполнив её в соответствии с образцом.

Формат входных данных

На вход программе на одной строке подаются два натуральных числа

0

n и

0

m – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести указанную матрицу в соответствии с образцом.

Примечание. Для вывода элементов матрицы как в примерах отводите ровно

3

3 символа на каждый элемент. Для этого используйте строковый метод

ljust(). Можно обойтись и без ljust(), система примет и такое решение 😊

put your python code here

```
d=[]
s=0
n,m=input().split()
n,m=int(n),int(m)
for i in range(n):
    d.append([0]*m)
len_=len(str(n*m))+1
for j in range(m):
    for i in range(n):
        s=s+1
        d[i][j]=str(s)
for i in range(n):
    for j in range(m):
        print(d[i][j].ljust(len_),end = "")
    print()
```

Заполнение 3

На вход программе подается натуральное число

🕒

n. Напишите программу, которая создает матрицу размером

🕒

x

🕒

n×n, заполнив её в соответствии с образцом.

Формат входных данных

На вход программе подается натуральное число

🕒

n – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести указанную матрицу в соответствии с образцом: разместить единицы на главной и побочной диагоналях, остальные позиции матрицы заполнить нулями.

Примечание. Для вывода элементов матрицы как в примерах отводите ровно 3

3 символа на каждый элемент. Для этого используйте строковый метод ljust(). Можно обойтись и без ljust(), система примет и такое решение 😊

```
n =int(input())
d=[]
for i in range(n):
    d.append([1]*n)
```

```

for i in range(n):
    for j in range(n):
        if i+j+1 < n:
            d[i][j]=0
        elif i+j+1>n:
            d[i][j]=0
        if [i] ==[j]:
            d[i][j]=1
for i in d:
    print(*i)

```

Заполнение 4

На вход программе подается натуральное число

0

n. Напишите программу, которая создает матрицу размером

0

x

0

n×n, заполнив её в соответствии с образцом.

Формат входных данных

На вход программе подается натуральное число

0

n – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести указанную матрицу в соответствии с образцом.

Примечание. Для вывода элементов матрицы как в примерах отводите ровно

3

3 символа на каждый элемент. Для этого используйте строковый метод ljust(). Можно обойтись и без ljust(), система примет и такое решение 😊

put your python code here

```

n=int(input())
d=[]
for i in range(n):
    d.append([0]*n)
for i in range(n):
    for j in range(n):
        if i <= j and i <= n-1-j:
            d[i][j]=1
        elif i >= j and i >= n-1-j:
            d[i][j]=1

for i in d:
    print(*i)

```

Заполнение 5 На вход программе подаются два натуральных числа n и m

m . Напишите программу, которая создает матрицу размером $n \times m$, заполнив её в соответствии с образцом.

Формат входных данных На вход программе на одной строке подаются два натуральных числа n и m — количество строк и столбцов в матрице.

Формат выходных данных Программа должна вывести указанную матрицу в соответствии с образцом.

Примечание. Для вывода элементов матрицы как в примерах отводите ровно 3 символа на каждый элемент. Для этого используйте строковый метод `ljust()`. Можно обойтись и без `ljust()`, система примет и такое решение ☺

```
d=[]
s=0
n,m=input().split()
n,m=int(n),int(m)
for i in range(n):
    d.append([0]*m)
for i in range(n):
    for j in range(m):
        d[i][j]=i+j+1
        while d[i][j] > m:
            d[i][j]=d[i][j]-m
for i in range(n):
    for j in range(m):
        print(d[i][j],end = " ")
    print()
```

Заполнение змейкой

На вход программе подаются два натуральных числа

n

и

m .

Напишите программу, которая создает матрицу размером

n

\times

m

$n \times m$, заполнив её "змейкой" в соответствии с образцом.

Формат входных данных

На вход программе на одной строке подаются два натуральных числа

n

и

m

— количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести указанную матрицу в соответствии с образцом.

Примечание. Для вывода элементов матрицы как в примерах отводите ровно 3 символа на каждый элемент. Для этого используйте строковый метод ljust(). Можно обойтись и без ljust(), система примет и такое решение 😊

Тестовые данные

```
d=[]
s=[]
n,m=input().split()
n,m=int(n),int(m)
x,a=0,0
for i in range(n):
    d.append([0]*m)

for i in range (n):
    for j in range(m):
        a=a+1
        d[i][j]=str(a)
for i in d:
    x=x+1
    if x % 2 != 0:
        s.append(i[::-1])
    else:
        s.append(i[:])

for i in range(n):
    for j in range(m):
        print(s[i][j].ljust(3),end=" ")
    print()
```

Заполнение диагоналями

На вход программе подаются два натуральных числа

n и

m .

Напишите программу, которая создает матрицу размером

$n \times m$

и

заполняет её "диагоналями" в соответствии с образцом.

Формат входных данных

На вход программе на одной строке подаются два натуральных числа

n и

m .

m – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести указанную матрицу в соответствии с образцом.

Примечание. Для вывода элементов матрицы как в примерах отводите ровно

3

3 символа на каждый элемент. Для этого используйте строковый метод ljust(). Можно обойтись и без ljust(), система примет и такое решение 😊

```
n,m=input().split()
n=int(n)
m=int(m)
w=1
d=[]
for i in range(n):
    d.append([0]*m)
for s in range(n*m):
    for i in range(n):
        for j in range(m):
            if i + j ==s:
                d[i][j]=w
                w=w+1
for i in d:
    print(*i)

n, m = [int(el) for el in input().split()]
matrix = [[None for _ in range(m)] for _ in range(n)]
cnt = 1

# проходим по всем диагоналям
for d in range(n + m - 1):
    for i in range(n):
        for j in range(m):
            if i + j == d:
                matrix[i][j] = cnt
                cnt += 1

for i in range(n):
    for j in range(m):
        print(str(matrix[i][j]).ljust(3), end="")
    print()
```

Заполнение спиралью 🐱🐱

На вход программе подаются два натуральных числа

🕒

n и

🕒

m. Напишите программу, которая создает матрицу размером

0

x

0

$n \times m$, заполнив её "спиралью" в соответствии с образцом.

Формат входных данных

На вход программе на одной строке подаются два натуральных числа

0

n и

0

m – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести матрицу в соответствии образцом.

Примечание. Для вывода элементов матрицы как в примерах отводите ровно 3

3 символа на каждый элемент. Для этого используйте строковый метод ljust(). Можно обойтись и без ljust(), система примет и такое решение ☺

Тема урока: операции над матрицами Сложение матриц Умножение матрицы на число Умножение матриц Аннотация. Урок посвящен основным операциям над матрицами в математике.

Сложение матриц При сложении матриц A и B получается такая матрица C , каждый элемент которой представляет собой сумму пары соответствующих элементов исходных матриц A и B . Складывать можно только матрицы одинаковой размерности ($n \times m$), с равным количеством строк и столбцов. Таким образом, математически сумма матриц выглядит так:

<https://stepik.org/lesson/416756/step/1?unit=406264>

Сложение матриц

Напишите программу для вычисления суммы двух матриц.

Формат входных данных

На вход программе подаются два натуральных числа

0

n и

0

m – количество строк и столбцов в матрицах, затем элементы первой матрицы, затем пустая строка, далее следуют элементы второй матрицы.

Формат выходных данных

Программа должна вывести результирующую матрицу, разделяя элементы символом пробела.

Умножение матриц

Напишите программу, которая перемножает две матрицы.

Формат входных данных

На вход программе подаются два натуральных числа

и

и

и

m — количество строк и столбцов в первой матрице, затем элементы первой матрицы, затем пустая строка. Далее следуют числа

и

и

и

k — количество строк и столбцов второй матрицы затем элементы второй матрицы.

Формат выходных данных

Программа должна вывести результирующую матрицу, разделяя элементы символом пробела.

Возведение матрицы в степень Напишите программу, которая возводит квадратную матрицу в m -ую степень.

Формат входных данных На вход программе подаётся натуральное число n — количество строк и столбцов в матрице, затем элементы матрицы, затем натуральное число

m .

Формат выходных данных Программа должна вывести результирующую матрицу, разделяя элементы символом пробела.

Каждый n -ый элемент

На вход программе подается строка текста, содержащая символы и число

и

n . Из данной строки формируется список. Напишите программу, которая разделяет список на вложенные подписки так, что

и

n последовательных элементов принадлежат разным подпискам.

Формат входных данных

На вход программе подается строка текста, содержащая символы, отделенные символом пробела и число

и

n на отдельной строке.

Формат выходных данных

Программа должна вывести указанный вложенный список.

Примечание 1. Графическая иллюстрация для

```

1
1 теста:

# put your python code here
rez=[]
d=input().split()
n=int(input())
for j in range(n):
    r=[]
    for i in range(j,len(d),n):
        r.append(d[i])
    rez.append(r)
print(rez)

```

Максимальный в области 2

Напишите программу, которая выводит максимальный элемент в заштрихованной области квадратной матрицы.

Формат входных данных

На вход программе подаётся натуральное число

0

n — количество строк и столбцов в матрице, затем элементы матрицы.

Формат выходных данных

Программа должна вывести одно число — максимальный элемент в заштрихованной области квадратной матрицы.

Примечание. Элементы побочной диагонали также учитываются.

Тестовые данные

1.png

```

n=int(input())
rez=[]
d=[list(map(int,input().split())) for i in range(n)]
maks=d[0][0]
for i in range(n):
    for j in range(n):
        if i>= n-1-j and d[i][j] > maks:
            maks=d[i][j]
print(maks)

```

Транспонирование матрицы

Напишите программу, которая транспонирует квадратную матрицу.

Формат входных данных

На вход программе подаётся натуральное число



n – количество строк и столбцов в матрице, затем элементы матрицы.

Формат выходных данных

Программа должна вывести транспонированную матрицу.

Примечание 1. Транспонированная матрица – матрица, полученная из исходной матрицы заменой строк на столбцы.

Примечание 2. Задачу можно решить без использования вспомогательного списка.

put your python code here

```
r=[]
rez=[]
n=int(input())
d=[list(map(int,input().split())) for i in range(n)]
for i in range(n):
    r=[]
    for j in range(n):
        r.append(d[j][i])
    rez.append(r)
for i in rez:
    print(*i)
```

Снежинка

На вход программе подается нечетное натуральное число



n . Напишите программу, которая создает матрицу размером



\times



$n \times n$ заполнив её символами `.` . Затем заполните символами `*` среднюю строку и столбец матрицы, главную и побочную диагональ матрицы. Выведите полученную матрицу на экран, разделяя элементы пробелами.

Формат входных данных

На вход программе подается нечетное натуральное число



,

(



\geq

3

)

$n, (n \geq 3)$ – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести матрицу в соответствии с условием задачи.

```

* . * . *
. * * * .
* * * * *
. * * * .
* . * . *

```

```

n=int(input())
sred=n//2
d=[["."]*n for i in range(n)]
for i in range(n):
    for j in range(n):
        if i==sred or j== sred or i==j or j==n-i-1:
            d[i][j]="*"
for i in d:
    print(*i)

```

```

8
* . . . * . . *
. * . . * . * .
. . * . * * . .
. . . * * . . .
* * * * * * * *
. . * . * * . .
. * . . * . * .
* . . . * . . *

```

Симметричная матрица

Напишите программу проверки симметричности квадратной матрицы относительно побочной диагонали.

Формат входных данных

На вход программе подаётся натуральное число



n — количество строк и столбцов в матрице, затем элементы матрицы.

Формат выходных данных

Программа должна вывести YES, если матрица симметрична, и слово NO в противном случае.

put your python code here

```

n=int(input())
d=[list(map(int,input().split())) for i in range(n)]
d.reverse()
rez="YES"
for i in range(n):
    for j in range(n):
        if d[i][j] != d[j][i] and i !=j:
            rez="NO"
print(rez)

```

Латинский квадрат
Латинским квадратом порядка

n называется квадратная матрица размером

n

\times

n

$n \times n$, каждая строка и каждый столбец которой содержат все числа от 1 до

n .

Напишите программу, которая проверяет, является ли заданная квадратная матрица латинским квадратом.

Формат входных данных

На вход программе подаётся натуральное число

n

– количество строк и столбцов в матрице, затем элементы матрицы:

n

строк, по

n

чисел в каждой, разделённые пробелами.

Формат выходных данных

Программа должна вывести слово YES, если матрица является латинским квадратом, и слово NO, если не является.

Тестовые данные

```
# put your python code here
n=int(input())
rez=[]
rez_=[]
s=[list(range(1,n+1)) for i in range(n)]
d=[list(map(int,input().split())) for i in range(n)]
d_=[]
#print(d)
for i in d:
    z=sorted(i)
    rez.append(z)
#print(rez)
#print(s)
for i in range(n):
    r=[]
    for j in range(n):
        r.append(d[j][i])
    d_.append(r)
for i in d_:
    z=sorted(i)
    rez_.append(z)
```

```

if rez_==rez and s==rez_:
    print("YES")
else:
    print("NO")

```

Ходы ферзя

На шахматной доске

8

x

8

8×8 стоит ферзь. Отметьте положение ферзя на доске и все клетки, которые бьет ферзь. Клетку, где стоит ферзь, отметьте буквой Q, клетки, которые бьет ферзь, отметьте символами *, остальные клетки заполните точками.

Формат входных данных

На вход программе подаются координаты ферзя на шахматной доске в шахматной нотации (то есть в виде e4, где сначала записывается номер столбца (буква от а до h, слева направо), затем номер строки (цифра от

1

1 до

8

8, снизу вверх)).

Формат выходных данных

Программа должна вывести на экран изображение доски, разделяя элементы пробелами.

```

. . * . . * .
. . * . . * .
* . * . * . .
. * * * . . .
* * Q * * * *
. * * * . . .
* . * . * . .
. . * . . * .

```

```

col_,str_=input()
col=ord(col_)-97
str=8-int(str_)
d=[]
for i in range(8):
    d.append(["."]*8)
for i in range(8):
    for j in range(8):
        d[str][col]="Q"
        if str==i or col == j:
            d[i][j]="*"
        if str+col == i+j or str-col == i-j:
            d[i][j]="*"

```

```
for i in d:
    print(*i)
```

Диагонали, параллельные главной

На вход программе подается натуральное число

0

n. Напишите программу, которая создает матрицу размером

0

x

0

n×n и заполняет её по следующему правилу:

на главной диагонали на месте каждого элемента должно стоять число

0

0;

на двух диагоналях, прилегающих к главной, – число

1

1;

на следующих двух диагоналях – число

2

2, и т.д.

Формат входных данных

На вход программе подается натуральное число

0

n – количество строк и столбцов в матрице.

Формат выходных данных

Программа должна вывести матрицу в соответствии с условием задачи.

Тестовые данные

```
n=int(input())
d=[[0]*n for i in range(n)]
for i in range(n):
    s=0
    for j in range(n):
        s=s+1
        d[i][j]=abs(i-j)
for i in d:
    print(*i)
```

Тема урока: кортежи Тип данных tuple Особенности работы с кортежами Аннотация. Урок посвящен кортежам (тип данных tuple).

Мы изучили списки и строки. Списки – изменяемые коллекции, строки – неизменяемые последовательности Unicode символов. В Python имеются и неизменяемые последовательности содержащие, в отличие от строк, абсолютно произвольные данные. Такие коллекции называются кортежами (tuple, читается "тюпл").

Кортежи Рассмотрим следующий программный код:

`my_list = [1, 2, 3, 4, 5]` Мы объявили список чисел и присвоили его переменной `my_list`. Содержимое списка можно изменять.

Следующий программный код:

```
my_list = [1, 2, 3, 4, 5] my_list[0] = 9 my_list[4] = 7 print(my_list)
```

выведет:

`[9, 2, 3, 4, 7]` Заменяв квадратные скобки при объявлении списка на круглые, мы объявляем кортеж:

`my_tuple = (1, 2, 3, 4, 5)` Кортежи по своей природе (задумке) – неизменяемые аналоги списков. Поэтому программный код:

```
my_tuple = (1, 2, 3, 4, 5) my_tuple[0] = 9 my_tuple[4] = 7 print(my_tuple)
```

приводит к ошибке

`TypeError: 'tuple' object does not support item assignment` Кортеж (tuple) – ещё один вид коллекций в Python. Они похожи на списки, но являются неизменяемыми.

В литеральной форме кортеж записывается в виде последовательности элементов в круглых скобках, а список – в квадратных.

Примеры кортежей `empty_tuple = ()` # пустой кортеж `point = (1.5, 6.0)` # кортеж из двух чисел `names = ('Timur', 'Ruslan', 'Roman')` # кортеж из трех строк `info = ('Timur', 'Guev', 28, 170, 60, False)` # кортеж из 6 элементов разных типов `nested_tuple = (('one', 'two'), ['three', 'four'])` # кортеж из кортежа и списка в переменной `empty_tuple` хранится пустой кортеж; в переменной `point` хранится кортеж, состоящий из двух вещественных чисел (такой кортеж удобно использовать для представления точки на координатной плоскости); в переменной `names` хранится кортеж, содержащий три строковых значения; в переменной `info` содержится кортеж, содержащий 6 элементов разного типа (строки, числа, булевы переменные); в переменной `nested_tuple` содержится кортеж, содержащий другой кортеж и список. Кортежи могут хранить и содержать в себе объекты любых типов (даже составных) и поддерживают неограниченное количество уровней вложенности.

Кортеж с одним элементом Для создания кортежа с единственным элементом после значения элемента ставят закрывающую запятую:

```
my_tuple = (1,) print(type(my_tuple))
```

 # <class 'tuple'> Если запятую пропустить, то кортеж создан не будет. Например, приведенный ниже код просто присваивает переменной `my_tuple` целочисленное значение 1:

```
my_tuple = (1) print(type(my_tuple))
```

 # <class 'int'> Зачем использовать кортеж вместо списка? Списки могут делать то же, что кортежи, и даже больше. Но неизменяемость кортежей обеспечивает им особые свойства:

скорость – кортежи быстрее работают, так как из-за неизменяемости хранятся в памяти иначе, и операции с их элементами выполняются заведомо быстрее, чем с компонентами списка. Одна из причин существования кортежей – производительность. Обработка кортежа выполняется быстрее, чем обработка списка, поэтому кортежи удобны для обработки большого объема неизменяемых данных. безопасность – неизменяемость превращает их в идеальные константы. Заданные кортежами константы делают код более читаемым и безопасным. Кроме того, в кортеже можно безопасно хранить данные, не опасаясь, что они будут случайно или преднамеренно изменены в программе. В Python

существуют операции, требующие применения кортежа. По мере освоения языка Python вы будете чаще сталкиваться с кортежами.

Примечания Примечание 1. Мы уже сталкивались с кортежами, когда изучали функции, возвращающие несколько значений. Такие функции возвращают именно кортежи.

Рассмотрим функцию `get_powers()`, которая принимает в качестве аргумента число и возвращает его 2 2, 3 3 и 4 4 степень.

```
def get_powers(num): return num2, num3, num**4
```

 Результатом выполнения следующего кода:

```
result = get_powers(5) print(type(result)) print(result)
```

 будет:

```
<class 'tuple'> (25, 125, 625)
```

 Примечание 2. Списки предназначены для объединения неопределенного количества однородных сущностей. Кортежи, как правило, объединяют под одним именем несколько разнородных объектов, имеющих различный смысл.

Например, список удобен для хранения нескольких городов:

```
cities = ["Perth", "San Francisco", "Lisbon", "Sochi"]
```

 # список городов А кортеж удобен для хранения данных о людях:

```
person = ("Tony", 21, "Auckland")
```

 # кортеж с данными о человеке: имя, возраст, город

Примечание 3. Тот факт, что кортеж является неизменяемым вовсе не означает, что мы не можем поменять содержимое списка в кортеже.

Приведенный ниже код:

```
my_tuple = (1, 'python', [1, 2, 3]) print(my_tuple) my_tuple[2][0] = 100 my_tuple[2].append(17) print(my_tuple)
```

 выводит:

```
(1, 'python', [1, 2, 3]) (1, 'python', [100, 2, 3, 17])
```

 При этом важно понимать: меняется список, а не кортеж. Списки являются ссылочными типами данных, поэтому в кортеже хранится ссылка на список, которая не меняется при изменении самого списка.

Тема урока: кортежи

Функция `tuple()`

Особенности кортежей

Методы кортежей

Вложенные кортежи

Аннотация. Урок посвящен кортежам (тип данных `tuple`).

Функция `tuple()`

Встроенная функция `list()` может применяться для преобразования кортежа в список.

Приведенный ниже код:

```
number_tuple = (1, 2, 3, 4, 5)
number_list = list(number_tuple)
print(number_list)
```

ВЫВОДИТ:

```
[1, 2, 3, 4, 5]
```

Встроенная функция `tuple()` может применяться для преобразования списка в кортеж.

Приведенный ниже код:

```
str_list = ['один', 'два', 'три']
str_tuple = tuple(str_list)
print(str_tuple)
```

ВЫВОДИТ:

```
('один', 'два', 'три')
```

Аналогичным образом мы можем создать кортеж на основании строки.

Приведенный ниже код:

```
text = 'hello python'
str_tuple = tuple(text)
print(str_tuple)
```

ВЫВОДИТ:

```
('h', 'e', 'l', 'l', 'o', ' ', 'p', 'y', 't', 'h', 'o', 'n')
```

Обратите внимание, что символ пробела содержится в кортеже `str_tuple`.

Преобразование строки в список позволяет получить список символов строки. Это может быть полезно, например, когда надо изменить один символ строки:

```
s = 'СИМПОТИЧНЫЙ'
print(s)
```

```
a = list(s)
a[4] = 'а'
s = ''.join(a)
print(s)
```

Приведенный выше код выводит:

```
симпотичный
```

```
симпатичный
```

С этой же целью может потребоваться преобразование кортежа в список:

```
writer = ('Лев Толстой', 1827)
print(writer)
```

```
a = list(writer)
a[1] = 1828
writer = tuple(a)
print(writer)
```

Приведенный выше код выводит:

```
('Лев Толстой', 1827)
```

```
('Лев Толстой', 1828)
```

Особенности кортежей

Кортежи поддерживают те же операции, что и списки, за исключением изменяющих содержимое.

Кортежи поддерживают:

доступ к элементу по индексу (только для получения значений элементов);

методы, в частности `index()`, `count()`;

встроенные функции, в частности `len()`, `sum()`, `min()` и `max()`;

срезы;

оператор принадлежности `in`;

операторы конкатенации (+) и повторения (*).

Функция `len()`

Длиной кортежа называется количество его элементов. Для того, чтобы посчитать длину кортежа, мы используем встроенную функцию `len()`.

Следующий программный код:

```
numbers = (2, 4, 6, 8, 10)
```

```
languages = ('Python', 'C#', 'C++', 'Java')
```

```
print(len(numbers))      # выводим длину кортежа numbers
```

```
print(len(languages))    # выводим длину кортежа languages
```

```
print(len(('apple', 'banana', 'cherry'))) # выводим длину кортежа, состоящего из 3 элементов
```

выведет:

```
5
```

```
4
```

```
3
```

Оператор принадлежности `in`

Оператор `in` позволяет проверить, содержит ли кортеж некоторый элемент.

Рассмотрим следующий код:

```
numbers = (2, 4, 6, 8, 10)
```

```
if 2 in numbers:
```

```
    print('Кортеж numbers содержит число 2')
```

```
else:
```

```
    print('Кортеж numbers не содержит число 2')
```

Такой код проверяет, содержит ли кортеж `numbers` число

```
2
```

и выводит соответствующий текст:

Кортеж `numbers` содержит число `2`
Мы можем использовать оператор `in` вместе с логическим оператором `not`.
Например

```
numbers = (2, 4, 6, 8, 10)

if 0 not in numbers:
    print('Кортеж numbers не содержит нулей')
```

Индексация

При работе со списками (строками) мы использовали индексацию, то есть обращение к конкретному элементу списка (строки) по его индексу. Аналогично можно индексировать и элементы кортежей.

Для индексации кортежей в Python используются квадратные скобки `[]`, в которых указывается индекс (номер) нужного элемента в кортеже:

Пусть `numbers = (2, 4, 6, 8, 10)`.

Таблица ниже показывает, как работает индексация:

Выражение	Результат	Пояснение
-----------	-----------	-----------

<code>numbers[0]</code>		
-------------------------	--	--

<code>2</code>		
----------------	--	--

<code>2</code>	первый элемент кортежа	
----------------	------------------------	--

<code>numbers[1]</code>		
-------------------------	--	--

<code>4</code>		
----------------	--	--

<code>4</code>	второй элемент кортежа	
----------------	------------------------	--

<code>numbers[2]</code>		
-------------------------	--	--

<code>6</code>		
----------------	--	--

<code>6</code>	третий элемент кортежа	
----------------	------------------------	--

<code>numbers[3]</code>		
-------------------------	--	--

<code>8</code>		
----------------	--	--

<code>8</code>	четвертый элемент кортежа	
----------------	---------------------------	--

<code>numbers[4]</code>		
-------------------------	--	--

<code>10</code>		
-----------------	--	--

<code>10</code>	пятый элемент кортежа	
-----------------	-----------------------	--

Так же, как и в списках, для нумерации с конца разрешены отрицательные индексы:

Выражение	Результат	Пояснение
-----------	-----------	-----------

<code>numbers[-1]</code>		
--------------------------	--	--

<code>10</code>		
-----------------	--	--

<code>10</code>	пятый элемент кортежа	
-----------------	-----------------------	--

<code>numbers[-2]</code>		
--------------------------	--	--

<code>8</code>		
----------------	--	--

<code>8</code>	четвертый элемент кортежа	
----------------	---------------------------	--

<code>numbers[-3]</code>		
--------------------------	--	--

<code>6</code>		
----------------	--	--

<code>6</code>	третий элемент кортежа	
----------------	------------------------	--

```
numbers[-4]
```

```
4
```

4 второй элемент кортежа

```
numbers[-5]
```

```
2
```

2 первый элемент кортежа

Как и в списках, попытка обратиться к элементу кортежа по несуществующему индексу:

```
print(numbers[17])
```

вызовет ошибку:

IndexError: tuple index out of range

Срезы

Рассмотрим кортеж `numbers = (2, 4, 6, 8, 10)`.

С помощью среза мы можем получить несколько элементов кортежа, создав диапазон индексов, разделенных двоеточием `numbers[x:y]`.

Следующий программный код:

```
print(numbers[1:3])
```

```
print(numbers[2:5])
```

выводит:

```
(4, 6)
```

```
(6, 8, 10)
```

При построении среза `numbers[x:y]` первое число – это то место, где начинается срез (включительно), а второе – это место, где заканчивается срез (невключительно).

При использовании срезов с кортежами мы также можем опускать второй параметр в срезе `numbers[x:]` (но поставить двоеточие), тогда срез берется до конца кортежа. Аналогично, если опустить первый параметр `numbers[:y]`, то можно взять срез от начала кортежа.

Срез `numbers[:]` возвращает копию исходного кортежа.

Как и в списках, можно использовать отрицательные индексы в срезах кортежей.

Операция конкатенации + и умножения на число *

Операторы + и * применяют для кортежей, как и для списков.

Следующий программный код:

```
print((1, 2, 3, 4) + (5, 6, 7, 8))
```

```
print((7, 8) * 3)
```

```
print((0,) * 10)
```

выводит:

```
(1, 2, 3, 4, 5, 6, 7, 8)
(7, 8, 7, 8, 7, 8)
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Для генерации кортежей, состоящих строго из повторяющихся элементов, умножение на число – самый короткий и правильный путь.

Расширенные операторы `+=` и `*=` также можно использовать при работе с кортежами.

Следующий программный код:

```
a = (1, 2, 3, 4)
b = (7, 8)
a += b    # добавляем к кортежу a кортеж b
b *= 5    # повторяем кортеж b 5 раз
```

```
print(a)
print(b)
выводит:
```

```
(1, 2, 3, 4, 7, 8)
(7, 8, 7, 8, 7, 8, 7, 8)
```

Встроенные функции `sum()`, `min()`, `max()`

Встроенная функция `sum()` принимает в качестве параметра кортеж чисел и вычисляет сумму его элементов.

Следующий программный код:

```
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print('Сумма всех элементов кортежа =', sum(numbers))
выводит:
```

Сумма всех элементов кортежа = 55

Встроенные функции `min()` и `max()` принимают в качестве параметра кортеж и находят минимальный и максимальный элементы соответственно.

Следующий программный код:

```
numbers = (3, 4, 10, 3333, 12, -7, -5, 4)
print('Минимальный элемент кортежа =', min(numbers))
print('Максимальный элемент кортежа =', max(numbers))
выводит:
```

Минимальный элемент кортежа = -7

Максимальный элемент кортежа = 3333

Функции `min()` и `max()` можно применять только к кортежам с одним типом данных. Если кортеж содержит разные типы данных, скажем, целое число (`int`) и строку (`str`), то во время выполнения программы произойдет ошибка.

Метод `index()`

Метод `index()` возвращает индекс первого элемента, значение которого равняется переданному в метод значению. Таким образом, в метод передается один параметр:

`value`: значение, индекс которого требуется найти.

Если элемент в кортеже не найден, то во время выполнения происходит ошибка.

Следующий программный код:

```
names = ('Gvido', 'Roman' , 'Timur')
position = names.index('Timur')
```

```
print(position)
```

выведет:

2

Следующий программный код:

```
names = ('Gvido', 'Roman' , 'Timur')
position = names.index('Anders')
```

```
print(position)
```

приводит к ошибке:

`ValueError: tuple.index(x): x not in tuple`

Чтобы избежать таких ошибок, можно использовать метод `index()` вместе с оператором принадлежности `in`:

```
names = ('Gvido', 'Roman', 'Timur')
```

```
if 'Anders' in names:
    position = names.index('Anders')
    print(position)
```

```
else:
```

```
    print('Такого значения нет в кортеже')
```

Метод `count()`

Метод `count()` возвращает количество элементов в кортеже, значения которых равны переданному в метод значению.

Таким образом, в метод передается один параметр:

`value`: значение, количество вхождений которого нужно посчитать.

Если значение в кортеже не найдено, то метод возвращает

0

0.

Следующий программный код:

```
names = ('Timur', 'Gvido', 'Roman', 'Timur', 'Anders', 'Timur')
```

```
cnt1 = names.count('Timur')
cnt2 = names.count('Gvido')
cnt3 = names.count('Josef')
```

```
print(cnt1)
print(cnt2)
print(cnt3)
выведет:
```

```
3
1
0
```

Кортежи не поддерживают такие методы, как `append()`, `remove()`, `pop()`, `insert()`, `reverse()`, `sort()`, так как эти методы изменяют содержимое.

Вложенные кортежи

Подобно спискам, мы можем создавать вложенные кортежи.

Следующий программный код:

```
colors = ('red', ('green', 'blue'), 'yellow')
numbers = (1, 2, (4, (6, 7, 8, 9)), 10, 11)
```

```
print(colors[1][1])
print(numbers[2][1][3])
выводит:
```

```
blue
9
```

Тема урока: кортежи

Перебор кортежей

Сравнение кортежей

Сортировка кортежей

Преобразование кортежа в список и строку

Упаковка кортежей

Распаковка кортежей

Присваивание кортежей

Аннотация. Урок посвящен кортежам (тип данных `tuple`).

Перебор кортежей

Перебор элементов кортежа осуществляется точно так же как перебор элементов списка.

Для вывода каждого из элементов кортежа на отдельной строке можно использовать следующий код:

Вариант 1. Если нужны индексы элементов:

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```



```
for i in range(len(numbers)):
    print(numbers[i])
```

Вариант 2. Если индексы не нужны:

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
for num in numbers:
    print(num)
```

Можно также использовать операцию распаковки кортежа.

Приведенный ниже код:

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
languages = ('Python', 'C++', 'Java')
```

```
print(*numbers)
print(*languages, sep='\n')
```

Выводит:

```
0 1 2 3 4 5 6 7 8 9 10
```

```
Python
```

```
C++
```

```
Java
```

Сравнение кортежей

Кортежи можно сравнивать между собой.

Приведенный ниже код:

```
print((1, 8) == (1, 8))
print((1, 8) != (1, 10))
print((1, 9) < (1, 2))
print((2, 5) < (6,))
print(('a', 'bc') > ('a', 'de'))
```

Выводит:

```
True
```

```
True
```

```
False
```

```
True
```

```
False
```

Обратите внимание: операции == и != применимы к любым кортежам, независимо от типов элементов. А вот операции <, >, <=, >= применимы только в том случае, когда соответствующие элементы кортежей имеют один тип.

Приведенный ниже код:

```
print((7, 5) < ('java', 'python'))
```

Выводит:

`TypeError: '<' not supported between instances of 'int' and 'str'`

Сравнение кортежей происходит последовательно элемент за элементом, а если элементы равны – просматривается следующий элемент.

Сортировка кортежей

Как мы помним, списки имеют метод `sort()`, который осуществляет сортировку на месте, то есть меняет порядок исходного списка. Поскольку кортежи по своей природе неизменяемы, то встроенного метода `sort()` они не содержат, тем не менее с помощью встроенной функции `sorted()` (не путать с списочным методом `sort()`) мы можем сортировать значения в кортежах.

Приведенный ниже код:

```
not_sorted_tuple = (34, 1, 8, 67, 5, 9, 0, 23)
print(not_sorted_tuple)
```

```
sorted_tuple = tuple(sorted(not_sorted_tuple))
print(sorted_tuple)
```

выводит:

```
(34, 1, 8, 67, 5, 9, 0, 23)
(0, 1, 5, 8, 9, 23, 34, 67)
```

Обратите внимание, что функция `sorted()` возвращает список, но с помощью функции `tuple()` мы приводим результат сортировки к кортежу.

Для сортировки кортежа можно воспользоваться явным преобразованием в список и использовать метод `sort()`:

```
not_sorted_tuple = ('cc', 'aa', 'dd', 'bb')
tmp = list(not_sorted_tuple)
tmp.sort()
```

```
sorted_tuple = tuple(tmp)
print(sorted_tuple)
```

Преобразование кортежа в список и строку

Часто на практике нам приходится преобразовывать кортежи в списки и в строки. Для этого используются функции и методы `str()`, `list()`, `tuple()`, `join()`.

Преобразование кортежа в список и наоборот

Кортеж можно преобразовать в список с помощью функции `list()`.

Приведенный ниже код:

```
tuple1 = (1, 2, 3, 4, 5)
list1 = list(tuple1)
print(list1)
```

выводит:

```
[1, 2, 3, 4, 5]
```

Список можно преобразовать в кортеж с помощью функции `tuple()`.

Приведенный ниже код:

```
list1 = [1, 17.8, 'Python']  
tuple1 = tuple(list1)  
print(tuple1)
```

выводит:

```
(1, 17.8, 'Python')
```

Преобразование кортежа в строку и наоборот

Кортеж можно преобразовать в строку с помощью строкового метода `join()`.

Приведенный ниже код:

```
notes = ('Do', 'Re', 'Mi', 'Fa', 'Sol', 'La', 'Si')  
string1 = ''.join(notes)  
string2 = '.'.join(notes)
```

```
print(string1)
```

```
print(string2)
```

выводит:

```
DoReMiFaSolLaSi
```

```
Do.Re.Mi.Fa.Sol.La.Si
```

Обратите внимание, что для применения строкового метода `join()` кортеж должен содержать именно строковые элементы. Если элементы кортежа отличны от строк, то требуется предварительно их преобразовать.

Строку можно преобразовать в кортеж с помощью функции `tuple()`.

Приведенный ниже код:

```
letters = 'abcdefghijkl'  
tpl = tuple(letters)  
print(tpl)
```

выводит:

```
('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l')
```

Обратите внимание, что следующий код:

```
number = 12345  
tpl = tuple(number)  
print(tpl)
```

приведет к ошибке:

```
TypeError: 'int' object is not iterable
```

Это происходит, поскольку тип данных `int` не является итерируемым

объектом. Для преобразования числа в кортеж сначала нужно число преобразовать в строку и уже только потом использовать функцию `tuple()`.

Множество

Тема урока: множества в Python

Создание множеств

Пустые множества

Встроенная функция `set()`

Вывод множеств

Аннотация. Начинаем изучение множеств в Python (тип данных `set`). Этот тип данных аналогичен математическим множествам, он поддерживает быстрые операции проверки наличия элемента в множестве, добавления и удаления элементов, операции объединения, пересечения и многие другие.

Множества

В прошлых уроках мы изучили три типа коллекций в Python:

списки — изменяемые коллекции элементов;

строки — неизменяемые коллекции символов;

кортежи — неизменяемые коллекции элементов.

Следующий тип коллекций (наборов данных) — множество.

Множество — структура данных, организованная так же, как математические множества.

Важно знать:

все элементы множества различны (уникальны), два элемента не могут иметь одинаковое значение;

множества неупорядочены, то есть элементы не хранятся в каком-то определенном порядке;

элементы множества должны относиться к неизменяемым типам данных;

хранящиеся в множестве элементы могут иметь разные типы данных.

Структура данных (data structure) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных.

Создание множества

Чтобы создать множество, нужно перечислить его элементы через запятую в фигурных скобках:

```
numbers = {2, 4, 6, 8, 10}
languages = {"Python", "C#", "C++", "Java"}
```

Множество `numbers` состоит из

5

5 элементов, и каждый из них — целое число.

Множество `languages` состоит из

4

4 элементов, каждый из которых — строка.

Множества могут содержать значения разных типов данных:

```
info = {'Timur', 1992, 61.5}
```

Множество `info` содержит строковое значение, целое число и число с плавающей точкой.

Не создавайте переменные с именем `set`. Это очень плохая практика.

Пустое множество

Создать пустое множество можно с помощью встроенной функции, которая называется `set()`:

```
myset = set() # пустое множество
```

Обратите внимание — создать пустое множество с помощью пустых фигурных скобок нельзя:

```
myset = {} # создается словарь
```

С помощью пустых фигурных скобок создаются словари: так сложилось исторически. Дело в том, что словари появились в Python раньше, чем множества.

Пустое множество создаётся исключительно через `set()`.

Вывод множества

Для вывода всего множества можно использовать функцию `print()`:

```
numbers = {2, 4, 6, 8, 10}
languages = {"Python", "C#", "C++", "Java"}
mammals = {"cat", "dog", "fox", "elephant"}
```

```
print(numbers)
print(languages)
print(mammals)
```

Функция `print()` выводит на экран элементы множества в фигурных скобках, разделенные запятыми:

```
{2, 4, 6, 8, 10}
{"C#", "Python", "Java", "C++"}
{"dog", "cat", "fox", "elephant"}
```

Обратите внимание: при выводе множества порядок элементов может отличаться от существовавшего при его создании, поскольку множества — неупорядоченные коллекции данных.

Встроенная функция `set()`

Встроенная функция `set()` помимо создания пустого множества может преобразовывать некоторые типы объектов в множества.

В функцию `set()` можно передать один аргумент. Передаваемый аргумент должен быть итерируемым объектом, таким как список, кортеж или строковое значение. Отдельные элементы объекта, передаваемого в качестве аргумента, становятся элементами множества:

```
myset1 = set(range(10))           # множество из элементов
                                  # последовательности
myset2 = set([1, 2, 3, 4, 5])     # множество из элементов списка
myset3 = set('abcd')             # множество из элементов строки
myset4 = set((10, 20, 30, 40))    # множество из элементов кортежа
```

Пустое множество также можно создать передав функции `set()` в качестве аргумента пустой список, строку или кортеж:

```
emptyset1 = set([])              # пустое множество из пустого списка
emptyset2 = set('')              # пустое множество из пустой строки
emptyset3 = set(())              # пустое множество из пустого кортежа
```

Дубликаты при создании множеств

Множества не могут содержать повторяющиеся элементы. Если в функцию `set()` передать аргумент, содержащий повторяющиеся элементы, то в множестве появится только один из этих повторяющихся элементов.

Приведенный ниже код:

```
myset1 = {2, 2, 4, 6, 6}
myset2 = set([1, 2, 2, 3, 3])
myset3 = set("aaaaabbbbccccddd")
```

```
print(myset1)
print(myset2)
print(myset3)
```

выводит (порядок элементов может отличаться):

```
{2, 4, 6}
{1, 2, 3}
{"b", "c", "d", "a"}
```

Если требуется создать множество, в котором каждый элемент — строковое значение, содержащее более одного символа, то используем код:

```
myset = set(['aaa', 'bbbb', 'cc'])
```

```
print(myset)
```

Приведенный выше код выводит (порядок элементов может отличаться):

```
{'bbbb', 'aaa', 'cc'}
```

Если же создать множество следующим образом:

```
myset = set('aaa bbbb cc')
```

```
print(myset)
```

то мы получим (порядок элементов может отличаться):

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

Обратите внимание на наличие пробела в качестве элемента множества myset.

Примечания

Примечание 1. Элементы множества могут принадлежать любому неизменяемому типу данных: быть числами, строками, кортежами и т.д. Элементы изменяемых типов данных не могут входить в множества, в частности, нельзя сделать элементом множества список или другое множество. Требование неизменяемости элементов множества накладывается особенностями представления множеств в Python.

Приведенный ниже код:

```
myset1 = {1, 2, [5, 6], 7} # множество не может содержать список
myset2 = {1, 2, {5, 6}, 7} # множество не может содержать множество
приводит к ошибке:
```

```
TypeError: unhashable type: 'list'
```

```
TypeError: unhashable type: 'set'
```

Однако приведенный ниже код:

```
myset = {1, 2, (5, 6), 7} # множество может содержать кортеж
работает, как полагается.
```

Примечание 2. Документация по множествам доступна по ссылке.

Примечание 3. Отличная статья с хабра про множества.

Тема урока: множества в Python

Встроенные функции `len()`, `sum()`, `min()`, `max()`

Оператор принадлежности `in`

Перебор множеств

Форматированный вывод множеств

Сравнение множеств

Функция `sorted()`

Аннотация. В этом уроке мы изучим основной функционал при работе с множествами.

Основы работы с множествами

Работа с множествами очень сильно напоминает работу со списками, поскольку и множества, и списки содержат отдельные элементы, хотя элементы множества уникальны, а списки могут содержать повторяющиеся элементы. Многие из того, что мы делали со списками, доступно и при работе со множествами.

Функция `len()`

Длиной множества называется количество его элементов. Чтобы посчитать длину множества, используют встроенную функцию `len()` (от слова `length` – длина).

Следующий программный код:

```
myset1 = {2, 2, 4, 6, 6}
myset2 = set([1, 2, 2, 3, 3, 4, 4, 5, 5])
myset3 = set('aaaaabbbbccccddd')
```

```
print(len(myset1))
print(len(myset2))
print(len(myset3))
```

выведет:

```
3
5
4
```

Оператор принадлежности `in`

Оператор `in` позволяет проверить, содержит ли множество некоторый элемент.

Рассмотрим следующий код:

```
numbers = {2, 4, 6, 8, 10}

if 2 in numbers:
    print('Множество numbers содержит число 2')
else:
    print('Множество numbers не содержит число 2')
```

Такой код проверяет, содержит ли множество `numbers` число

```
2
```

`2` и выводит соответствующий текст:

Множество `numbers` содержит число `2`

Мы можем использовать оператор `in` вместе с логическим оператором `not`.

Например

```
numbers = {2, 4, 6, 8, 10}

if 0 not in numbers:
    print('Множество numbers не содержит нулей')
```

Оператор принадлежности `in` работает очень быстро на множествах – намного быстрее, чем на списках. Поэтому если требуется часто осуществлять поиск в коллекции уникальных данных, то множество – подходящий выбор.

Встроенные функции `sum()`, `min()`, `max()`

Встроенная функция `sum()` принимает в качестве аргумента множество

чисел и вычисляет сумму его элементов.

Следующий программный код:

```
numbers = {2, 2, 4, 6, 6}
print('Сумма всех элементов множества =', sum(numbers))
```

выводит:

Сумма всех элементов множества = 12
Встроенные функции `min()` и `max()` принимают в качестве аргумента множество и находят минимальный и максимальный элементы соответственно.

Следующий программный код:

```
numbers = {2, 2, 4, 6, 6}
print('Минимальный элемент =', min(numbers))
print('Максимальный элемент =', max(numbers))
```

выводит:

Минимальный элемент = 2
Максимальный элемент = 6

Примечания

Примечание 1. Индексация и срезы недоступны для множеств.

Примечание 2. Операция конкатенации `+` и умножения на число `*` недоступны для множеств.

Перебор элементов множества

Перебор элементов множества осуществляется точно так же, как и перебор элементов списка, то есть с помощью цикла `for`.

Для вывода элементов множества каждого на отдельной строке можно использовать следующий код:

```
numbers = {0, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7}
```

```
for num in numbers:
    print(num)
```

Такой код выведет (порядок элементов может отличаться):

```
0
1
2
3
5
6
7
```

Мы также можем использовать операцию распаковки множества.

Приведенный ниже код:

```
numbers = {0, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7}
```

```
print(*numbers, sep='\n')
```

выводит (порядок элементов может отличаться):

```
0
1
2
3
5
6
7
```

Не стоит забывать, что множества – неупорядоченные коллекции, поэтому полагаться на порядок вывода элементов не стоит. Если нужно гарантировать порядок вывода элементов (по возрастанию/убыванию), то необходимо воспользоваться встроенной функцией `sorted()`.

Приведенный ниже код:

```
numbers = {0, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7}
```

```
sorted_numbers = sorted(numbers)
```

```
print(*sorted_numbers, sep='\n')
```

будет гарантированно выводить элементы множества в порядке возрастания.

Обратите внимание на то, что функция `sorted()` возвращает отсортированный список, а не множество. Не путайте встроенную функцию `sorted()` и списочный метод `sort()`. Множества не содержат метода `sort()`.

Сравнение множеств

Множества можно сравнивать между собой. Равные множества имеют одинаковую длину и содержат равные элементы. Для сравнения множеств используются операторы `==` и `!=`.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 3, 3, 3}
```

```
myset2 = {2, 1, 3}
```

```
myset3 = {1, 2, 3, 4}
```

```
print(myset1 == myset2)
```

```
print(myset1 == myset3)
```

```
print(myset1 != myset3)
```

выводит:

```
True
False
```

True

Примечания

Примечание 1. Встроенная функция `sorted()` имеет опциональный параметр `reverse`. Если установить этот параметр в значение `True`, произойдет сортировка по убыванию.

Приведенный ниже код:

```
numbers = {0, 1, 1, 2, 3, 3, 3, 5, 6, 7, 7}
```

```
sortnumbers = sorted(numbers, reverse=True)
```

```
print(*sortnumbers, sep='\n')
```

гарантированно выводит:

```
7
6
5
3
2
1
0
```

Примечание 2. Код для работы с множествами нужно писать так, чтобы результат его выполнения не зависел от расположения элементов и был одинаковым при любом порядке обхода, последовательного обращения ко всем элементам.

Тема урока: методы множеств

Метод добавления элемента `add()`

Методы удаления элементов `remove()`, `discard()`, `pop()`

Метод удаления всех элементов `clear()`

Аннотация. Урок посвящен методам добавления и удаления элементов множеств.

Добавление элементов

Мы научились создавать множества, элементы которых известны на этапе создания. Следующий шаг – научиться добавлять элементы в уже существующие множества.

Метод `add()`

Для добавления нового элемента в множество используется метод `add()`.

Следующий программный код:

```
numbers = {1, 1, 2, 3, 5, 8, 3} # создаем множество
```

```
numbers.add(21) # добавляем число 21 в множество
```

```
numbers.add(34) # добавляем число 34 в множество
```

```
print(numbers)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 3, 34, 5, 8, 21}
```

Не забывайте, что порядок элементов при выводе множества абсолютно произвольный.

Обратите внимание, для использования метода `add()` требуется предварительно созданное множество, при этом оно может быть пустым.

Следующий программный код:

```
numbers = set() # создаем пустое множество
```

```
numbers.add(1)
numbers.add(2)
numbers.add(3)
numbers.add(1)
```

```
print(numbers)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 3}
```

Если требуется внести несколько значений в множество, то можно воспользоваться циклом `for`.

Следующий программный код:

```
numbers = set() # создаем пустое множество
```

```
for i in range(10):
    numbers.add(i * i + 1)
```

```
print(numbers)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 65, 5, 37, 10, 17, 50, 82, 26}
```

Удаление элемента

Для удаления элементов из множества используются методы:

```
remove();
discard();
pop().
```

Метод `remove()`

Метод `remove()` удаляет элемент из множества с генерацией исключения (ошибки) в случае, если такого элемента нет.

Следующий программный код:

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

```
numbers.remove(3)
print(numbers)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 4, 5}
```

Следующий программный код:

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

```
numbers.remove(10)
```

```
print(numbers)
```

приводит к возникновению ошибки `KeyError`, так как элемент

```
10
```

`10` отсутствует в множестве.

Метод `discard()`

Метод `discard()` удаляет элемент из множества без генерации исключения (ошибки), если элемент отсутствует.

Следующий программный код:

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

```
numbers.discard(3)
```

```
print(numbers)
```

выводит (порядок элементов может отличаться):

```
{1, 2, 4, 5}
```

Следующий программный код:

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

```
numbers.discard(10)
```

```
print(numbers)
```

не приводит к возникновению ошибки и выводит (порядок элементов может отличаться):

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

Метод `pop()`

Метод `pop()` удаляет и возвращает случайный элемент из множества с генерацией исключения (ошибки) при попытке удаления из пустого множества.

Рассмотрим программный код:

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

```
print('до удаления:', numbers)
```

```
num = numbers.pop()
```

множества, возвращая его

удаляет случайный элемент

```
print('удалённый элемент:', num)
```

```
print('после удаления:', numbers)
```

Результат работы такого кода случаен, например, такой код может вывести:

до удаления: {1, 2, 3, 4, 5}

удалённый элемент: 1

после удаления: {2, 3, 4, 5}

Метод pop() можно воспринимать как неконтролируемый способ удаления элементов по одному из множества.

Метод clear()

Метод clear() удаляет все элементы из множества.

Следующий программный код:

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

```
numbers.clear()
```

```
print(numbers)
```

выведет:

```
set()
```

В результате получили пустое множество.

Обратите внимание на то, что пустое множество выводится как set(), а не как {}. С помощью {} выводится пустой словарь.

Примечания

Примечание 1. Если мы не изменяли множество, порядок обхода элементов при помощи цикла for не изменится.

Примечание 2. После изменения множества (методы add(), remove(), и т.д.) порядок элементов может измениться произвольным

```
s={}
```

```
print(type(s))
```

```
<class 'dict'>
```

```
myset = set(['ъъъ', 'эээ', 'ююю', 'яяя'])
```

```
print(myset)
```

```
{'ъъъ', 'яяя', 'ююю', 'эээ'}
```

```
myset = set([1, 2, 2222, "asaaaa", 4, "asaaaa", 4, 4])
```

```
print(len(myset))
```

```
print(myset)
```

```
5
```

```
{1, 2, 4, 2222, 'asaaaa'}
```

```
myset = set('1, 2, 2222, "asaaaa", 4,"asaaaa", 4, 4')
print(myset)
```

```
{'a', ',', '1', 's', '4', '2', ' ', ' '}
```

```
myset = set("'ььь', 'эээ', 'ююю', 'яяя'")
print(myset)
```

```
{'я', ',', 'ь', 'ю', '"', 'э', ' '}
```

```
myset1 = set([1, 2, 3, 4, 5])
myset2 = set("12345")
print(myset1, myset2)
```

```
{1, 2, 3, 4, 5} {'3', '1', '4', '2', '5'}
```

```
s=set()
s1={i for i in range(9)}
s.add(4)
s1.remove(7)
# Error s1.remove(10)
s1.discard(10)
a=s1.pop()
w=s1.copy()
print("a",a)
w.clear()
print(s,s1,w)
```

```
a 0
{4} {1, 2, 3, 4, 5, 6, 8} set()
```

Уникальные символы 1

Напишите программу для вывода количества уникальных символов каждого считанного слова без учета регистра.

Формат входных данных

На вход программе в первой строке подается число

🔊

n – общее количество слов. Далее идут

🔊

n строк с словами.

Формат выходных данных

Программа должна вывести на отдельной строке количество уникальных символов для каждого слова.

```
n=int(input())
s=[list(input()) for i in range(n)]
for i in s:
    if len(i) == len(set(i)):
```

```

    print("YES")
else:
    print("NO")

```

```

3
asd
aqwew
12bnb2
YES
NO
NO

```

Уникальные символы 2 Напишите программу для вывода общего количества уникальных символов во всех считанных словах без учета регистра.

Формат входных данных На вход программе в первой строке подается число n – общее количество слов. Далее идут n строк со словами.

Формат выходных данных Программа должна вывести одно число – общее количество уникальных символов во всех словах без учета регистра.

```

rez=set()
n=int(input())
d=[list(input().lower()) for i in range(n)]
for i in d:
    rez.update(set(i))
print(len(rez))

```

```

3
55
aAs
BbaA
4
d
[['5', '5'], ['a', 'a', 's'], ['b', 'b', 'a', 'a']]
rez
{'5', 'a', 'b', 's'}

```

Количество слов в тексте Напишите программу для определения общего количества различных слов в строке текста.

Формат входных данных На вход программе подается строка текста.

Формат выходных данных Программа должна вывести одно число – общее количество различных слов в строке без учета регистра.

Примечание 1. Словом считается последовательность непробельных символов, идущих подряд, слова разделены одним или большим числом пробелов.

Примечание 2. Знаками препинания .,:;-?! пренебрегаем.

Количество слов в тексте Напишите программу для определения общего количества различных слов в строке текста.

Формат входных данных На вход программе подается строка текста.

Формат выходных данных Программа должна вывести одно число – общее количество различных слов в строке без учета регистра.

Примечание 1. Словом считается последовательность непробельных символов, идущих подряд, слова разделены одним или большим числом пробелов.

Примечание 2. Знаками препинания .,:;-?! пренебрегаем.

```
text_="Milk is white and so is glue, Ghosts are white and they say B00!".lower()
znaki="!@#$$%^&*()_,+<>?/"
for i in text_:
    if i in znaki:
        text_=text_.replace(i,"")
        text_set=set(text_.split())
        d=set(text_)
print(len(text_set))
print(d)

11
{'w', 'e', 's', 'm', 'o', 'k', 'g', 'i', 'y', 'h', 'a', 'r', 'd', 'l', 'n', 'b', 'u', ' ', 't'}

text_="Milk is white and so is >><glue, Ghosts are white and they say B00!"
znaki="!@#$$%^&*()_,+<>?/"

text_=text_.strip(",<>")
print(text_,sep="\n")

Milk is whi<>te and so is >><glue, Ghosts are white and they say B00!
```

Встречалось ли число раньше? На вход программе подается строка текста, содержащая числа. Для каждого числа выведите слово YES (в отдельной строке), если это число ранее встречалось в последовательности или NO, если не встречалось.

Формат входных данных На вход программе подается строка текста, содержащая числа, разделенные символом пробела.

Формат выходных данных Программа должна вывести текст в соответствии с условием задачи.

Примечание. Ведущие нули в числах должны игнорироваться.

```

x="1 1 2 2 5 5 5 5 6 7 8".split()
x_=[]
for i in x:
    set_=set(x_)
    x_.append(i)
    if i in set_:
        print("YES")
    else:
        print("NO")

```

```

NO
YES
NO
YES
NO
YES
YES
YES
NO
NO
NO

```

Тема урока: методы множеств Методы union(), intersection(), difference(), symmetric_difference() Методы update(), intersection_update(), difference_update(), symmetric_difference_update() Операторы &, |, -, ^ Аннотация. Урок посвящен методам множеств, которые реализуют основные операции над множествами.

Операции над множествами Основные операции над множествами:

объединение множеств; пересечение множеств; разность множеств; симметрическая разность множеств. Для каждой операции есть метод и оператор.

Объединение множеств: метод union() Объединение множеств – это множество, состоящее из элементов, принадлежащих хотя бы одному из объединяемых множеств. Для этой операции существует метод union().

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1.union(myset2) print(myset3)
```

выводит (порядок элементов может отличаться):

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

Обратите внимание, метод union() возвращает новое множество в которое входят все элементы множеств myset1 и myset2. Для изменения текущего множества используется метод update().

Для объединения двух множеств можно также использовать оператор |.

Результат выполнения приведенного ниже кода:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1 | myset2 print(myset3) аналогичен предыдущему.
```

Пересечение множеств: метод `intersection()` Пересечение множеств – это множество, состоящее из элементов, принадлежащих одновременно каждому из пересекающихся множеств. Для этой операции существует метод `intersection()`.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1.intersection(myset2) print(myset3) выводит (порядок элементов может отличаться):
```

{3, 4} Обратите внимание, метод `intersection()` возвращает новое множество в которое входят общие элементы множеств `myset1` и `myset2`. Для изменения текущего множества используется метод `intersection_update()`.

Для пересечения двух множеств можно также использовать оператор `&`.

Результат выполнения приведенного ниже кода:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1 & myset2 print(myset3) аналогичен предыдущему.
```

Разность множеств: метод `difference()` Разность множеств – это множество, в которое входят все элементы первого множества, не входящие во второе множество. Для этой операции существует метод `difference()`.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1.difference(myset2) print(myset3) выводит (порядок элементов может отличаться):
```

{1, 2, 5} Для разности двух множеств можно также использовать оператор `-`.

Результат выполнения приведенного ниже кода:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1 - myset2 print(myset3) аналогичен предыдущему.
```

Обратите внимание: для операции разности множеств важен порядок, в котором указаны множества. Если поменять местами `myset1` и `myset2`, нас ожидает совсем другой результат: элементы, входящие в множество `myset2` и которых нет в множестве `myset1`.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset2.difference(myset1) print(myset3)
```

 выводит (порядок элементов может отличаться):

{8, 6, 7} Симметрическая разность: метод `symmetric_difference()` Симметрическая разность множеств – это множество, включающее все элементы исходных множеств, не принадлежащие одновременно обоим исходным множествам. Для этой операции существует метод `symmetric_difference()`.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1.symmetric_difference(myset2) print(myset3)
```

 выводит (порядок элементов может отличаться):

{1, 2, 5, 6, 7, 8} Для симметрической разности двух множеств можно также использовать оператор `^`.

Результат выполнения приведенного ниже кода:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset3 = myset1 ^ myset2 print(myset3)
```

 аналогичен предыдущему.

Обратите внимание: для операции симметрической разности порядок множеств не важен, на то она и симметрическая: `myset1 ^ myset2 == myset2 ^ myset1`.

Методы множеств, изменяющие текущие множества Методы `union()`, `intersection()`, `difference()`, `symmetric_difference()` не изменяют исходные множества, а возвращают новые. Часто на практике нужно изменять исходные множества. Для таких целей используются парные методы `update()`, `intersection_update()`, `difference_update()`, `symmetric_difference_update()`.

Метод `update()` Метод `update()` изменяет исходное множество по объединению.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset1.update(myset2) # изменяем множество myset1 print(myset1)
```

 выводит (порядок элементов может отличаться):

{1, 2, 3, 4, 5, 6, 7, 8} Аналогичный результат получается, если использовать оператор `|=`:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

```
myset1 |= myset2 print(myset1)
```

 Метод `intersection_update()` Метод `intersection_update()` изменяет исходное множество по пересечению.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

`myset1.intersection_update(myset2)` # изменяем множество `myset1` `print(myset1)` выводит (порядок элементов может отличаться):

`{3, 4}` Аналогичный результат получается, если использовать оператор `&=`:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

`myset1 &= myset2 print(myset1)` Метод `difference_update()` Метод `difference_update()` изменяет исходное множество по разности.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

`myset1.difference_update(myset2)` # изменяем множество `myset1` `print(myset1)` выводит (порядок элементов может отличаться):

`{1, 2, 5}` Аналогичный результат получается, если использовать оператор `-=`:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

`myset1 -= myset2 print(myset1)` Метод `symmetric_difference_update()` Метод `symmetric_difference_update()` изменяет исходное множество по симметрической разности.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

`myset1.symmetric_difference_update(myset2)` # изменяем множество `myset1` `print(myset1)` выводит (порядок элементов может отличаться):

`{1, 2, 5, 6, 7, 8}` Аналогичный результат получается, если использовать оператор `^=`:

```
myset1 = {1, 2, 3, 4, 5} myset2 = {3, 4, 6, 7, 8}
```

`myset1 ^= myset2 print(myset1)` Примечания Примечание 1. Все основные операции над множествами выполняются двумя способами: при помощи метода или соответствующего ему оператора. Различие заключается в том, что метод может принимать в качестве аргумента не только множество (тип данных `set`), но и любой итерируемый объект (список, строку, кортеж..).

Приведенный ниже код:

```
mylist = [2021, 2020, 2019, 2018, 2017, 2016] mytuple = (2021, 2020, 2016) mystr = 'abcd'
```

```
myset = {2009, 2010, 2016}
```

`print(myset.union(mystr))` # объединяем со строкой `print(myset.intersection(mylist))` # пересекаем со списком `print(myset.difference(mytuple))` # находим разность с кортежем выводит (порядок элементов может отличаться):

`{2016, 'c', 'b', 'a', 'd', 2009, 2010} {2016} {2009, 2010}` Приведенный ниже код:

```
mylist = [2021, 2020, 2019, 2018, 2017, 2016] mytuple = (2021, 2020, 2016) mystr = 'abcd'
myset = {2009, 2010, 2016}
```

print(myset | mystr) print(myset & mylist) print(myset - mytuple) приводит к возникновению ошибок:

TypeError: unsupported operand type(s) for |: 'set' and 'str' TypeError: unsupported operand type(s) for &: 'set' and 'list' TypeError: unsupported operand type(s) for -: 'set' and 'tuple'
Примечание 2. Некоторые методы (union(), intersection(), difference()) и операторы (|, &, -, ^) позволяют совершать операции над несколькими множествами сразу.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5, 6} myset2 = {2, 3, 4, 5} myset3 = {5, 6, 7, 8}
union1 = myset1.union(myset2, myset3) union2 = myset1 | myset2 | myset3
difference1 = myset1.difference(myset2, myset3) difference2 = myset1 - myset2 - myset3 #
порядок выполнения слева-направо
print(union1 == union2) print(difference1 == difference2) выводит:
```

True True Примечание 3. Оператор ^ симметрической разности позволяет использовать несколько множеств, а метод symmetric_difference() – нет.

Приведенный ниже код:

```
myset1 = {1, 2, 3, 4, 5, 6} myset2 = {2, 3, 4, 7} myset3 = {6, 20, 30}
symdifference = myset1 ^ myset2 ^ myset3 # порядок выполнения слева-направо
print(symdifference) выводит (порядок элементов может отличаться):
{1, 5, 7, 20, 30} Приведенный ниже код:
```

```
myset1 = {1, 2, 3, 4, 5, 6} myset2 = {2, 3, 4, 7} myset3 = {6, 20, 30}
symdifference = myset1.symmetric_difference(myset2, myset3)
print(symdifference) приводит к ошибке:
```

TypeError: symmetric_difference() takes exactly one argument (2 given) Примечание 4.
Таблица соответствия методов и операторов 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 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 A ^= B
A.symmetric_difference_update(B) Записывает в A симметрическую разность множеств A и B
Примечание 5. Приоритет операторов в порядке убывания (верхние операторы имеют
более высокий приоритет, чем нижние) имеет вид:

Оператор Описание

- разность & пересечение ^ симметрическая разность | объединение Тут можно посмотреть про операторы и их приоритеты в Python.

Онлайн-школа BEEGEEK 5 Каждый ученик, обучающийся в онлайн-школе BEEGEEK изучает либо математику, либо информатику, либо оба этих предмета. У руководителя школы есть списки учеников, изучающих каждый предмет. Случайно списки всех учеников перемешались.

Напишите программу, которая позволит руководителю выяснить, сколько учеников изучает только один предмет.

Формат входных данных На вход программе в первых двух строках подаются числа m и n – количества учеников, изучающих математику и информатику соответственно. Далее идут

- $m+n$ строк — фамилии учеников, изучающих математику и информатику, в произвольном порядке.

Формат выходных данных Программа должна вывести количество учеников, которые изучают только один предмет. Если таких учеников не окажется, то необходимо вывести NO.

Примечание. Гарантируется, что среди учеников школы BEEGEEK нет однофамильцев.

#Словари Новый раздел

Особенности словарей

Словари (тип данных `dict`) довольно просты, но о нескольких моментах следует помнить при их использовании.

Ключи должны быть уникальными

Словарь не может иметь два и более значений по одному и тому же ключу. Если при создании словаря (в литеральной форме) указать дважды один и тот же ключ, будет использовано последнее из указанных значений.

Приведенный ниже код:

```
info = {'name': 'Ruslan',
        'age': 28,
        'name': 'Timur'}
```

```
print(info['name'])
```

выводит:

Timur

Ключи должны быть неизменяемым типом данных

Ключом словаря могут быть данные любого неизменяемого типа:

число;
строка;

булево значение;
кортеж;
замороженное множество (`frozenset`);

...

Приведенный ниже код создает словарь, ключами которого являются неизменяемые типы данных:

```
my_dict = {198: 'beegeek', 'name': 'Bob', True: 'a', (2, 2): 25}
```

Ключ словаря не может относиться к изменяемому типу данных:

список;
множество;
словарь;

...

Приведенный ниже код приводит к возникновению ошибки:

```
my_dict = {[2, 2]: 25, {1, 2}: 'python', 'name': 'Bob'}
```

Значения могут относиться к любому типу данных, их тип данных произволен

Нет никаких ограничений для значений, хранящихся в словарях. Значения в словарях могут принадлежать к произвольному типу данных и повторяться для разных ключей многократно.

```
my_dict1 = {'a': [1, 2, 3], 'b': {1, 2, 3}}           # значения —  
изменяемый тип данных
```

```
my_dict2 = {'a': [1, 2], 'b': [1, 2], 'c': [1, 2]}   # значения  
повторяются
```

Тема урока: словари в Python

Новый тип коллекции

Отличия словарей от списков

Создание словарей

Обращение по ключу

Встроенная функция `dict()`

Создание словарей на основе списков и кортежей

Пустой словарь

Вывод словаря

Особенности словарей

Аннотация. В этом уроке мы начнем изучение словарей в Python, тип данных — `dict`. Этот тип данных похож на списки и применяется при решении многих задач.

Словари

В прошлых уроках мы изучили четыре типа коллекций в Python:

списки — изменяемые коллекции элементов, индексируемые;
строки — неизменяемые коллекции символов, индексируемые;
кортежи — неизменяемые коллекции элементов, индексируемые;
множества — изменяемые коллекции уникальных элементов,

неиндексируемые.

Следующий тип – словари – изменяемые коллекции элементов с произвольными индексами – ключами. Если в списках элементы индексируются целыми числами, начиная с 0, то в словарях – любыми ключами, в том числе в виде строк.

Как нам уже известно, списки – удобный и самый популярный способ хранения большого количества данных в одной переменной. Списки индексируют все хранящиеся в них элементы, что позволяет быстро обращаться к элементу, зная его индекс.

Приведенный ниже код:

```
languages = ['Python', 'C#', 'Java', 'C++']
```

```
print(languages[0])
```

```
print(languages[2])
```

выводит:

Python

Java

Допустим, мы хотим хранить имя создателя каждого языка программирования. Это можно сделать несколькими способами.

Способ 1. Хранить еще один список, где по соответствующему индексу будет находиться имя создателя языка программирования.

Приведенный ниже код:

```
languages = ['Python', 'C#', 'Java', 'C++']
```

```
creators = ['Гвидо ван Россум', 'Андерс Хейлсберг', 'Джеймс Гослинг',  
'Бьёрн Страуструп']
```

```
print('Создателем языка', languages[0], 'является', creators[0])
```

выводит:

Создателем языка Python является Гвидо ван Россум

Подход рабочий, но хранить данные в двух коллекциях не очень удобно.

Способ 2. Хранить список кортежей с парами значений "язык - имя создателя" в каждом.

Приведенный ниже код:

```
languages = [('Python', 'Гвидо ван Россум'),  
             ('C#', 'Андерс Хейлсберг'),  
             ('Java', 'Джеймс Гослинг'),  
             ('C++', 'Бьёрн Страуструп')]
```

```
print('Создателем языка', languages[2][0], 'является', languages[2]  
[1])
```

Выводит:

Создателем языка Java является Джеймс Гослинг
Тоже рабочий подход, однако не очень эффективный. Придется написать цикл `for` для поиска по всем элементам списка `languages` кортежа, первый элемент которого равен искомому (названию языка). Чтобы найти автора языка C++ , нужно будет в цикле пройти мимо Python, C# и Java. Не получится угадать заранее, что язык C++ лежит после них.

Приведенный ниже код:

```
languages = [('Python', 'Гвидо ван Россум'),
              ('C#', 'Андерс Хейлсберг'),
              ('Java', 'Джеймс Гослинг'),
              ('C++', 'Бьёрн Страуструп')]

for item in languages:
    if item[0] == 'C++':
        print('Создателем языка', item[0], 'является', item[1])
```

Выводит:

Создателем языка C++ является Бьёрн Страуструп
Списки индексируются целыми числами, но в этом случае удобно было бы находить информацию не по числу, а по строке — названию языка программирования. В списках строки не могут быть индексами, однако в словарях это возможно.

Словарь (тип данных `dict`), как и список, позволяет хранить много данных. В отличие от списка, в словаре для каждого элемента можно произвольно определить «индекс» — ключ, по которому он будет доступен.

Словарь — реализация структуры данных "ассоциативный массив" или "хеш-таблица". В других языках аналогичная структура называется `map`, `HashMap`, `Dictionary`.

Создание словаря

Чтобы создать словарь, нужно перечислить его элементы — пары ключ-значение — через запятую в фигурных скобках, как и элементы множества. Первым указывается ключ, после двоеточия — значение, доступное в словаре по этому ключу.

Приведенный ниже код:

```
languages = {'Python': 'Гвидо ван Россум',
             'C#': 'Андерс Хейлсберг',
             'Java': 'Джеймс Гослинг',
             'C++': 'Бьёрн Страуструп'}
```

создает словарь, в котором ключом служит строка — название языка программирования, а значением — имя создателя языка.

Обращение к элементу словаря

Извлечь значение элемента словаря можно, обратившись к нему по его ключу. Чтобы получить значение по заданному ключу, как и в списках, используем квадратные скобки [], индексируем по ключу.

Способ 3. Приведенный ниже код:

```
languages = {'Python': 'Гвидо ван Россум',
             'C#': 'Андерс Хейлсберг',
             'Java': 'Джеймс Гослинг',
             'C++': 'Бьёрн Страуструп'}

print('Создателем языка C# является', languages['C#'])
```

выводит:

Создателем языка C# *является Андерс Хейлсберг*
В отличие от списков, номеров позиций в словарях нет.

Приведенный ниже код:

```
languages = {'Python': 'Гвидо ван Россум',
             'C#': 'Андерс Хейлсберг',
             'Java': 'Джеймс Гослинг',
             'C++': 'Бьёрн Страуструп'}

print('Создателем языка C# является', languages[1])
```

приводит к возникновению ошибки `KeyError`.

Ошибка `KeyError` возникнет и при попытке извлечь значение по несуществующему ключу. В качестве ключа можно указать выражение: Python вычислит его значение и обратится к искомому элементу.

Способ 4. Приведенный ниже код:

```
languages = {'Python': 'Гвидо ван Россум',
             'C#': 'Андерс Хейлсберг',
             'Java': 'Джеймс Гослинг',
             'C++': 'Бьёрн Страуструп'}

print('Создателем языка C# является', languages['C' + '#'])
```

выводит:

Создателем языка C# *является Андерс Хейлсберг*
Создание словаря с помощью функции `dict()`
Если ключи словаря — строки без каких-либо специальных символов, то для создания словаря можно использовать функцию `dict()`.

Приведенный ниже код:

```
info = dict(name = 'Timur', age = 28, job = 'Teacher')
```

создает словарь с тремя элементами, ключами которого служат строки

'name', 'age', 'job', а значениями — 'Timur', 28, 'Teacher'.

Создание словаря на основании списков и кортежей

Создавать словари можно на основе списков кортежей или кортежей списков. Первый элемент списка или кортежа станет ключом, второй — значением.

Приведенный ниже код:

```
info_list = [('name', 'Timur'), ('age', 28), ('job', 'Teacher')] #  
список кортежей
```

```
info_dict = dict(info_list) # создаем словарь на основе списка  
кортежей
```

создаст словарь с тремя элементами, где ключи — строки name, age, job, а соответствующие им значения — 'Timur', 28, 'Teacher'.

Аналогично работает приведенный ниже код:

```
info_tuple = ('name', 'Timur'), ('age', 28), ('job', 'Teacher') #  
кортеж списков
```

```
info_dict = dict(info_tuple) # создаем словарь на основе кортежа  
списков
```

Если необходимо создать словарь, каждому ключу которого соответствует одно и то же значение, можно воспользоваться методом fromkeys().

Приведенный ниже код:

```
dict1 = dict.fromkeys(['name', 'age', 'job'], 'Missed information')  
создаст словарь с тремя элементами, где ключи — строки 'name', 'age',  
'job', а соответствующие им значения: 'Missed information', 'Missed  
information', 'Missed information'.
```

Если методу fromkeys() не передать второй параметр, то по умолчанию присваивается значение None.

Приведенный ниже код:

```
dict1 = dict.fromkeys(['name', 'age', 'job'])  
создаст словарь с тремя элементами, в которых ключи — строки 'name',  
'age', 'job', а значения — None, None, None.
```

Пустой словарь

Пустой словарь можно создать двумя способами:

с помощью пустых фигурных скобок;

с помощью функции dict().

Приведенный ниже код:

```
dict1 = {}
```

```
dict2 = dict()
```

```
print(dict1)
print(dict2)
print(type(dict1))
print(type(dict2))
```

создает два пустых словаря и выводит:

```
{ }
{ }
<class 'dict'>
<class 'dict'>
```

Вспомните, что создать пустое множество можно, только используя функцию `set()`, потому что пустые фигурные скобки зарезервированы для создания словаря.

Вывод словаря

Для вывода всего словаря можно использовать функцию `print()`:

```
languages = {'Python': 'Гвидо ван Россум',
             'C#': 'Андерс Хейлсберг',
             'Java': 'Джеймс Гослинг'}
```

```
info = dict(name = 'Timur', age = 28, job = 'Teacher')
```

```
print(languages)
print(info)
```

Функция `print()` выводит на экран элементы словаря в фигурных скобках, разделенные запятыми:

```
{'Python': 'Гвидо ван Россум', 'C#': 'Андерс Хейлсберг', 'Java':
'Dжеймс Гослинг'}
{'name': 'Timur', 'age': 28, 'job': 'Teacher'}
```

Начиная с версии Python

3.6

3.6, словари являются упорядоченными, то есть сохраняют порядок следования ключей в порядке их внесения в словарь.

Примечания

Примечание 1. Словари принципиально отличаются от списков по структуре хранения в памяти. Список — последовательная область памяти, то есть все его элементы (указатели на элементы) действительно хранятся в указанном порядке, расположены последовательно. Благодаря этому и можно быстро «прыгнуть» к элементу по его индексу. В словаре же используется специальная структура данных — хеш-таблица. Она позволяет вычислять числовой хеш от ключа и использовать обычные списки, где в качестве индекса элемента берется этот хеш.

Примечание 2. В рамках одного словаря каждый ключ уникален.

Примечание 3. Словари удобно использовать для хранения различных сущностей. Например, если нужно работать с информацией о человеке, то можно хранить все необходимые сведения, включающие такие разные сущности как "возраст", "профессия", "название города", "адрес электронной почты" в одном словаре `info` и легко обращаться к его элементам по ключам:

```
info = {'name': 'Timur',
        'age': 28,
        'job': 'Teacher',
        'city': 'Moscow',
        'email': 'timyr-guev@yandex.ru'}

print(info['name'])
print(info['email'])
```

Примечание 4. Создать словарь на основании двух списков (кортежей) можно с помощью встроенной функции-упаковщика `zip()`, о которой расскажем позже.

Приведенный ниже код:

```
keys = ['name', 'age', 'job']
values = ['Timur', 28, 'Teacher']

info = dict(zip(keys, values))
```

```
print(info)
```

выводит (порядок элементов может отличаться):

```
{'name': 'Timur', 'age': 28, 'job': 'Teacher'}
```

В случае несовпадения длины списков функция самостоятельно отсекает лишние элементы.

```
s={1094482}
s1=set([1094482])
s2=list(str(s1))
s2=set(s2)
print(s)
print(s1)
print(s2)
```

```
{1094482}
{1094482}
{'1', '}', '4', '9', '8', '2', '{', '0'}
```

Особенности словарей Словари (тип данных `dict`) довольно просты, но о нескольких моментах следует помнить при их использовании.

Ключи должны быть уникальными. Словарь не может иметь два и более значений по одному и тому же ключу. Если при создании словаря (в литеральной форме) указать дважды один и тот же ключ, будет использовано последнее из указанных значений.

Приведенный ниже код:

```
info = {'name': 'Ruslan', 'age': 28, 'name': 'Timur'}  
  
print(info['name'])
```

 выводит:

Timur Ключи должны быть неизменяемым типом данных. Ключом словаря могут быть данные любого неизменяемого типа:

число; строка; булево значение; кортеж; замороженное множество (frozenset); ...

Приведенный ниже код создает словарь, ключами которого являются неизменяемые типы данных:

```
my_dict = {198: 'beegeek', 'name': 'Bob', True: 'a', (2, 2): 25}
```

 Ключ словаря не может относиться к изменяемому типу данных:

список; множество; словарь; ... Приведенный ниже код приводит к возникновению ошибки:

```
my_dict = {[2, 2]: 25, {1, 2}: 'python', 'name': 'Bob'}
```

 Значения могут относиться к любому типу данных, их тип данных произволен. Нет никаких ограничений для значений, хранящихся в словарях. Значения в словарях могут принадлежать к произвольному типу данных и повторяться для разных ключей многократно.

```
my_dict1 = {'a': [1, 2, 3], 'b': {1, 2, 3}}
```

 # значения – изменяемый тип данных

```
my_dict2 = {'a': [1, 2], 'b': [1, 2], 'c': [1, 2]}
```

 # значения повторяются

Тема урока: словари в Python. Встроенные функции len(), sum(), min(), max(). Оператор принадлежности in. Перебор словарей. Распаковка словаря. Форматированный вывод словарей. Сравнение словарей. Методы keys(), values() и items(). Аннотация. В этом уроке мы изучим основной функционал при работе со словарями.

Основы работы со словарями. Работа со словарями похожа на работу со списками, поскольку и словари, и списки содержат в качестве отдельных элементов пары: в словарях ключ: значение, в списках индекс: значение.

Функция len(). Длиной словаря называется количество его элементов. Для определения длины словаря используют встроенную функцию len() (от слова length – длина).

Следующий программный код:

```
fruits = {'Apple': 70, 'Grape': 100, 'Banana': 80}  
capitals = {'Россия': 'Москва', 'Франция': 'Париж'}
```

```
print(len(fruits)) print(len(capitals))
```

 выведет:

3 2 Оператор принадлежности in. Оператор in позволяет проверить, содержит ли словарь заданный ключ.

Рассмотрим код:

```
capitals = {'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага'}
```

if 'Франция' in capitals: print('Столица Франции - это', capitals['Франция']) Такой код проверяет, содержит ли словарь capitals элемент с ключом Франция и выводит соответствующий текст:

Столица Франции - это Париж Можно использовать оператор in вместе с логическим оператором not.

Не забывайте, что при обращении по несуществующему ключу, возникнет ошибка во время выполнения программы.

Приведенный ниже код:

```
capitals = {'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага'}
```

```
print(capitals['Англия'])
```

 приводит к возникновению ошибки:

KeyError: 'Англия' Оператор принадлежности in на словарях работает очень быстро, намного быстрее, чем на списках, поэтому если нужен многократный поиск в коллекции данных, словарь – подходящий выбор.

Встроенные функции sum(), min(), max() Встроенная функция sum() принимает в качестве аргумента словарь с числовыми ключами и вычисляет сумму его ключей.

Следующий программный код:

```
my_dict = {10: 'Россия', 20: 'США', 30: 'Франция'}
```

```
print('Сумма всех ключей словаря =', sum(my_dict))
```

 выводит:

Сумма всех ключей словаря = 60 Для корректной работы функции sum() ключами словаря должны быть именно числа.

Встроенные функции min() и max() принимают в качестве аргумента словарь и находят минимальный и максимальный ключ соответственно, при этом ключ может принадлежать к любому типу данных, для которого возможны операции порядка <, <=, >, >= (числа, строки, и т.д.).

Следующий программный код:

```
capitals = {'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага'} months = {1: 'Январь', 2: 'Февраль', 3: 'Март'}
```

```
print('Минимальный ключ =', min(capitals)) print('Максимальный ключ =', max(months))
```

 выводит:

Минимальный ключ = Россия Максимальный ключ = 3 Сравнение словарей Словари можно сравнивать между собой. Равные словари имеют одинаковое количество элементов и содержат равные элементы (ключ: значение). Для сравнения словарей используются операторы == и !=.

Приведенный ниже код:


```
months1 = {1: 'Январь', 2: 'Февраль'} months2 = {1: 'Январь', 2: 'Февраль', 3: 'Март'} months3 = {3: 'Март', 1: 'Январь', 2: 'Февраль'}
```

`print(months1 == months2) print(months2 == months3) print(months1 != months3)` выводит:

False True True
Примечания
Примечание 1. Обращение по индексу и срезы недоступны для словарей.

Примечание 2. Операция конкатенации `+` и умножения на число `*` недоступны для словарей.

Примечание 3. Словари нужно использовать в следующих случаях:

Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключи — названия объектов, а значения — их количество. Хранение каких-либо данных, связанных с объектом. Ключи — наименования объектов, значения — связанные с ними данные. Например, если нужно по названию месяца определить его порядковый номер, то это можно сделать при помощи словаря `num = {'January': 1, 'February': 2, 'March': 3, ...}`. Установка соответствия между объектами (например, “родитель—потомок”). Ключ — объект, значение — соответствующий ему объект. Если нужен обычный список, где максимальное значение индекса элемента очень велико, но при этом используются не все возможные индексы (так называемый “разреженный список”), то для экономии памяти можно использовать словарь. Примечание 4. О том, как устроен словарь (тип `dict`) в Python можно почитать в статье.

Примечание 5. Исходный код словаря (тип `dict`) в Python можно найти тут.

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

for i in capitals.items():
    print(i)

('Россия', 'Москва')
('Франция', 'Париж')
('Чехия', 'Прага')

capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

for i in capitals:
    print(i, "-", capitals[i])

Россия - Москва
Франция - Париж
Чехия - Прага
```

Дополните приведенный код, чтобы он вывел имена всех пользователей (в алфавитном порядке), чей номер оканчивается на 8

Примечание. Имена необходимо вывести на одной строке, разделяя символом пробела.

```

students = {
    "Alice": "A",
    "Bob": "B",
    "Charlie": "C",
    "Dave": "B",
    "Eve": "A"
}

target_grade = "B"

keys = list(students.keys())
values=list(students.values())
index = list(students.values()).index(target_grade)
key = keys[index]
print(keys )
print(values)
print(index )
print(key )

```

```

['Alice', 'Bob', 'Charlie', 'Dave', 'Eve']
['A', 'B', 'C', 'B', 'A']
1
Bob

```

```

students = {
    "Alice": "A",
    "Bob": "B",
    "Charlie": "C",
    "Dave": "B",
    "Eve": "A"
}

target_grade = "B"

keys = [a for a in students if students[a] == target_grade]
print("Keys with target grade:", keys)

```

```

Keys with target grade: ['Bob', 'Dave']

```

```

users = [{ 'name': 'Todd', 'phone': '551-1414', 'email':
'todd@gmail.com'},
        { 'name': 'Helga', 'phone': '555-1618'},
        { 'name': 'Olivia', 'phone': '449-3141', 'email': ''},
        { 'name': 'LJ', 'phone': '555-2718', 'email': 'lj@gmail.net'},
        { 'name': 'Ruslan', 'phone': '422-145-9098', 'email': 'rus-
lan.cha@yandex.ru'},
        { 'name': 'John', 'phone': '233-421-32', 'email': ''},
        { 'name': 'Lara', 'phone': '+7998-676-2532', 'email':
'g.lara89@gmail.com'},
        { 'name': 'Alina', 'phone': '+7948-799-2434'},

```

```

        {'name': 'Robert', 'phone': '420-2011', 'email': ''},
        {'name': 'Riyad', 'phone': '128-8890-128', 'email':
'r.mahrez@mail.net'},
        {'name': 'Khabib', 'phone': '+7995-600-9080', 'email':
'kh.nurmag@gmail.com'},
        {'name': 'Olga', 'phone': '6449-314-1213', 'email': ''},
        {'name': 'Roman', 'phone': '+7459-145-8059'},
        {'name': 'Maria', 'phone': '12-129-3148', 'email':
'm.sharapova@gmail.com'},
        {'name': 'Fedor', 'phone': '+7445-341-0545', 'email': ''},
        {'name': 'Tim', 'phone': '242-449-3141', 'email':
'timm.ggg@yandex.ru'}}]
users_out=[]
for i in users:
    if 'email' not in i or i['email'] == "":
        users_out.append(i["name"])
print(*sorted(users_out))

```

Alina Fedor Helga John Olga Olivia Robert Roman

Дополните приведенный код, чтобы он вывел имена всех пользователей (в алфавитном порядке), у которых нет информации об электронной почте.

Примечание 1. Ключ email может отсутствовать в словаре.

Примечание 2. Имена необходимо вывести на одной строке, разделяя символом пробела.

```

users = [{'name': 'Todd', 'phone': '551-1414', 'email':
'todd@gmail.com'},
        {'name': 'Helga', 'phone': '555-1618'},
        {'name': 'Olivia', 'phone': '449-3141', 'email': ''},
        {'name': 'LJ', 'phone': '555-2718', 'email': 'lj@gmail.net'},
        {'name': 'Ruslan', 'phone': '422-145-9098', 'email': 'rus-
lan.cha@yandex.ru'},
        {'name': 'John', 'phone': '233-421-32', 'email': ''},
        {'name': 'Lara', 'phone': '+7998-676-2532', 'email':
'g.lara89@gmail.com'},
        {'name': 'Alina', 'phone': '+7948-799-2434'},
        {'name': 'Robert', 'phone': '420-2011', 'email': ''},
        {'name': 'Riyad', 'phone': '128-8890-128', 'email':
'r.mahrez@mail.net'},
        {'name': 'Khabib', 'phone': '+7995-600-9080', 'email':
'kh.nurmag@gmail.com'},
        {'name': 'Olga', 'phone': '6449-314-1213', 'email': ''},
        {'name': 'Roman', 'phone': '+7459-145-8059'},
        {'name': 'Maria', 'phone': '12-129-3148', 'email':
'm.sharapova@gmail.com'},
        {'name': 'Fedor', 'phone': '+7445-341-0545', 'email': ''},
        {'name': 'Tim', 'phone': '242-449-3141', 'email':
'timm.ggg@yandex.ru'}}]

```

```

users_out=[]
for i in users:
    if 'email' not in i or i['email'] == "":
        users_out.append(i["name"])
print(*sorted(users_out))

```

Строковое представление Напишите программу, которая будет превращать натуральное число в строку, заменяя все цифры в числе на слова:

0 0 на zero; 1 1 на one; 2 2 на two; 3 3 на three; 4 4 на four; 5 5 на five; 6 6 на six; 7 7 на seven; 8 8 на eight; 9 9 на nine.

```

d = {
    0: "zero",
    1: "one",
    2: "two",
    3: "three",
    4: "four",
    5: "five",
    6: "six",
    7: "seven",
    8: "eight",
    9: "nine"
}
n=list(map(int,list(input())))
d_=[]

for i in n:
    d_.append(d[i])
print(*d_)

123
one two three

```

Информация об учебных курсах Напишите программу, которая по номеру курса выводит информацию о данном курсе.

Номер курса (ключ) Номер аудитории (значение) Преподаватель (значение) Время
 (значение) CS101 3004 Хайнс 8:00 CS102 4501 Альварадо 9:00 CS103 6755 Рич 10:00 NT110
 1244 Берк 11:00 CM241 1411 Ли 13:00

```

nd = { 'CS101': '3004, Хайнс, 8:00',
        'CS102': '4501, Альварадо, 9:00',
        'CS103': '6755, Рич, 10:00',
        'NT110': '1244, Берк, 11:00',
        'CM241': '1411, Ли, 13:00' }
d=input()
if d in nd:
    print(nd[d])

```

CS101
3004, Хайнс, 8:00

Набор сообщений На мобильных кнопочных телефонах текстовые сообщения можно отправлять с помощью цифровой клавиатуры. Поскольку с каждой клавишей связано несколько букв, для большинства букв требуется несколько нажатий клавиш. При однократном нажатии цифры генерируется первый символ, указанный для этой клавиши. Нажатие цифры 2, 3, 4 2,3,4 или 5 5 раз генерирует второй, третий, четвертый или пятый символ клавиши.

1 .,:! 2 ABC 3 DEF 4 GHI 5 JKL 6 MNO 7 PQRS 8 TUV 9 WXYZ 0 space (пробел) Напишите программу, которая отображает нажатия клавиш, необходимые для введенного сообщения.

Формат входных данных На вход программе подается одна строка – текстовое сообщение.

Формат выходных данных Программа должна вывести нажатия клавиш, необходимых для введенного сообщения.

Примечание 1. Ваша программа должна обрабатывать как прописные, так и строчные буквы.

Примечание 2. Ваша программа должна игнорировать любые символы, не указанные в приведенной выше таблице.

Примечание 3. Nokia 3310, чтобы вспомнить как это было

Код Морзе Код Морзе для представления цифр и букв использует тире и точки.

Напишите программу для кодирования текстового сообщения в соответствии с кодом Морзе.

Символ Код Символ Код Символ Код Символ Код Символ Код А .- J .---- S ... 1 .---- В -... К -.- Т - 2 .---- С -.-. L .-. U .-. 3 ...- D -. M -- V ...- 4- Е . N -. W .-- 5 F ..- О --- Х -.- 6 -.... G --. Р .-. Y -.- 7 --... Н ... Q --.- Z --. 8 ----. I .. R .-. О ----- 9 ----. Формат входных данных На вход программе подается одна строка – текстовое сообщение.

Формат выходных данных Программа должна вывести закодированное с помощью кода Морзе сообщение, оставляя пробел между каждым закодированным символом (последовательностью тире и точек).

Примечание 1. Ваша программа должна игнорировать любые символы, не перечисленные в таблице.

Примечание 2. Код Морзе был разработан в XIX веке и все еще используется, спустя более 160 лет после создания.

```
# put your python code here
d={".":'1', ",":'11', "?":'111', "!":'1111', ":":'11111',
   "A":'2', "B":'22', "C":'222',
   "D":'3', "E":'33', "F":'333',
   "G":'4', "H":'44', "I":'444',
```



```

'...', '...', '...', '']
morse_dict=dict(zip(letters,morse))
mess_in=[]
mess_out=list((input()).upper())
for i in mess_out:
    if i in morse_dict:
        mess_in.append(morse_dict[i])
s="".join(mess_in)
print(s)
print(mess_in)

```

Interstellar

[illegible]

Interstellar

```
[ '.', '-.', '-', '.-', '.--', '-.-', '--', '-.-.', '-.-.-', '-.-.-.', '-.-.-.-']  
  
d=["1","2","3","4"]  
print(*d)  
s=" ".join(d)  
print(s)
```

1	2	3	4
1	2	3	4

Тема урока: методы словарей
Добавление и изменение элементов в словаре
Удаление элементов из словаря

Методы `get()`, `update()`
Методы `pop()`, `popitem()`
Методы `clear()`, `copy()`
Метод `setdefault()`

Аннотация. В этом уроке мы изучим основные методы словарей.

Методы словарей

Словари, как и списки, имеют много полезных методов для упрощения работы с ними и решения повседневных задач. В прошлом уроке мы уже познакомились с тремя словарными методами:

метод `items()` – возвращает словарные пары ключ: значение, как соответствующие им кортежи;

метод `keys()` – возвращает список ключей словаря;

метод `values()` – возвращает список значений словаря.

Добавление и изменение элементов в словаре

Чтобы изменить значение по определенному ключу в словаре, достаточно использовать индексацию вместе с оператором присваивания. При этом если ключ уже присутствует в словаре, его значение заменяется новым, если же ключ отсутствует – то в словарь будет добавлен новый элемент.

Приведенный ниже код:

```
info = {'name': 'Sam',  
       'age': 28,  
       'job': 'Teacher'}
```

```
info['name'] = 'Timur'           # изменяем значение по ключу  
name
```

```
info['email'] = 'timyr-guev@yandex.ru' # добавляем в словарь элемент  
с ключом email
```

```
print(info)
```

выводит (порядок элементов может отличаться):

```
{'name': 'Timur', 'age': 28, 'job': 'Teacher', 'email': 'timyr-  
guev@yandex.ru'}
```

Обратите внимание на отличие в поведении словарей и списков:

Если в списке `lst` нет элемента с индексом `7`, то попытка обращения к нему, например, с помощью строки кода `print(lst[7])` приведет к возникновению ошибки. И попытка присвоить ему значение `lst[7] = 100` тоже приведет к возникновению ошибки.

Если в словаре `dct` нет элемента с ключом `name`, то попытка обращения к нему, например, с помощью строки кода `print(dct['name'])` приведет к возникновению ошибки. Однако попытка присвоить значение по отсутствующему ключу `dct['name'] = 'Timur'` ошибки не вызовет.

Решим следующую задачу: пусть задан список чисел `numbers`, где некоторые числа встречаются неоднократно. Нужно узнать, сколько именно раз встречается каждое из чисел.


```
numbers = [9, 8, 32, 1, 10, 1, 10, 23, 1, 4, 10, 4, 2, 2, 2, 2, 1, 10, 1, 2, 2, 32, 23, 23]
```

Первый код, который приходит в голову, имеет вид:

```
numbers = [9, 8, 32, 1, 10, 1, 10, 23, 1, 4, 10, 4, 2, 2, 2, 2, 1, 10, 1, 2, 2, 32, 23, 23]
```

```
result = {}  
for num in numbers:  
    result[num] += 1
```

Однако просто так сделать `result[num] += 1` нельзя, так как если ключа `num` в словаре еще нет, то возникнет ошибка `KeyError`.

Следующий программный код полностью решает поставленную задачу:

```
numbers = [9, 8, 32, 1, 10, 1, 10, 23, 1, 4, 10, 4, 2, 2, 2, 2, 1, 10, 1, 2, 2, 32, 23, 23]
```

```
result = {}  
for num in numbers:  
    if num not in result:  
        result[num] = 1  
    else:  
        result[num] += 1
```

Цикл `for` перебирает все элементы списка `numbers` и для каждого проверяет, присутствует ли он уже в качестве ключа в словаре `result`. Если значение отсутствует, значит, число встретилось впервые и мы инициализируем значение `result[num] = 1`. Если значение уже присутствует в словаре, увеличим `result[num]` на единицу.

Этот код можно улучшить с помощью метода `get()`.

Метод `get()`

Мы можем получить значение в словаре по ключу с помощью индексации, то есть квадратных скобок. Однако, как мы знаем, в случае отсутствия ключа будет происходить ошибка `KeyError`.

Приведенный ниже код:

```
info = {'name': 'Bob',  
        'age': 25,  
        'job': 'Dev'}
```

```
print(info['name'])
```

выводит:

Bob

Приведенный ниже код:

```
info = {'name': 'Bob',  
        'age': 25,  
        'job': 'Dev'}
```

```
print(info['salary'])
```

приводит к возникновению ошибки:

KeyError: 'salary'

Для того чтобы избежать возникновения ошибки в случае отсутствия ключа в словаре, можно использовать метод `get()`, способный кроме ключа принимать и второй аргумент — значение, которое вернется, если заданного ключа нет. Когда второй аргумент не указан, то метод в случае отсутствия ключа возвращает **None**.

Приведенный ниже код:

```
info = {'name': 'Bob',  
        'age': 25,  
        'job': 'Dev'}
```

```
item1 = info.get('salary')  
item2 = info.get('salary', 'Информации о зарплате нет')
```

```
print(item1)  
print(item2)
```

выводит:

None

Информации о зарплате нет

С помощью словарного метода `get()` можно упростить код в задаче о повторяющихся числах.

```
numbers = [9, 8, 32, 1, 10, 1, 10, 23, 1, 4, 10, 4, 2, 2, 2, 2, 1, 10,  
1, 2, 2, 32, 23, 23]
```

```
result = {}  
for num in numbers:  
    result[num] = result.get(num, 0) + 1
```

Цикл `for` перебирает все элементы списка `numbers` и для каждого элемента с помощью метода `get()` мы получаем либо его значение из словаря `result`, либо значение по умолчанию, равное **0**. К данному значению прибавляется единица, и результат записывается обратно в словарь по нужному ключу.

Метод `update()`

Метод `update()` реализует своеобразную операцию конкатенации для словарей. Он объединяет ключи и значения одного словаря с ключами и значениями другого. При совпадении ключей в итоге сохранится значение словаря, указанного в качестве аргумента метода `update()`.

Приведенный ниже код:

```
info1 = {'name': 'Bob',  
        'age': 25,  
        'job': 'Dev'}  
  
info2 = {'age': 30,  
        'city': 'New York',  
        'email': 'bob@web.com'}
```

```
info1.update(info2)
```

```
print(info1)
```

выводит (порядок элементов может отличаться):

```
{'name': 'Bob', 'age': 30, 'job': 'Dev', 'city': 'New York', 'email':  
'bob@web.com'}
```

В Python 3.9 появились операторы `|` и `|=`, которые реализуют операцию конкатенации словарей.

Приведенный ниже код:

```
info1 = {'name': 'Bob',  
        'age': 25,  
        'job': 'Dev'}  
  
info2 = {'age': 30,  
        'city': 'New York',  
        'email': 'bob@web.com'}
```

```
info1 |= info2
```

```
print(info1)
```

аналогичен предыдущему коду.

Метод `setdefault()`

Метод `setdefault()` позволяет получить значение из словаря по заданному ключу, автоматически добавляя элемент словаря, если он отсутствует.

Метод принимает два аргумента:

key: ключ, значение по которому следует получить, если таковое имеется в словаре, либо создать.

default: значение, которое будет использовано при добавлении нового элемента в словарь.

В зависимости от значений параметров `key` и `default` возможны следующие сценарии работы данного метода.

Сценарий 1. Если ключ `key` присутствует в словаре, то метод возвращает значение по заданному ключу (независимо от того, передан параметр `default` или нет).

Приведенный ниже код:

```
info = {'name': 'Bob',  
       'age': 25}  
  
name1 = info.setdefault('name')      # параметр default не задан  
name2 = info.setdefault('name', 'Max') # параметр default задан  
  
print(name1)  
print(name2)  
выводит:
```

Bob

Bob

Сценарий 2. Если ключ key отсутствует в словаре, то метод вставляет переданное значение default по заданному ключу.

Приведенный ниже код:

```
info = {'name': 'Bob',  
       'age': 25}  
  
job = info.setdefault('job', 'Dev')  
print(info)  
print(job)  
выводит:
```

```
{'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

Dev

При этом если значение default не передано в метод, то вставится значение **None**.

Приведенный ниже код:

```
info = {'name': 'Bob',  
       'age': 25}  
  
job = info.setdefault('job')  
print(info)  
print(job)  
выводит:
```

```
{'name': 'Bob', 'age': 25, 'job': None}
```

None

Удаление элементов из словаря

Существует несколько способов удаления элементов из словаря:

оператор **del**;

метод **pop()**;

метод **popitem()**;

метод `clear()`.

Оператор `del`

С помощью оператора `del` можно удалять элементы словаря по определенному ключу.

Следующий программный код:

```
info = {'name': 'Sam',  
        'age': 28,  
        'job': 'Teacher',  
        'email': 'timyr-guev@yandex.ru'}
```

```
del info['email']      # удаляем элемент имеющий ключ email  
del info['job']        # удаляем элемент имеющий ключ job
```

```
print(info)
```

выводит (порядок элементов может отличаться):

```
{'name': 'Sam', 'age': 28}
```

Если удаляемого ключа в словаре нет, возникнет ошибка `KeyError`.

Метод `pop()`

Оператор `del` удаляет из словаря элемент по заданному ключу вместе с его значением. Если требуется получить само значение удаляемого элемента, то нужен метод `pop()`.

Следующий программный код:

```
info = {'name': 'Sam',  
        'age': 28,  
        'job': 'Teacher',  
        'email': 'timyr-guev@yandex.ru'}
```

```
email = info.pop('email')      # удаляем элемент по ключу email,  
                               # возвращая его значение  
job = info.pop('job')          # удаляем элемент по ключу job,  
                               # возвращая его значение
```

```
print(email)
```

```
print(job)
```

```
print(info)
```

выводит:

```
timyr-guev@yandex.ru
```

```
Teacher
```

```
{'name': 'Sam', 'age': 28}
```

Единственное отличие этого способа удаления от оператора `del` — он возвращает удаленное значение. В остальном этот способ идентичен оператору `del`. В частности, если удаляемого ключа в словаре нет, возникнет ошибка `KeyError`.

Чтобы ошибка не появлялась, этому методу можно передать второй аргумент. Он будет возвращен, если указанного ключа в словаре нет. Это позволяет реализовать безопасное удаление элемента из словаря:

```
surname = info.pop('surname', None)
```

Если ключа surname в словаре нет, то в переменной surname будет храниться значение `None`.

Метод `popitem()`

Метод `popitem()` удаляет из словаря последний добавленный элемент и возвращает удаляемый элемент в виде кортежа (ключ, значение).

Следующий программный код:

```
info = {'name': 'Bob',  
       'age': 25,  
       'job': 'Dev'}
```

```
info['surname'] = 'Sinclar'
```

```
item = info.popitem()
```

```
print(item)
```

```
print(info)
```

ВЫВОДИТ:

```
('surname', 'Sinclar')
```

```
{'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

В версиях Python ниже 3.6 метод `popitem()` удалял случайный элемент.

Метод `clear()`

Метод `clear()` удаляет все элементы из словаря.

Следующий программный код:

```
info = {'name': 'Bob',  
       'age': 25,  
       'job': 'Dev'}
```

```
info.clear()
```

```
print(info)
```

выведет:

```
{}
```

Метод `copy()`

Метод `copy()` создает поверхностную копию словаря.

Следующий программный код:

```
info = {'name': 'Bob',  
        'age': 25,  
        'job': 'Dev'}
```

```
info_copy = info.copy()
```

```
print(info_copy)
```

выведет:

```
{'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

Не стоит путать копирование словаря (метод `copy()`) и присвоение новой переменной ссылки на старый словарь.

Следующий программный код:

```
info = {'name': 'Bob',  
        'age': 25,  
        'job': 'Dev'}
```

```
new_info = info  
new_info['name'] = 'Tim'
```

```
print(info)
```

выводит:

```
{'name': 'Tim', 'age': 25, 'job': 'Dev'}
```

Оператор присваивания (`=`) не копирует словарь, а лишь присваивает ссылку на старый словарь новой переменной.

Таким образом, когда мы изменяем словарь `new_info`, меняется и словарь `info`. Если необходимо изменить один словарь, не изменяя второй, используют метод `copy()`.

Следующий программный код:

```
info = {'name': 'Bob',  
        'age': 25,  
        'job': 'Dev'}
```

```
new_info = info.copy()  
new_info['name'] = 'Tim'
```

```
print(info)
```

```
print(new_info)
```

выводит:

```
{'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

```
{'name': 'Tim', 'age': 25, 'job': 'Dev'}
```

Примечания

Примечание 1. Словарь можно использовать вместо нескольких вложенных условий, если вам нужно проверить число на равенство. Например, вместо

```
num = int(input())

if num == 1:
    description = 'One'
elif num == 2:
    description = 'Two'
elif num == 3:
    description = 'Three'
else:
    description = 'Unknown'
```

```
print(description)
```

можно написать

```
num = int(input())

description = {1: 'One', 2: 'Two', 3: 'Three'}
```

```
print(description.get(num, 'Unknown'))
```

На практике такой код встречается достаточно часто, особенно если в программе необходимо часто осуществлять проверку указанного типа.

```
capitals = {'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага'}
months = {1: 'Январь', 2: 'Февраль', 3: 'Март'}
asd=capitals.setdefault("Yemen","Sanaa")
print('Минимальный ключ =', max(capitals))
print('Максимальный ключ =', max(months))
print(capitals,asd)
```

Минимальный ключ = Чехия

Максимальный ключ = 3

```
{'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага', 'Yemen':
'Sanaa'} Sanaa
```

```
for i,y in capitals.items():
    if y=="Sanaa":
        print(i)
```

Yemen

```
capitals = {'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага'}
months = {1: 'Январь', 2: 'Февраль', 3: 'Март'}
asd=capitals.setdefault("Yemen","Sanaa")
print('Минимальный ключ =', max(capitals))
print('Максимальный ключ =', max(months))
```


Минимальный ключ = Чехия
Максимальный ключ = 3

```
result = {}  
num=[i for i in range(16)]  
num_2=[i**2 for i in range(16)]  
result = zip(num,num_2)  
print(result)
```

<zip object at 0x7dc8fe795980>

```
dict1 = {'a': 100, 'z': 333, 'b': 200, 'c': 300, 'd': 45, 'e': 98,  
        't': 76, 'q': 34, 'f': 90, 'm': 230}  
dict2 = {'a': 300, 'b': 200, 'd': 400, 't': 777, 'c': 12, 'p': 123,  
        'w': 111, 'z': 666}
```

```
for i in dict1:  
    if i in dict2:  
        dict2[i]=dict1[i]+dict2[i]  
dict1.update(dict2)  
result={}  
result=dict1
```

Дополните приведенный код, чтобы в переменной result хранился словарь, в котором ключи – числа от 11 до 15 (включительно), а значения представляют собой квадраты ключей.

Примечание. Выводить содержимое словаря result не нужно.

```
result={}  
for i in range(1,16):  
    result[i]=i**2
```

Дополните приведенный код так, чтобы он объединил содержимое двух словарей dict1 и dict2: если ключ есть в обоих словарях, добавьте его в результирующий словарь со значением, равным сумме соответствующих значений из первого и второго словаря; если ключ есть только в одном из словарей, добавьте его в результирующий словарь с его текущим значением. Результирующий словарь необходимо присвоить переменной result.

Примечание. Выводить содержимое словаря result не нужно.

Для отладки кода [] Напишите программу. Те

```
dict1 = {'a': 100, 'z': 333, 'b': 200, 'c': 300, 'd': 45, 'e': 98,  
        't': 76, 'q': 34, 'f': 90, 'm': 230}  
dict2 = {'a': 300, 'b': 200, 'd': 400, 't': 777, 'c': 12, 'p': 123,  
        'w': 111, 'z': 666}  
for i in dict1:  
    if i in dict2:
```

```
dict2[i]=dict1[i]+dict2[i]
dict1.update(dict2)
result={}
result=dict1
```

Дополните приведенный код так, чтобы в переменной result хранился словарь, в котором для каждого символа строки text будет подсчитано количество его вхождений.

Примечание. Выводить содержимое словаря result не нужно.

Для отладки кода  Напишите программу.

```
result = {}
text =
'footballcyberpunkextraterritorialityconversationalistblockophthalmosc
opicinterdependencemamauserfff'
for i in text:
    result[i] = text.count(i)
```

Дополните приведенный код, чтобы он вывел наиболее часто встречающееся слово строки s. Если таких слов несколько, должно быть выведено то, что меньше в лексикографическом порядке.

```
s = 'orange strawberry barley gooseberry apple apricot barley currant
orange melon pomegranate banana banana orange barley apricot plum
grapefruit banana quince strawberry barley grapefruit banana grapes
melon strawberry apricot currant currant gooseberry raspberry apricot
currant orange lime quince grapefruit barley banana melon pomegranate
barley banana orange barley apricot plum banana quince lime grapefruit
strawberry gooseberry apple barley apricot currant orange melon
pomegranate banana banana orange apricot barley plum banana grapefruit
banana quince currant orange melon pomegranate barley plum banana
quince barley lime grapefruit pomegranate barley'
repeat_={}
for i in s.split():
    repeat_[i]=s.count(i)
print(repeat_)
max_repeat = max(repeat_, key=repeat_.get)
max_repeat_dict={}
for key,value in repeat_.items():
    if value == max_repeat:
        print (key)
        print(max_repeat)
        max_repeat_dict[key]=value
print(max_repeat_dict)

{'orange': 8, 'strawberry': 4, 'barley': 12, 'gooseberry': 3, 'apple':
2, 'apricot': 7, 'currant': 6, 'melon': 5, 'pomegranate': 5, 'banana':
```

```
12, 'plum': 4, 'grapefruit': 6, 'quince': 5, 'grapes': 1, 'raspberry':  
1, 'lime': 3}  
barley  
{}
```

Вам доступен список `pets`, содержащий информацию о собаках и их владельцах. Каждый элемент списка – это кортеж вида (кличка собаки, имя владельца, фамилия владельца, возраст владельца).

Дополните приведенный код так, чтобы в переменной `result` хранился словарь, в котором для каждого владельца будут перечислены его собаки. Ключом словаря должен быть кортеж (имя, фамилия, возраст владельца), а значением – список кличек собак (сохранив исходный порядок следования).

Примечание 1. Не забывайте: кортежи являются неизменяемыми, поэтому могут быть ключами словаря.

Примечание 2. Обратите внимание, что у некоторых владельцев по несколько собак.

Примечание 3. Выводить содержимое словаря `result` не нужно.

```
pets = [('Hatiko', 'Parker', 'Wilson', 50),  
        ('Rusty', 'Josh', 'King', 25),  
        ('Fido', 'John', 'Smith', 28),  
        ('Butch', 'Jake', 'Smirnoff', 18),  
        ('Odi', 'Emma', 'Wright', 18),  
        ('Balto', 'Josh', 'King', 25),  
        ('Barry', 'Josh', 'King', 25),  
        ('Snake', 'Hannah', 'Taylor', 40),  
        ('Horry', 'Martha', 'Robinson', 73),  
        ('Giro', 'Alex', 'Martinez', 65),  
        ('Zooma', 'Simon', 'Nevel', 32),  
        ('Lassie', 'Josh', 'King', 25),  
        ('Chase', 'Martha', 'Robinson', 73),  
        ('Ace', 'Martha', 'Williams', 38),  
        ('Rocky', 'Simon', 'Nevel', 32)]  
  
result={}  
for i in pets:  
    result.setdefault(i[1:], []).append(i[0])  
print(result)  
  
{('Parker', 'Wilson', 50): ['Hatiko'], ('Josh', 'King', 25): ['Rusty',  
'Balto', 'Barry', 'Lassie'], ('John', 'Smith', 28): ['Fido'], ('Jake',  
'Smirnoff', 18): ['Butch'], ('Emma', 'Wright', 18): ['Odi'],  
(('Hannah', 'Taylor', 40): ['Snake'], ('Martha', 'Robinson', 73):  
['Horry', 'Chase'], ('Alex', 'Martinez', 65): ['Giro'], ('Simon',  
'Nevel', 32): ['Zooma', 'Rocky'], ('Martha', 'Williams', 38): ['Ace']}
```

Самое редкое слово На вход программе подается строка текста. Напишите программу, которая выводит слово, которое встречается реже всего, без учета регистра. Если таких слов несколько, выведите то, которое меньше в лексикографическом порядке.

Формат входных данных На вход программе подается строка текста.

Формат выходных данных Программа должна вывести слово (в нижнем регистре), встречаемое реже всего.

Примечание 1. Программа не должна быть чувствительной к регистру, слова apple и Apple должна воспринимать как одинаковые.

Примечание 2. Слово — последовательность букв. Кроме слов в тексте могут присутствовать пробелы и знаки препинания .,!?:;- , которые нужно игнорировать. Других символов в тексте нет.

```
s = "London is the capital of Great Britain. More than six million  
people live in London. London lies on both banks of the river Thames.  
It is the largest city in Europe and one of the largest cities in the  
world. London is not only the capital of the country, it is also a  
very big port, one of the greatest commercial centres in the world, a  
university city, and the seat of the government of Great Britain!"  
repeat_={}  
replace_="!@#$%^&*()_+<>?/,.".  
for i in s:  
    if i in replace_  
        s=s.replace(i,"")  
  
for i in s.lower().split():  
    repeat_[i]=s.count(i)  
  
d = [key for key, value in repeat_.items() if value == 1]  
#print(d)  
  
#print(min(repeat_, key=repeat_.get))  
print(sorted(d)[0])  
  
['great', 'than', 'six', 'million', 'people', 'live', 'lies', 'both',  
'banks', 'river', 'cities', 'not', 'only', 'country', 'also', 'very',  
'big', 'port', 'greatest', 'commercial', 'centres', 'university',  
'seat', 'government']  
london  
also
```

Исправление дубликатов На вход программе подается строка, содержащая строки-идентификаторы. Напишите программу, которая исправляет их так, чтобы в результирующей строке не было дубликатов. Для этого необходимо прибавлять к повторяющимся идентификаторам постфикс _n, где n – количество раз, сколько такой идентификатор уже встречался.

Формат входных данных На вход программе подается строка текста, содержащая строки-идентификаторы, разделенные символом пробела.

Формат выходных данных Программа должна вывести исправленную строку, не содержащую дубликатов сохранив при этом исходный порядок.

```
s="How are you? how Are?"
repeat_={}
replace_="!@#$$%^&*()_+<>?/,. "
for i in s:
    if i in replace_:
        s=s.replace(i,"")
s= s.lower().split()

for i in s:
    repeat_[i]=s.count(i)
    print(i)
min_ = min(repeat_, key=repeat_.get)
#print(repeat_)
#print(s)
#d = [key for key, value in repeat_.items() if value == min_]
#print(d,min_)

#print(min(repeat_, key=repeat_.get))
print(min_)

how
are
you
how
are
{'how': 2, 'are': 2, 'you': 1}
['how', 'are', 'you', 'how', 'are']
you

lst = input().split()
res = {}
for c in lst:
    if c in res:
        print(f'{c}_{res[c]}', end=' ')
    else:
        print(c, end=' ')
    res[c] = res.get(c, 0) + 1
```

Словарь программиста Программисты, как вы уже знаете, постоянно учатся, а в общении между собой используют весьма специфический язык. Чтобы систематизировать ваш пополняющийся профессиональный лексикон, мы придумали эту задачу. Напишите программу создания небольшого словаря сленговых программистских выражений, чтобы она потом по запросу возвращала значения из этого словаря.

Формат входных данных В первой строке задано одно целое число n — количество слов в словаре. В следующих n строках записаны слова и их определения, разделенные двоеточием и символом пробела. В следующей строке записано целое число m — количество поисковых слов, чье определение нужно вывести. В следующих m строках записаны сами слова, по одному на строке.

Формат выходных данных Для каждого слова, независимо от регистра символов, если оно присутствует в словаре, необходимо вывести его определение. Если слова в словаре нет, программа должна вывести "Не найдено", без кавычек.

Примечание 1. Мини-словарь для начинающих разработчиков можно посмотреть тут.

Примечание 2. Гарантируется, что в определяемом слове или фразе отсутствует двоеточие (:), следом за которым идёт пробел.

```
# put your python code here
n=int(input())
s=[input().split(": ") for i in range(n)]
d={i[0].lower():i[1] for i in s}
m=int(input())
zapros=[input().lower() for i in range(m)]
for i in zapros:
    print(d.get(i,"Не найдено"))

mydict = {}

for _ in range(int(input())):
    key, value = input().split(': ')
    mydict[key.lower()] = value

for _ in range(int(input())):
    print(mydict.get(input().lower(), 'Не найдено'))
```

Анаграммы 1 Анаграмма – слово (словосочетание), образованное путём перестановки букв, составляющих другое слово (или словосочетание). Например, английские слова evil и live – это анаграммы.

На вход программе подаются два слова. Напишите программу, которая определяет, являются ли они анаграммами.

Формат входных данных На вход программе подаются два слова, каждое на отдельной строке.

Формат выходных данных Программа должна вывести YES если слова являются анаграммами и NO в противном случае.

```
s=[input() for i in range(2)]
if sorted(list(s[0])) == sorted(list(s[1])):
    print("YES")
else:
    print("NO")
```

Анаграммы 2 На вход программе подаются два предложения. Напишите программу, которая определяет, являются ли они анаграммами или нет. Ваша программа должна игнорировать регистр символов, знаки препинания и пробелы.

Формат входных данных На вход программе подаются два предложения, каждое на отдельной строке.

Формат выходных данных Программа должна вывести YES, если предложения – анаграммы и NO в противном случае.

Примечание. Кроме слов в тексте могут присутствовать пробелы и знаки препинания .,!?;- . Других символов в тексте нет.

```
s=[input().lower()for i in range(2)]
#s=["asd az&?", "asd az"]
s1='!?,.@$%^&*()'
for i in range(2):
    for j in s[i]:
        if j in '!?,.@$%^&*()':
            s[i]=s[i].replace(j,"")
if list(sorted((s[0]).replace(" ","")) ==
list(sorted((s[1]).replace(" ",""))):
    print("YES")
else:
    print("NO")


#print(list(sorted((s[0]).replace(" ",""))))
#print(list(sorted((s[0]).replace(" ",""))))

def s(word):
    res = {}
    for i in word.lower():
        if i.isalpha():
            res[i] = res.get(i, 0) + 1
    return res

print(("NO", "YES")[s(input()) == s(input())])
```

Словарь синонимов Вам дан словарь, состоящий из пар слов-синонимов. Повторяющихся слов в словаре нет. Напишите программу, которая для одного данного слова определяет его синоним.

Формат входных данных На вход программе подается количество пар синонимов 

n. Далее следует  n строк, каждая строка содержит два слова-синонима. После этого следует одно слово, синоним которого надо найти.

Формат выходных данных Программа должна вывести одно слово, синоним введенного.

Примечание 1. Гарантируется, что синоним введенного слова существует в словаре.

Примечание 2. Все слова в словаре начинаются с заглавной буквы.

```
# put your python code here
dict_={}
n=int(input())
for i in range(n):
    d=input().split()
    dict_[d[0]]=d[1]
    dict_[d[1]]=d[0]

#print(dict_)
find_=input()
print(dict_[find_])

words = {}
for _ in range(int(input())):
    a, b = input().split()
    words[a], words[b] = b, a
print(words[input()])
```

Страны и города

На вход программе подается список стран и городов каждой страны. Затем даны названия городов. Напишите программу, которая для каждого города выводит, в какой стране он находится.

Формат входных данных

Программа получает на вход количество стран

?

n. Далее идет

?

n строк, каждая строка начинается с названия страны, затем идут названия городов этой страны. В следующей строке записано число

?

m, далее идут

?

m запросов – названия каких-то

?

m городов, из перечисленных выше.

Формат выходных данных

Программа должна вывести название страны, в которой находится данный город в соответствии с примером.

```
# put your python code here
dict_={}
n=int(input())
for i in range(n):
    d=input().split()
    dict_[d[0]]=d[1:]

m=int(input())
```



```
finders=[input() for i in range(m)]
for y in finders:
    for i,j in dict_.items():
        #print(i,j)
        if y in j:
            print(i)

d = {}
for _ in range(int(input())):
    country, *cities = input().split()
    d.update(dict.fromkeys(cities, country))
for _ in range(int(input())):
    print(d[input()])
```

Телефонная книга Тимур записал телефоны всех своих друзей, чтобы автоматизировать поиск нужного номера.

У каждого из друзей Тимура может быть один или более телефонных номеров. Напишите программу, которая поможет Тимур находить все номера определённого друга.

Формат входных данных В первой строке задано одно целое число n — количество номеров телефонов, информацию о которых Тимур сохранил в телефонной книге. В следующих n строках заданы телефоны и имена их владельцев через пробел. В следующей строке записано целое число m — количество поисковых запросов от Тимура. В следующих m строках записаны сами запросы, по одному на строке. Каждый запрос — это имя друга, чьи телефоны Тимур хочет найти.

Формат выходных данных Для каждого запроса от Тимура выведите в отдельной строке все телефоны, принадлежащие человеку с этим именем (независимо от регистра имени). Если в телефонной книге нет телефонов человека с таким именем, выведите в соответствующей строке «абонент не найден» (без кавычек).

Примечание 1. Телефоны одного человека выводите в одну строку через пробел в том порядке, в каком они были заданы во входных данных.

Примечание 2. Количество строк в ответе должно быть равно числу m .

m .

Примечание 3. Телефон — это несколько цифр, записанных подряд, а имя может состоять из букв русского или английского алфавита. Записи не повторяются.

```
f={}
n=int(input()) # Ввод данные
for i in range(n):
    d=input().lower().split()
    if d[-1].lower() in f: # проверка повторения
        # ключей
        f[d[-1].lower()]=f[d[-1]]+d[:-1]
    else:
        f[d[-1].lower()]=d[:-1]
```

```

#print(d,f)
m=int(input())
zapros=[input().lower() for i in range(m)]      # Ввод звпросов
#print(zapros)
for i in zapros:                                # Вывод результат
    if i in f:
        print(*f[i])
    else:
        print("абонент не найден")

dct = {}
for _ in range(int(input())):
    phone, name = input().lower().split()
    dct.setdefault(name, []).append(phone)
for _ in range(int(input())):
    print(*dct.get(input().lower(), ['абонент не найден']))

```

Секретное слово Напишите программу для расшифровки секретного слова методом частотного анализа.

Формат входных данных В первой строке задано зашифрованное слово. Во второй строке задано одно целое число n – количество букв в словаре. В следующих n строках записано, сколько раз конкретная буква алфавита встречается в этом слове – $:$.

Формат выходных данных Программа должна вывести дешифрованное слово.

Примечание. Гарантируется, что частоты букв не повторяются.

```

s=input()
s_=list(s)
res={}
d={s_.count(i):i for i in s}
m=int(input())
z=[input().split(": ") for i in range(m)]
z_={i[1]:i[0] for i in z}
for i in d:
    x=z_[str(i)]
    y=d[i]
    res[y]=x
for i in s:
    s=s.replace(i,res[i])
print(s)

dct, word = {}, {}
s = input()
for c in s:
    word[c] = word.get(c, 0) + 1
for _ in range(int(input())):
    a, b = input().split(': ')
    dct[int(b)] = a

```

```
for c in s:
    print(dct[word[c]], end='')
```

Дополните приведенный код, используя генератор, так чтобы получить словарь result, в котором ключом будет позиция числа в списке numbers (начиная с 0 0), а значением – его квадрат.

Примечание. Выводить содержимое словаря result не нужно.

```
numbers = [34, 10, -4, 6, 10, 23, -90, 100, 21, -35, -95, 1, 36, -38,
-19, 1, 6, 87]
result = {i:numbers[i]**2 for i in range(len(numbers))}
```

Дополните приведенный код, используя генератор, чтобы получить словарь result, состоящий из всех элементов словаря colors, кроме тех, у которых значением является None.

Примечание. Выводить содержимое словаря result не нужно.

```
colors = {'c1': 'Red', 'c2': 'Grey', 'c3': None, 'c4': 'Green', 'c5':
'Yellow', 'c6': 'Pink', 'c7': 'Orange', 'c8': None, 'c9': 'White',
'c10': 'Black', 'c11': 'Violet', 'c12': 'Gold', 'c13': None, 'c14':
'Amber', 'c15': 'Azure', 'c16': 'Beige', 'c17': 'Bronze', 'c18': None,
'c19': 'Lilac', 'c20': 'Pearl', 'c21': None, 'c22': 'Sand', 'c23':
None}

result = {i:colors[i] for i in colors.keys() if colors[i] != None }
```

Дополните приведенный код, используя генератор, чтобы получить словарь result, состоящий из всех элементов словаря favorite_numbers, значения которых являются двузначными числами.

Примечание. Выводить содержимое словаря result не нужно.

```
favorite_numbers = {'timur': 17, 'ruslan': 7, 'larisa': 19, 'roman':
123, 'rebecca': 293, 'ronald': 76, 'dorothy': 62, 'harold': 36,
'matt': 314, 'kim': 451, 'rosaly': 18, 'rustam': 89, 'soltan': 111,
'amir': 654, 'dima': 390, 'amiran': 777, 'geor': 999, 'sveta': 75,
'rita': 909, 'kirill': 404, 'olga': 271, 'anna': 55, 'madlen': 876}

result = {i:favorite_numbers[i] for i in favorite_numbers.keys() if
len(str(favorite_numbers[i])) == 2 }
```

Дополните приведенный код, используя генератор, так, чтобы получить словарь result, состоящий из всех элементов словаря months, в которых ключ и значение поменялись местами.

Примечание. Выводить содержимое словаря result не нужно.

```
months = {1: 'January', 2: 'February', 3: 'March', 4: 'April', 5: 'May', 6: 'June', 7: 'July', 8: 'August', 9: 'September', 10: 'October', 11: 'November', 12: 'December'}
```

```
result = {months[i]:i for i in months}
```

В переменной `s` хранится строка пар число:слово. Дополните приведенный код, используя генератор, чтобы получить словарь `result`, в котором числа будут ключами, а слова – значениями.

Примечание 1. Ключи словаря должны быть целыми числами (иметь тип `int`), значения – строками (иметь тип `str`).

Примечание 2. Выводить содержимое словаря `result` не нужно.

```
s = '1:men 2:kind 90:number 0:sun 34:book 56:mountain 87:wood 54:car 3:island 88:power 7:box 17:star 101:ice'
result={int(i.split(":")[0]):i.split(":")[1] for i in s.split()}
```

Используя генератор, дополните приведенный код, чтобы получить словарь `result`, где ключом будет элемент списка `numbers`, а значением – отсортированный по возрастанию список всех его делителей начиная с 1

Примечание 1. Если бы список `numbers` имел вид: `numbers = [1, 6, 18]`, то результатом был бы словарь

`result = {1: [1], 6: [1, 2, 3, 6], 18: [1, 2, 3, 6, 9, 18]}` Примечание 2. Выводить содержимое словаря `result` не нужно.

```
numbers = [34, 10, 4, 6, 10, 23, 90, 100, 21, 35, 95, 1, 36, 38, 19, 1, 6, 87, 1000, 13456, 360]
def key_v(x_key):
    x_value=[i for i in range(1,x_key+1) if x_key % i == 0 ]
    return (x_key ,x_value)

result={key_v(i)[0]:key_v(i)[1] for i in numbers}
```

Дополните приведенный код, используя генератор, так, чтобы получить словарь `result`, в котором ключом будет строка – элемент списка `words`, а значением – список соответствующих кодов ASCII символов данной строки.

Примечание 1. Если бы список `words` имел вид: `words = ['yes', 'hello']`, то результатом был бы словарь

`result = {'yes': [121, 101, 115], 'hello': [104, 101, 108, 108, 111]}` Примечание 2. Для получения ASCII кода символа используйте функцию `ord()`.

Примечание 3. Выводить содержимое словаря `result` не нужно.

```

words = ['hello', 'bye', 'yes', 'no', 'python', 'apple', 'maybe',
'stepik', 'beegeek']

def key_v(x_key):
    x_value=[ord(i) for i in x_key ]
    return (x_key ,x_value)

result={key_v(i)[0]:key_v(i)[1] for i in words}

```

Дополните приведенный код, используя генератор, чтобы получить словарь result, состоящий из всех элементов словаря letters , за исключением тех, ключи которых есть в списке remove_keys.

Примечание. Выводить содержимое словаря result не нужно.

```

letters = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7:
'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15:
'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W',
23: 'X', 24: 'Y', 26: 'Z'}

remove_keys = [1, 5, 7, 12, 17, 19, 21, 24]

result = {}

```

В переменной students хранится словарь, содержащий информацию о росте (в см) и массе (в кг) студентов.

Дополните приведенный код, используя генератор, чтобы получить словарь result, состоящий из всех элементов словаря students , где указан рост больше 167 167 см, а масса меньше 75 75 кг.

Примечание. Выводить содержимое словаря result не нужно.

```

students = {'Timur': (170, 75), 'Ruslan': (180, 105), 'Soltan': (192,
68), 'Roman': (175, 70), 'Madlen': (160, 50), 'Stefani': (165, 70),
'Tom': (190, 90), 'Jerry': (180, 87), 'Anna': (172, 67), 'Scott':
(168, 78), 'John': (186, 79), 'Alex': (195, 120), 'Max': (200, 110),
'Barak': (180, 89), 'Donald': (170, 80), 'Rustam': (186, 100),
'Alice': (159, 59), 'Rita': (170, 80), 'Mary': (175, 69), 'Jane':
(190, 80)}

result = {k: v for k, v in students.items() if v[0] >167 and v[1] < 75
}

```

Список tuples содержит кортежи, состоящие из трех чисел.

Дополните приведенный код, используя генератор, чтобы получить словарь result, в котором ключом является первый элемент каждого кортежа, а значением – кортеж из оставшихся двух элементов.

Примечание. Выводить содержимое словаря result не нужно.

```
tuples = [(1, 2, 3), (4, 5, 6), (7, 8, 9), (10, 11, 12), (13, 14, 15),
(16, 17, 18), (19, 20, 21), (22, 23, 24), (25, 26, 27), (28, 29, 30),
(31, 32, 33), (34, 35, 36)]

result = {i[0]:i[1:] for i in tuples}
```

Даны три списка student_ids, student_names, student_grades, содержащие информацию о студентах.

Дополните приведенный код, используя генератор, так чтобы получить список result, содержащий вложенные словари в соответствии с образцом: [{'S001': {'Camila Rodriguez': 86}}, {'S002': {'Juan Cruz': 98}},...].

Примечание 1. Для параллельной итерации по всем трем спискам одновременно можно использовать встроенную функцию zip().

Примечание 2. Выводить содержимое списка result не нужно.

```
student_ids = ['S001', 'S002', 'S003', 'S004', 'S005', 'S006', 'S007',
'S008', 'S009', 'S010', 'S011', 'S012', 'S013']
student_names = ['Camila Rodriguez', 'Juan Cruz', 'Dan Richards', 'Sam
Boyle', 'Batista Cesare', 'Francesco Totti', 'Khalid Hussain', 'Ethan
Hawke', 'David Bowman', 'James Milner', 'Michael Owen', 'Gary Oldman',
'Tom Hardy']
student_grades = [86, 98, 89, 92, 45, 67, 89, 90, 100, 98, 10, 96, 93]

result = [{student_ids[i]:{student_names[i]:student_grades[i]}} for i
in range(len(student_grades))]
```

Дополните приведенный код, чтобы в списках значений элементов словаря my_dict не было чисел, больших 20

1. При этом порядок оставшихся элементов меняться не должен.

Примечание. Необходимо изменить словарь my_dict, выводить ничего не надо.

Для отладки кода []

```
my_dict = {'C1': [10, 20, 30, 7, 6, 23, 90], 'C2': [20, 30, 40, 1, 2,
3, 90, 12], 'C3': [12, 34, 20, 21], 'C4': [22, 54, 209, 21, 7], 'C5':
[2, 4, 29, 21, 19], 'C6': [4, 6, 7, 10, 55], 'C7': [4, 8, 12, 23, 42],
'C8': [3, 14, 15, 26, 48], 'C9': [2, 7, 18, 28, 18, 28]}

my_dict={key:[i for i in my_dict[key] if i <= 20] for key in my_dict}
```

Словарь emails содержит информацию об электронных адресах пользователей, сгруппированных по домену. Дополните приведенный код, чтобы он вывел все электронные адреса в алфавитном порядке, каждый на отдельной строке, в формате [username@domain](#).

Примечание 1. Для сортировки в алфавитном порядке используйте встроенную функцию `sorted()`, либо списочный метод `sort()`.

Примечание 2. Группировать электронные адреса по доменам не нужно.

```
emails = {'nosu.edu': ['timyr', 'joseph', 'svetlana.gaeva',  
                    'larisa.mamuk'],  
          'gmail.com': ['ruslan.chaika', 'rustam.mini', 'stepik-  
best'],  
          'msu.edu': ['apple.fruit', 'beegeek', 'beegeek.school'],  
          'yandex.ru': ['surface', 'google'],  
          'hse.edu': ['tomas-henders', 'cream.soda', 'zivert'],  
          'mail.ru': ['angel.down', 'joanne', 'the.fame.moster']}  
res=sorted([i + "@" + key for key,value in emails.items() for i in  
value])  
print(*res, sep="\n")
```

Минутка генетики Как известно из курса биологии ДНК и РНК – последовательности нуклеотидов.

Четыре нуклеотида в ДНК:

аденин (A); цитозин (C); гуанин (G); тимин (T). Четыре нуклеотида в РНК:

аденин (A); цитозин (C); гуанин (G); урацил (U). Цепь РНК строится на основе цепи ДНК последовательным присоединением комплементарных нуклеотидов:

$G \rightarrow C$; $C \rightarrow G$; $T \rightarrow A$; $A \rightarrow U$. Напишите программу, переводящую цепь ДНК в цепь РНК.

```
# put your python code here  
d = {'A': 'U', 'C': 'G', 'G': 'C', 'T': 'A'}  
res = [d[elem] for elem in input()]  
  
print(''.join(res))
```

Порядковый номер Дана строка текста на русском языке, состоящая из слов и символов пробела. Словом считается последовательность букв, слова разделены одним пробелом или несколькими.

Напишите программу, определяющую для каждого слова порядковый номер его вхождения в текст именно в этой форме, с учетом регистра. Для первого вхождения слова программа выведет 1 1, для второго вхождения того же слова — 2 2 и т. д.

Формат входных данных Программа получает на вход единственную строку текста, состоящую только из русских букв и символов пробела.

Формат выходных данных Для каждого слова исходного текста программа выводит одно целое число — номер вхождения этого слова в текст. Числа необходимо вывести на одной строке через пробел.

Примечание. Количество чисел должно совпадать с количеством слов исходного текста.

```
# put your python code here
s=input()
s_=s.split()
#print(s_)
for i in range(0,len(s_)):
    #print(s_[i])
    res=s_[:i+1]
    print(res.count(s_[i]),end=" ")
```

Scrabble game В игре Scrabble каждая буква приносит определенное количество баллов. Общая стоимость слова – сумма баллов его букв. Чем реже буква встречается, тем больше она ценится:

Баллы Буква <https://stepik.org/lesson/492141/step/3?unit=483447>

```
# put your python code here
s={1: ['A', 'E', 'I', 'L', 'N', 'O', 'R', 'S', 'T', 'U'], 2: ['D', 'G'], 3: ['B', 'C', 'M', 'P'], 4: ['F', 'H', 'V', 'W', 'Y'], 5: ["k"], 8: ["J", "X"], 10: ["Q", "Z"]}
str_=input()
counter=0
for key,value in s.items():
    for i in str_:
        if i in value:
            counter=counter+key
print(counter)
```

Строка запроса Строка запроса (query string) — часть URL адреса, содержащая ключи и их значения. Она начинается после вопросительного знака и идет до конца адреса. Например:

<https://beegeek.ru?name=timur> # строка запроса: name=timur Если параметров в строке запроса несколько, то они отделяются символом амперсанда &:

<https://beegeek.ru?name=timur&color=green> # строка запроса: name=timur&color=green
Напишите функцию build_query_string(), которая принимает на вход словарь с параметрами и возвращает строку запроса, сформированную из этих параметров.

Примечание 1. В итоговой строке параметры должны быть отсортированы в лексикографическом порядке ключей словаря.

Примечание 2. Следующий программный код:

```
print(build_query_string({'name': 'timur', 'age': 28})) print(build_query_string({'sport': 'hockey', 'game': 2, 'time': 17}))
```

должен выводить:

age=28&name=timur game=2&sport=hockey&time=17 Примечание 3. Вызывать функцию build_query_string() не нужно, требуется только реализовать.

```
def build_query_string(params):
    result = ''
```



```

for elem in sorted(params):
    result += elem
    result += '='
    result += str(params[elem])
    result += '&'
return result[:-1]

```

Слияние словарей Напишите функцию merge(), объединяющую словари в один общий. Функция должна принимать список словарей и возвращать словарь, каждый ключ которого содержит множество (тип данных set) уникальных значений собранных из всех словарей переданного списка.

Примечание 1. Следующий программный код:

result = merge([{'a': 1, 'b': 2}, {'b': 10, 'c': 100}, {'a': 1, 'b': 17, 'c': 50}, {'a': 5, 'd': 777}]) создает словарь:

result = {'a': {1, 5}, 'b': {2, 10, 17}, 'c': {50, 100}, 'd': {777}}

Примечание 2. Вызывать функцию merge() не нужно, требуется только реализовать.

Примечание 3. Слияние пустых словарей должно быть пустым словарем.

```

def merge(values):
    result = {}
    for elem in values:
        for i in elem:
            if i in result:
                result[i].add(elem[i])
            else:
                result[i] = set()
                result[i].add(elem[i])

    return result

```

Опасный вирус 🐼 В файловую систему компьютера, на котором развернута наша ♥ платформа Stepik, проник опасный вирус и сломал контроль прав доступа к файлам. Говорят, вирус написал один из студентов курса Python для начинающих.

Для каждого файла известно, с какими действиями можно к нему обращаться:

запись W (write, доступ на запись в файл); чтение R (read, доступ на чтение из файла); запуск X (execute, запуск на исполнение файла). Напишите программу для восстановления контроля прав доступа к файлам. Ваша программа для каждого запроса должна будет возвращать ОК если выполняется допустимая операция, и Access denied, если операция недопустима.

Формат входных данных Программа получает на вход количество файлов n , содержащихся в файловой системе компьютера. Далее идет n строк, на каждой имя файла и допустимые с ним операции, разделенные символом пробела. В следующей строке записано число m — количество запросов к файлам, и затем m строк с запросами

вида операция файл. Одному и тому же файлу может быть адресовано любое количество запросов.

Формат выходных данных Программа должна вывести для каждого из n запросов в отдельной строке Access denied или OK.

```
s = {'write': 'W', 'read': 'R', 'execute': 'X'}
d = {}
n = int(input())
for i in range(n):
    arr = input().split()
    d[arr[0]] = arr[1:]
m = int(input())
for i in range(m):
    arr = input().split()
    if s[arr[0]] in d[arr[1]]:
        print('OK')
    else:
        print('Access denied')

transform = {'execute': 'X', 'write': 'W', 'read': 'R'}
mydict = {}

for _ in range(int(input())):
    name, *operations = input().split()
    mydict[name] = operations

for _ in range(int(input())):
    operation, name = input().split()
    if transform[operation] in mydict[name]:
        print('OK')
    else:
        print('Access denied')
```

Покупки в интернет-магазине Напишите программу для подсчета количества единиц каждого вида товара из приобретенных каждым покупателем интернет-магазина.

Формат входных данных На вход программе подается число n — количество строк в базе данных о продажах интернет-магазина. Далее следует n строк с записями вида покупатель товар количество, где покупатель — имя покупателя (строка без пробелов), товар — название товара (строка без пробелов), количество — количество приобретенных единиц товара (натуральное число).

Формат выходных данных Программа должна вывести список всех покупателей в лексикографическом порядке, после имени каждого покупателя — двоеточие, затем список названий всех приобретенных им товаров в лексикографическом порядке, после названия каждого товара — количество единиц товара. Информация о каждом товаре выводится на отдельной строке.

Примечание. Обратите внимание на второй тест. Если позиции товаров повторяются, то в итоговый список попадает суммарное количество товара по данной позиции.

```

baza={}
n=int(input())
for i in range(n):
    #s="naser potato 10"
    #s_=s.split()
    s=input()
    s_=s.split()
    baza.setdefault(s_[0],{})
    baza[s_[0]][s_[1]]=baza[s_[0]].get(s_[1],0)+int(s_[2])
for i in sorted(baza):
    print(i,":",sep="")
    for j,d in sorted(baza[i].items()):
        print(j,d)

```

Тема урока: модуль random Случайные числа Псевдослучайные числа Модуль random
 Аннотация. Урок посвящен модулю random, который содержит функции по работе с псевдослучайными числами.

Случайные числа Случайные числа – последовательность чисел, в которой невозможно предсказать следующее число, зная все предыдущие.

Случайные числа широко используются в различных задачах программирования:

в играх (имитация подбрасывания игрального кубика и другие подобные ситуации); в программах имитационного моделирования; в статистических программах, случайным образом отбирающих данные для анализа; в компьютерной безопасности для шифрования уязвимых данных. Для создания истинно случайных чисел можно бросать монету, игральные кости, или измерять какой-нибудь шумовой сигнал.

Псевдослучайные числа Ставить сложные электронные приборы на каждый компьютер для генерации истинно случайных чисел дорого, поэтому математики и программисты создали алгоритмы получения псевдослучайных чисел.

Для обычного человека псевдослучайные числа практически не отличаются от случайных, однако они вычисляются по математической формуле, и зная первое число и формулу, можно определить любое следующее.



Python предлагает встроенные функции для работы с псевдослучайными числами. Эти функции хранятся в модуле random в стандартной библиотеке.

Модуль random Модуль random предоставляет функции для генерации псевдослучайных чисел, букв и случайного выбора элементов последовательности (списка, строки и т.д.).

Для использования этих функций в начале программы необходимо подключить модуль, что делается командой import:

import random После подключения модуля мы можем использовать его функции.

Функция randint()

Функция randint() принимает два обязательных аргумента a и b и возвращает псевдослучайное целое число из отрезка [ ; ] [a;b].

Следующий код выводит два псевдослучайных целых числа: num1 из отрезка [0 ; 17] [0;17] и num2 из отрезка [- 5 ; 5] [-5;5].

```
import random
```

```
num1 = random.randint(0, 17) num2 = random.randint(-5, 5)
```

 Левая и правая граница a и b включаются в диапазон генерируемых псевдослучайных чисел. Результатом вызова функции random.randint(2, 9) может быть любое число от 2 до 9, включая 2 и 9

Следующий код выводит 10 псевдослучайных целых чисел из диапазона [1 ; 100] [1;100]:

```
import random
```

```
for _ in range(10): print(random.randint(1, 100))
```

 Среди этих чисел возможны повторения.

Функция randrange() Если вы помните, как применять функцию range(), то почувствуете себя непринужденно с функцией randrange(). Функция randrange() принимает такие же аргументы, что и функция range(). Различие состоит в том, что функция randrange() не возвращает саму последовательность чисел. Вместо этого она возвращает случайно выбранное число из последовательности чисел.

Следующий код присваивает переменной num псевдослучайное число в диапазоне от 0 до 9:

```
import random
```

```
num = random.randrange(10)
```

 Аргумент 10 задает конечный предел последовательности значений. Функция возвратит случайно выбранное число из последовательности чисел от 0 до конечного предела, исключая сам предел.

Следующий код задает начальное значение и конечный предел последовательности:

```
import random
```

```
num = random.randrange(5, 10)
```

 Таким образом в переменной num будет храниться псевдослучайное число в диапазоне от 5 до 9

Следующий код задает начальное значение, конечный предел и величину шага:

```
import random
```

```
num = random.randrange(0, 101, 10)
```

 Таким образом в переменной num будет храниться псевдослучайное число из последовательности чисел: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

Функция random() Функции randint() и randrange() возвращают псевдослучайное целое число. А вот функция random() возвращает псевдослучайное число с плавающей точкой (вещественное число). В функцию random() никаких аргументов не передается. Функция random() возвращает случайное число с плавающей точкой в диапазоне от 0.0 до 1.0 (исключая 1.0).

Следующий код выводит случайное число с плавающей точкой из диапазона [0.0 ; 1.0) [0.0;1.0):

```
import random
```

`num = random.random() print(num)` Функция `uniform()` Функция `uniform()` тоже возвращает случайное число с плавающей точкой, но при этом она позволяет задавать диапазон для отбора значений.

Следующий код выводит псевдослучайное число с плавающей точкой из диапазона [1.5 ; 17.3] [1.5;17.3] (включительно):

```
import random
```

`num = random.uniform(1.5, 17.3) print(num)` Функция `seed()` Как уже было сказано псевдослучайные числа вычисляются на основе некой формулы. Генерация случайных чисел иницируется начальным значением. Оно используется в вычислении, возвращающем следующее случайное число в ряду. Когда модуль `random` импортируется, он получает системное время из внутреннего генератора тактовых импульсов компьютера и использует его как начальное значение. Системное время – целое число, представляющее собой текущую дату и время вплоть до сотой секунды. Если бы всегда использовалось одно и то же начальное значение, функции генерации случайных чисел всегда возвращали бы один и тот же ряд псевдослучайных чисел. Поскольку системное время меняется каждую сотую долю секунды, можно утверждать, что всякий раз, когда импортируется модуль `random`, будет создана отличающаяся от предыдущих последовательность случайных чисел.

Вместе с тем, некоторые программы требуют генерации одной и той же последовательности случайных чисел. Для этого можно вызвать функцию `seed()`, задав начальное значение.

Следующий код генерирует 10 псевдослучайных чисел, и при этом содержит инструкцию, явно устанавливающую начальное значение для генератора случайных чисел:

```
import random
```

```
random.seed(17) # явно устанавливаем начальное значение для генератора случайных чисел
```

```
for _ in range(10): print(random.randint(1, 100))
```

 Результат выполнения такого кода:


67 54 39 47 38 23 99 91 91 70 Если выполнить код еще раз, получим ту же самую последовательность псевдослучайных чисел.

Примечания Примечание 1. Подключение модуля следующим образом:

```
from random import *
```

 позволяет в дальнейшем не писать название модуля и символ точки при вызове функций модуля.

Примечание 2. Функции модуля `random` на самом деле являются методами одноименного класса `random`.

Примечание 3. В Python для генерации псевдослучайных чисел используется один из самых совершенных алгоритмов генерации псевдослучайных чисел – "вихрь Мерсенна", разработанный в 1997 году. Реализация выполнена на языке  C, является быстрой и потокобезопасной.

Примечание 4. Настоящие случайные числа можно получить с сайта. Данный сайт использует атмосферный шум для создания по-настоящему случайных чисел.

Примечание 5. Пусть r – случайное число из интервала $(0; 1)$ $(0;1)$. Для того, чтобы перевести такое число в интервал $(a; b)$ $(a;b)$ можно воспользоваться формулой

$$(b - a) \cdot r + a$$

Напишите программу, которая с помощью модуля random моделирует броски монеты. Программа принимает на вход количество попыток и выводит результаты бросков: Орел или Решка (каждое на отдельной строке).

Примечание. Например, при $n = 7$ $n=7$ ваша программа может вывести:

Орел Решка Решка Орел Орел Орел Решка

```
import random

n = int(input()) # количество попыток
for i in range(n):
    s = random.randint(1, 2)
    if s == 1:
        print("Орел")
    else:
        print("Решка")
```

Напишите программу, которая с помощью модуля random моделирует броски игрального кубика с 6 6 гранями. Программа принимает на вход количество попыток и выводит результаты бросков — выпавшее число, которое написано на грани кубика (каждое на отдельной строке).

Примечание. Например, при $n = 7$ $n=7$ ваша программа может вывести:

5 5 6 6 2 6

```
import random

n = int(input()) # количество попыток
for i in range(n):
    s = random.randint(1, 6)
    print(s)
```

Напишите программу, которая с помощью модуля random генерирует случайный пароль. Программа принимает на вход длину пароля и выводит случайный пароль, содержащий только символы английского алфавита a..z, A..Z (в нижнем и верхнем регистре).

Примечание 1. Символам A..Z английского языка соответствуют номера с 65 65 по 90 90 в таблице символов ASCII.

Примечание 2. Символам a..z английского языка соответствуют номера с 97 97 по 122 122 в таблице символов ASCII.

Примечание 3. Используйте функцию chr() для получения символа по его номеру в таблице символов ASCII.

Примечание 4. Например, при длине пароля, равной 15 15 символам, ваша программа может выводить:

```
import random

length = int(input())    # длина пароля
for i in range(length):
    s=random.randint(1,2)
    if s== 1 :
        d=random.randint(65,90)
    if s == 2:
        d=random.randint(97,122)
    print(chr(d),end="")
```

import random

length = int(input()) # длина пароля for i in range(length): s=random.randint(1,2) if s== 1 :
d=random.randint(65,90) if s == 2: d=random.randint(97,122) print(chr(d),end="")

```
import random
res = []
for i in range(7):
    s = random.randint(1,49)
    res.append(s)
res = sorted(res)
print(*res)
```

Тема урока: модуль random Метод shuffle() Метод choice() Метод sample() Модуль string
Аннотация. Урок посвящен модулю random, в частности, методам работы с последовательностями.

Метод shuffle() Метод shuffle() принимает список в качестве обязательного аргумента и перемешивает его случайным образом.

Следующий код перемешивает список numbers случайным образом, а затем выводит его содержимое:

import random

numbers = [1, 2, 3, 4, 5, 6, 7, 8] random.shuffle(numbers) print(numbers) Результатом работы такого кода может быть:

[4, 7, 8, 1, 2, 3, 6, 5] Метод choice() Метод choice() принимает список (строку, кортеж) в качестве обязательного аргумента и возвращает один случайный элемент.

Следующий код выводит по одному случайному элементу из строки 'BEEGEEK', списка [1, 2, 3, 4] и кортежа ('a', 'b', 'c', 'd'):

import random

```
print(random.choice('BEEGEEK')) print(random.choice([1, 2, 3, 4])) print(random.choice(('a', 'b', 'c', 'd')))
```

 Результатом работы такого кода может быть:

Е 3 с Метод `sample()` Метод `sample()` принимает два обязательных аргумента: первый – список (строка, кортеж, множество), второй – количество случайных элементов. Возвращает список из указанного количества уникальных (имеющих разные индексы) случайных элементов.

Результатом работы кода:

```
import random
```

```
numbers = [2, 5, 8, 9, 12]
```

```
print(random.sample(numbers, 1)) print(random.sample(numbers, 2))  
print(random.sample(numbers, 3)) print(random.sample(numbers, 5))
```

 может быть:

[9] [12, 5] [9, 2, 8] [12, 8, 9, 5, 2] Количество случайных элементов не должно превышать длину исходного списка (строки). Следующий код:

```
import random
```

```
numbers = [2, 5, 8, 9, 12]
```

```
print(random.sample(numbers, 6))
```

 приведет к ошибке:

`ValueError: Sample larger than population or is negative` Модуль `string` Встроенный модуль `string` раньше использовался для расширения стандартных возможностей (функционала) строкового типа данных `str`. На текущий момент все функции из модуля `string` переехали в методы строкового типа данных `str`, однако в модуле `string` остались удобные константные строки, которые можно использовать при решении задач.

Приведенный ниже код:

```
import string
```

```
print(string.ascii_letters) print(string.ascii_uppercase) print(string.ascii_lowercase)  
print(string.digits) print(string.hexdigits) print(string.octdigits) print(string.punctuation)  
print(string.printable)
```

 выводит:

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 0123456789
```

```
0123456789abcdefABCDEF 01234567 !"# %&'()*+,-./:;<=>?@[ ]^_`{|}~ \t\n\r\x0b\x0c
```

Примечания Примечание 1. Помимо рассмотренных в уроке методов, модуль `random` содержит много дополнительных методов. Подробнее о модуле `random` можно почитать в документации.

IP адрес состоит из четырех чисел из диапазона от 0 0 до 255 255 (включительно) разделенных точкой.

Напишите функцию `generate_ip()`, которая с помощью модуля `random` генерирует и возвращает случайный корректный IP адрес.

Примечание 1. Пример правильного (неправильного) IP адреса:

192.168.5.250 # правильный 199.300.521.255 # неправильный Примечание 2. Вызывать функцию generate_ip() не нужно, требуется только реализовать.

```
import random
def generate_ip():
    rez=str(random.randint(0, 255))+ '.'+str(random.randint(0, 255))
    + '.'+str(random.randint(0, 255))+ '.'+str(random.randint(0, 255))
    return rez
```

Почтовый индекс в Латверии имеет вид: LetterLetterNumber_NumberLetterLetter, где Letter – заглавная буква английского алфавита, Number – число от 0 0 до 99 99 (включительно).

Напишите функцию generate_index(), которая с помощью модуля random генерирует и возвращает случайный корректный почтовый индекс Латверии.

Примечание 1. Пример правильного (неправильного) индекса Латверии:

AB23_56VG # правильный V3F_231GT # неправильный Примечание 2. Обратите внимание на символ _ в почтовом индексе.

Примечание 3. Вызывать функцию generate_index() не нужно, требуется только реализовать.

```
import random
# Почтовый индекс в Латверии имеет вид:
LetterLetterNumber_NumberLetterLetter,
# где Letter – заглавная буква английского алфавита, Number – число
def generate_index():
    letter_=[chr(random.randint(65,90))for i in range(4)]
    num_=[random.randint(0,99) for i in range(2)]

    ind_=[letter_[0],letter_[1],str(num_[0]),"_",str(num_[1]),letter_[2],letter_[3]]
    d="".join(ind_)
    return d
```

Напишите программу, которая с помощью модуля random перемешивает случайным образом содержимое матрицы (двумерного списка).

Примечание. Выводить содержимое матрицы не нужно

```
import random
matrix = [[1, 2, 3, 4],
          [5, 6, 7, 8],
          [9, 10, 11, 12],
          [13, 14, 15, 16]]
for i in matrix:
    random.shuffle(i)
```

Напишите программу, которая с помощью модуля random генерирует 100 100 случайных номеров лотерейных билетов и выводит их каждый на отдельной строке. Обратите внимание, вы должны придерживаться следующих условий:

номер не может начинаться с нулей; номер лотерейного билета должен состоять из 7 7 цифр; все 100 100 лотерейных билетов должны быть различными.

```
import random
d=set()
while len(d) != 100:
    a=random.randint(1000000,9999999)
    d.add(a)
for i in d:
    print(i)
```

Анаграмма – это слово образованное путём перестановки букв, составляющих другое слово.

Например, слова пила и липа или пост и стоп – анаграммы.

Напишите программу, которая считывает одно слово и выводит с помощью модуля random его случайную анаграмму.

Примечание. Обратите внимание на то, что метод shuffle() работает со списком, а не со строкой.

Для отладки кода []

```
import random
word = list(input())
random.shuffle(word)
print(*word, sep=" ")
```

<https://stepik.org/lesson/356380/step/11?unit=340495>

```
# put your python code here
import random
s={i for i in range(1,75)}
d=[]
d=random.sample(s,26)
matrix=[[0]*5 for i in range(5)]
matrix[1][1]=4
count=0
for i in range (5):
    for j in range (5):
        matrix[i][j]=d[count]
        count=count+1
matrix[2][2]=0
for i in matrix:
    for d in i:
```

```

    print(str(d).ljust(3),end="")
print()

```

```

from random import sample

```

```

numbers = sample(list(range(1, 76)), 25)
bingo = [numbers[i:i + 5] for i in range(0, 21, 5)]
bingo[2][2] = 0

```

```

for i in range(5):
    for j in range(5):
        print(str(bingo[i][j]).ljust(3), end=' ')
    print()

```

Тайный друг Напишите программу, которая случайным образом назначает каждому ученику его тайного друга, который будет вместе с ним решать задачи по программированию.

Формат входных данных На вход программе в первой строке подается число n – общее количество учеников. Далее идут n строк, содержащих имена и фамилии учеников.

Формат выходных данных Программа должна вывести имя и фамилию ученика (в соответствии с исходным порядком) и имя и фамилию его тайного друга, разделённые дефисом.

Примечание 1. Обратите внимание, что нельзя быть тайным другом самому себе и нельзя быть тайным другом для нескольких учеников.

Примечание 2. Приведенные ниже тесты это лишь образцы возможного ответа. Возможны и другие способы выбора тайных друзей.

```

n=int(input())
d=[input() for i in range(n)]
import random
#d=["Светлана Зуева" ," Борис Боков","Аркадий Белых"]
for i in range(len(d)):
    print(d[i],"-",d[i-1])

```

Генератор паролей 1

Напишите программу, которая с помощью модуля random генерирует

?

n паролей длиной

?

m символов, состоящих из строчных и прописных английских букв и цифр, кроме тех, которые легко перепутать между собой:

«l» (L маленькое);

«I» (i большое);

«1» (цифра);

«o» и «0» (маленькая и большая буквы);

«0» (цифра).

Формат входных данных

На вход программе подаются два числа

0

n и

0

m, каждое на отдельной строке.

Формат выходных данных

Программа должна вывести

0

n паролей длиной

0

m символов в соответствии с условием задачи, каждый на отдельной строке.

Примечание 1. Считать, что числа

0

n и

0

m всегда таковы, что требуемые пароли сгенерировать возможно.

Примечание 2. В каждом пароле обязательно должна присутствовать хотя бы одна цифра и буква в верхнем и нижнем регистре.

Примечание 3. Решение задачи удобно оформить в виде двух вспомогательных функций:

функция generate_password(length) – возвращает случайный пароль длиной length символов;

функция generate_passwords(count, length) – возвращает список, состоящий из count случайных паролей длиной length символов.

Примечание 4. Приведенные ниже тесты – это лишь образцы возможного ответа. Возможны и другие способы генерации паролей

```
import string
```

```
import random
```

```
def generate_password(length):
```

```
    s=set(string.digits+string.ascii_lowercase+string.ascii_uppercase)-
```

```
    set("Il10o0")
```

```
    d=random.sample(s,length)
```

```
    return "".join(d)
```

```
def generate_passwords(count,length):
```

```
    return [generate_password(length) for i in range(count)]
```

```
n=int(input())
```

```

m=int(input())
print(*generate_passwords(n,m), sep="\n")

from string import *
from random import sample

LETTER = ''.join((set(ascii_letters) | set(digits)) - set('lIlloO0'))

def generate_password(length):
    return ''.join(sample(LETTER, length))

def generate_passwords(count, length):
    return [generate_password(length) for _ in range(count)]

n, m = int(input()), int(input())
print(*generate_passwords(n, m), sep='\n')

```

Генератор паролей 2 Напишите программу, которая с помощью модуля random генерирует \blacklozenge n паролей длиной \blacklozenge m символов, состоящих из строчных и прописных английских букв и цифр, кроме тех, которые легко перепутать между собой:

«l» (L маленькое); «I» (i большое); «1» (цифра); «o» и «O» (большая и маленькая буквы); «0» (цифра). Дополнительное условие: в каждом пароле обязательно должна присутствовать хотя бы одна цифра и как минимум по одной букве в верхнем и нижнем регистре.

Формат входных данных На вход программе подаются два числа \blacklozenge n и \blacklozenge m, каждое на отдельной строке.

Формат выходных данных Программа должна вывести \blacklozenge n паролей длиной \blacklozenge m символов в соответствии с условием задачи, каждый на отдельной строке.

Примечание 1. Считать, что числа \blacklozenge n и \blacklozenge m всегда таковы, что требуемые пароли сгенерировать возможно.

Примечание 2. Решение задачи удобно оформить в виде двух вспомогательных функций:

функция generate_password(length) – возвращает случайный пароль длиной length символов; функция generate_passwords(count, length) – возвращает список, состоящий из count случайных паролей длиной length символов. Примечание 3. Приведенные ниже тесты – это лишь образцы возможного ответа. Возможны и другие способы генерации паролей.

```

import string
import random

def generate_password(length):
    w=length//3

    #s=set(string.digits+string.ascii_lowercase+string.ascii_uppercase)-
    set("Il0oO")
    st_l=set(string.digits)-set("10")

```

```

    st_2=set(string.ascii_lowercase)-set("lo")
    st_3=set(string.ascii_uppercase)-set("IO")
    d=random.sample(st_1,w)+random.sample(st_2,w)
+random.sample(st_3,length-2*w)
    return "".join(d)
def generate_passwords(count,length):
    return [generate_password(length) for i in range(count)]

n=int(input())
m=int(input())
print(*generate_passwords(n,m),sep="\n")

import string
from random import choice, shuffle

chars1 = [c for c in string.ascii_uppercase if c not in 'OI']
chars2 = [c for c in string.ascii_lowercase if c not in 'ol']
chars3 = list(string.digits[2:])
chars = chars1 + chars2 + chars3

def generate_password(length):
    result = [choice(i) for i in (chars1, chars2, chars3)] +
[choice(chars) for _ in range(3, length)]
    shuffle(result)
    return ''.join(result)

def generate_passwords(count, length):
    result = set()
    while len(result) < count:
        result.add(generate_password(length))
    return list(result)

for i in generate_passwords(int(input()), int(input())):
    print(i)

a="asd # fdf&gdggd . ttet !"
d='!@#$$%^&*()_+'
for x in a:
    if x in d:
        a=a.replace(x, "")
print(a)

asd fdfgdggd . ttet

a= 'все вхождения.<?!@#$$%^&*() подстроки  заменены на подстроку'

for x in a:
    if x in d:
        a=a.replace('.<?#$$%^&*()', "")

```

```

print(a)

все вхождения.<?!@#$$%^&*() подстроки  заменены на подстроку
myset = {'Yellow', 'Orange', 'Black'}
myset.update(['Blue', 'Green', 'Red', 'Orange'])
print(myset)

{'Black', 'Orange', 'Red', 'Green', 'Yellow', 'Blue'}

set1
{'Black', 'Yellow'}

set1 = {'p', 'a', 't', 'f'}
set2 = {'a', 't', 'f'}

print(set1 - set2)

{'p'}

pokaz=input().split()
set_pokaz=set(pokaz)
print(pokaz)
print(set_pokaz)
print(len(pokaz)-len(set_pokaz))

10 20 30 10
['10', '20', '30', '10']
{'10', '20', '30'}
1

```

Города Тимур и Руслан играют в игру города. Они очень любят эту игру и знают много городов, особенно Тимур, однако к концу игры ввиду своего возраста забывают, какие города уже называли.

Напишите программу, считывающую информацию об игре и сообщающую ребятам, что очередной город назван повторно.

Формат входных данных На вход программе в первой строке подаётся натуральное число n – количество названных городов, в последующих n строках вводятся названные города и ещё одна строка с новым, только что названным городом.

```

n=int(input())
citys=[input() for i in range(n+1)]
if len(citys) == len(set(citys)):
    print("OK")
else:
    print("REPEAT")

```

```

m,n = int(input()),int(input())
ruslans={input() for i in range(m)}
for i in range(n):
    if input() in ruslans:
        print("YES")
    else:
        print("NO")

3
1
asd
zxc
qwe
zxc
YES

shc={int(i) for i in input().split() }
kandidat={int(i) for i in input().split() }
if shc == kandidat:
    print("YES")
else:
    print("NO")

22 55 9 8
2 5 22 9 8
NO

m = int(input())
n = int(input())
d={}
stud_list_mat = [input() for _ in range(m)]
stud_list_info= [input() for _ in range(n)]
for i in stud_list_mat:
    counter=0
    if i in stud_list_info:
        repeatl=i
        counter=counter+1
        d[i]=counter
if len(d) != 0:
    if m+n-2*len(d) ==0:
        print("NO")
    print(m+n-2*len(d))
elif len(d) == 0:
    print(m+n)

3
3
asd
qwe

```



```

zxc
zxc
asd
qwe
NO
0

kol_yrokov=int(input())
rez=[]
d=[] for i in range(kol_yrokov)]
list_stud=[]
for i in range(kol_yrokov):
    for j in range(int(input())):
        d[i].append(input())
stud_in_vse_yroki=[]
for i in d:
    for j in i:
        stud_in_vse_yroki.append(j)

print(stud_in_vse_yroki)
for i in stud_in_vse_yroki:
    if stud_in_vse_yroki.count(i)==kol_yrokov:
        rez.append(i)

for i in set(rez):
    print(i)

```

```

2
3
zxc
asd
qwe
2
asd
zxc
['zxc', 'asd', 'qwe', 'asd', 'zxc']
asd
zxc

#fraza="asd zxc poi asd poi tgb zxc".split()
fraza="home sweet home sweet".split()
d={}
for i in fraza:
    d[i]=fraza.count(i)
print(d)
print(min(d))
print(min(d, key=d.get))

```

```

{'home': 2, 'sweet': 2}
home
home

#s="home sweet home sweet.".split()
s=input().lower()

repeat_={}
for i in s:
    if i in '!.,:?':
        s=s.replace(i,"")
for i in s.lower().split():
    repeat_[i]=s.count(i)

min_repeat = min(repeat_, key=repeat_.get)

min_repeat_dict={}

for key in repeat_:
    if repeat_[key] == repeat_[min_repeat]:
        min_repeat_dict[key]= repeat_[min_repeat]

print(s)
print(s.lower().split())
print(min_repeat)
print(min_repeat_dict)

print(*min_repeat_dict.keys())

print(min(min_repeat_dict))

```

I bought two books: a new book and an old book. The new book was more expensive than the old book.

i bought two books a new book and an old book the new book was more expensive than the old book

```
['i', 'bought', 'two', 'books', 'a', 'new', 'book', 'and', 'an',
'old', 'book', 'the', 'new', 'book', 'was', 'more', 'expensive',
'than', 'the', 'old', 'book']
```

bought

```
{'bought': 1, 'two': 1, 'books': 1, 'and': 1, 'was': 1, 'more': 1,
'expensive': 1, 'than': 1}
```

bought two books and was more expensive than
and

s=""

```
s="home sweet home sweet.".lower().split()
text=[i.strip(',.;;!') for i in s]
print(text )
```

```

repeat={}
for i in list(text):
    repeat[i]=text.count(i)

min_repeat=min(repeat.values())
print(min_repeat)
res=[]
for i,j in repeat.items():
    if j == min_repeat:
        res.append(i)
print(min(res))

['home', 'sweet', 'home', 'sweet']
2
home

s=input().lower().split()
text=[i.strip(',.?;:;!') for i in s]
print(text )

```

```

repeat={}
for i in list(text):
    repeat[i]=text.count(i)

min_repeat=min(repeat.values())
print(min_repeat)
res=[]
for i,j in repeat.items():
    if j == min_repeat:
        res.append(i)
print(min(res))

home sweet home sweet.
['home', 'sweet', 'home', 'sweet']
2
home

```

```

s="a b c a a d c".split()
repeat=[]
for i in range(len(s)):
    n=0
    if s[i] not in repeat:
        repeat.append(s[i])
    else:
        print(s[i],s[0:i+1].count(s[i]))
        repeat.append(s[i]+"_"+str(s[0:i+1].count(s[i])-1))
print(repeat)

```

```
a 2
a 3
c 2
['a', 'b', 'c', 'a_1', 'a_2', 'd', 'c_1']
```

```
n=int(input())
d={}
s=[input()for i in range(n)]
for i in s:
    print(i)
    e=i.split(":")
    d[e[0].lower()]=e[1]
m=int(input())
zapros=[input().lower() for i in range(m)]
for i in zapros:
    if i in d:
        print(d[i])
    else:
        print("Не найдено")
```

```
3
Змея: язык программирования Python
Баг: от англ. bug – жучок, клоп, ошибка в программе
Конфа: конференция
Змея: язык программирования Python
Баг: от англ. bug – жучок, клоп, ошибка в программе
Конфа: конференция
```

```
2
баг
фя
    от англ. bug – жучок, клоп, ошибка в программе
Не найдено
```

```
s=[input().lower(). for i in range(2)]
print(s)
if sorted(list((s[0]).replace('!? ;;', '').replace(" ", ""))) ==
sorted(list((s[1]).replace('!? ;;', '').replace(" ", ""))):
    print(sorted(list((s[0]).replace('!? ;;', '').replace(" ", ""))))
    print(sorted(list((s[1]).replace('!? ;;', '').replace(" ", ""))))
    print("YES")
else:
    print("NO")
```

```
ertyqw ipou
qwerty! poiou?
['ertyqw ipou', 'qwerty! poiou?']
NO
```

```

s=[input().lower(). for i in range(2)]
print(s)
if sorted(list((s[0]).replace('!? :;', '').replace(" ", ""))) ==
sorted(list((s[1]).replace('!? :;', '').replace(" ", ""))):
    print(sorted(list((s[0]).replace('!? :;', '').replace(" ", ""))))
    print(sorted(list((s[1]).replace('!? :;', '').replace(" ", ""))))
    print("YES")
else:
    print("NO")

```

```

ertyqw ipou
qwerty! poiou?
['ertyqw ipou', 'qwerty! poiou?']
NO

```

```

s=[input().lower()for i in range(2)]
#s=["asd az&?", "asd az"]
s1='!?,.@#$$%^&*()'
for i in range(2):
    for j in s[i]:
        if j in '!?,.@#$$%^&*()':
            s[i]=s[i].replace(j, "")
if list(sorted((s[0]).replace(" ", ""))) ==
list(sorted((s[1]).replace(" ", ""))):
    print("YES")
else:
    print("NO")

print(list(sorted((s[0]).replace(" ", ""))))
print(list(sorted((s[0]).replace(" ", ""))))

```

```

YES
['a', 'a', 'd', 's', 'z']
['a', 'a', 'd', 's', 'z']

```

```

(s[0]).replace('!? :;', '').replace(" ", "")

```

```

{"type": "string"}

```

```

dict_={}
n=int(input())
for i in range(n):
    d=input().split()
    dict_[d[0]]=d[1]
    dict_[d[1]]=d[0]

```

```

#print(dict_)
find_=input()

```

```

print(dict_[find_])

3
Kind Affable
Intellect Mind
Popular Celebrated
{'Kind': 'Affable', 'Affable': 'Kind', 'Intellect': 'Mind', 'Mind':
'Intellect', 'Popular': 'Celebrated', 'Celebrated': 'Popular'}
Mind
Intellect

dict_={}
n=int(input())
for i in range(n):
    d=input().split()
    dict_[d[0]]=d[1:]

m=int(input())
finders=[input() for i in range(m)]
for y in finders:
    for i,j in dict_.items():
        #print(i,j)
        if y in j:
            print(i)

3
asd fg hj kl
qwe rt yu io
zxc vb nm
2
rt
vb
asd ['fg', 'hj', 'kl']
qwe ['rt', 'yu', 'io']
qwe
zxc ['vb', 'nm']
asd ['fg', 'hj', 'kl']
qwe ['rt', 'yu', 'io']
zxc ['vb', 'nm']
zxc

colors = {'c1': 'Red', 'c2': 'Grey', 'c3': None, 'c4': 'Green', 'c5':
'Yellow', 'c6': 'Pink', 'c7': 'Orange', 'c8': None, 'c9': 'White',
'c10': 'Black', 'c11': 'Violet', 'c12': 'Gold', 'c13': None, 'c14':
'Amber', 'c15': 'Azure', 'c16': 'Beige', 'c17': 'Bronze', 'c18': None,
'c19': 'Lilac', 'c20': 'Pearl', 'c21': None, 'c22': 'Sand', 'c23':
None}

result = {i:colors[i] for i in colors.keys() if colors[i] != None }

```

```
result
```

```
{'c1': 'Red',  
'c2': 'Grey',  
'c4': 'Green',  
'c5': 'Yellow',  
'c6': 'Pink',  
'c7': 'Orange',  
'c9': 'White',  
'c10': 'Black',  
'c11': 'Violet',  
'c12': 'Gold',  
'c14': 'Amber',  
'c15': 'Azure',  
'c16': 'Beige',  
'c17': 'Bronze',  
'c19': 'Lilac',  
'c20': 'Pearl',  
'c22': 'Sand'}
```

```
a = '1:men 2:kind 90:number 0:sun 34:book 56:mountain 87:wood 54:car  
3:island 88:power 7:box 17:star 101:ice'  
a=a.split()
```

```
for i in range(len(a)):  
    a[i]=list(str(a[i]).split(":"))  
print(a,sep="\n")
```

```
[['1', 'men'], ['2', 'kind'], ['90', 'number'], ['0', 'sun'], ['34',  
'book'], ['56', 'mountain'], ['87', 'wood'], ['54', 'car'], ['3',  
'island'], ['88', 'power'], ['7', 'box'], ['17', 'star'], ['101',  
'ice']]
```

```
info_tuple = (['name', 'Timur'], ['age', 28], ['job', 'Teacher']) #  
кортеж списков
```

```
info_dict = dict(info_tuple) # создаем словарь на основе кортежа  
списков
```

```
info_dict
```

```
{'name': 'Timur', 'age': 28, 'job': 'Teacher'}
```

```
s=dict(a = 'Timur', s = 28)  
info = dict(asd= 'Timur', age = 28, job = 'Teacher')  
print(info)
```

```
{'asd': 'Timur', 'age': 28, 'job': 'Teacher'}
```

```
s=input().split()  
f={}
```

```

f[s[-1]]=s[:-1]
print(s,f)

123 456 asd
['123', '456', 'asd'] {'asd': ['123', '456']}

f={'zxc': ['123', '678'], 'yhn': ['678'], 'kjhg': ['34']}
n=int(input())
for i in range(n):
    d=input().split()
    if d[-1] in f:
        f[d[-1]]=d[:-1]+f[d[-1]]
    else:
        f[d[-1]]=d[:-1]
print(d,f)
m=int(input())
zapros=[input().split() for i in range(m)]
print(zapros)
for i in zapros:
    if i in f:
        print(*f[i])
    else:
        print("обонент не найден")

888 zxc
222 ddd
999 000 yhn
['999', '000', 'yhn'] {'zxc': ['888', '123', '678'], 'yhn': ['999', '000', '678'], 'kjhg': ['34'], 'ddd': ['222']}

f={}
n=int(input())
for i in range(n):
    d=input().split()
    if d[-1] in f:
        f[d[-1]]=d[:-1]+f[d[-1]]
    else:
        f[d[-1]]=d[:-1]
#print(d,f)
m=int(input())
zapros=[input() for i in range(m)]
#print(zapros)
for i in zapros:
    if i in f:
        print(*f[i])
    else:
        print("обонент не найден")

3
234 asd

```



```

444 asd
777 yhn
2
www
asd
абонент не найден
444 234

n=int(input())
tel_books={}
for i in range(n):
    tel=input()
    tel=tel.split()
    if tel[1] not in tel_books:
        tel_books[tel[1]]=tel[0]
    else:
        tel_books[tel[1]]=tel[0], tel_books[tel[1]]
print(tel_books)

m=int(input())
name_spisok=[]
for i in range(m):
    name=input()
    name_spisok.append(name)
print(name_spisok)

for i in range(len(name_spisok)):
    if name_spisok[i] not in tel_books:
        print('абонент не найден')
    else:
        print(tel_books[name_spisok[i]])

3
123 asd
444 zxc
777 asd
{'asd': ['777', ['123']], 'zxc': ['444']}
2
asd
uio
['asd', 'uio']
['777', ['123']]
абонент не найден

```

Тема урока: модуль decimal Числа с плавающей точкой float Модуль decimal Тип данных Decimal Аннотация. Урок посвящен модулю decimal и типу данных Decimal.

Числовые типы данных В прошлых уроках мы изучили два числовых типа данных, представленных в Python:

int – целое число; float – число с плавающей точкой (аналог вещественного числа в математике). В Python есть три дополнительных числовых типа данных:

Decimal – десятичное число, для выполнения точных расчетов; Fraction – число, представляющее собой обыкновенную дробь, с заданным числителем и знаменателем; Complex – комплексное число. В этом уроке мы изучим числовой тип данных Decimal, аналог типа данных float на случай более точных вычислений.

Тип данных float Рассмотрим программный код:

```
if 0.3 == 0.3: print('YES') else: print('NO')
```

 Результатом выполнения такого кода будет как и полагается YES.

А теперь рассмотрим программный код:

```
num = 0.1 + 0.1 + 0.1
```

```
if num == 0.3: print('YES') else: print('NO')
```

 Вы будете удивлены, но результатом выполнения такого кода будет NO, так как на самом деле в переменной num хранится что-то типа 0.30000000000000004 0.30000000000000004.

Из-за ограничений в сохранении точного значения чисел, даже простейшие математические операции могут выдавать ошибочный результат. Поэтому, чтобы сравнивать два float числа мы должны использовать такой код:

```
num = 0.1 + 0.1 + 0.1 eps = 0.000000001 # точность сравнения
```

```
if abs(num - 0.3) < eps: # число num отличается от числа 0.3 менее чем 0.000000001
    print('YES') else: print('NO')
```

 Такой код выводит, как полагается, значение YES.

Не стоит сравнивать float числа с помощью оператора ==. Для сравнения float чисел нужно использовать указанный выше код.

Тип данных Decimal Тип данных Decimal – это класс из стандартного модуля decimal. Он представляет собой число с плавающей точкой, как и float. Однако, Decimal имеет ряд существенных отличий от float.

Тип Decimal создан, чтобы операции над вещественными числами в компьютере выполнялись как в математике, и равенство 0.1

- 0.1

- 0.1

= 0.3 0.1+0.1+0.1==0.3 было верным.

Точность результатов арифметических действий очень важна для научных вычислений, в сфере финансов и бизнеса. Для таких задач тип данных float не подходит.

В Python тип данных float реализован по стандарту IEEE-754 как число с плавающей точкой двойной точности (64 64 бита) с основанием экспоненты равным 2

1. Реализация таких чисел заложена прямо в железо любого современного процессора. Поэтому float в Python работает как аналогичный тип данных double в таких языках программирования как C#, C++, Java и т.д. И имеет такие же ограничения и «странности». Так как float поддерживается аппаратно, быстродействие при использовании этого типа данных сравнительно велико.

Тип данных Decimal – число с плавающей точкой с основанием экспоненты 10

1. Он реализован по стандарту IBM: General Decimal Arithmetic Specification, в свою очередь основанному на стандартах IEEE.

Тип данных Decimal реализован программно, поэтому он в разы медленнее типа данных float, реализованного аппаратно. Сам тип данных Decimal написан на языке C.

Тип данных Decimal оперирует числами с произвольной – задаваемой программистом, но конечной точностью. По умолчанию точность составляет 28 28 десятичных знаков.

Тип данных Decimal неизменяемый. Операции над ним приводят к созданию новых объектов, при этом старые не меняются.

Еще одно следствие того, что Decimal реализован программно – его можно на ходу настраивать, как угодно программисту. Для этого есть контекст – объект, содержащий настройки для выполнения операций. Операции, выполняемые в контексте, следуют заданным в нем правилам. Для float все правила фиксированы на аппаратном уровне.

Для типа данных Decimal можно настроить:

точность выполнения операций в количестве десятичных знаков; режимы округления; режимы обработки исключительных ситуаций (деление на ноль, переполнение и т. д). Создание Decimal чисел Создать Decimal число можно из обычного целого числа (int), из числа с плавающей точкой (float) или из строки (str).

Приведенный ниже программный код создает Decimal числа на основе целого числа и строки:

```
from decimal import *
```

```
d1 = Decimal(1) d2 = Decimal(567) d3 = Decimal(-93) d4 = Decimal('12345') d5 = Decimal('52.198')
```

```
print(d1, d2, d3, d4, d5, sep='\n') и выводит:
```

1 567 -93 12345 52.198 При создании Decimal чисел из чисел с плавающей точкой (float) возникают проблемы, так как float числа округляются внутри до ближайшего возможного, а Decimal об этом ничего не знает и копирует содержимое float.

Приведенный ниже программный код создает Decimal число на основе числа с плавающей точкой:

```
from decimal import *
```

```
num = Decimal(0.1)
```

```
print(num) и выводит:
```

0.10000000000000000055511151231257827021181583404541015625 Не рекомендуется создавать Decimal числа из float чисел. В Decimal попадет уже неправильно округленное число. Создавать Decimal числа нужно из целых чисел, либо из строк!

Арифметические операции над Decimal числами Тип данных Decimal отлично интегрирован в язык Python. С Decimal числами работают все привычные операции: сложение, вычитание, умножение, деление, возведение в степень.

Приведенный ниже код:

```
from decimal import *  
  
num1 = Decimal('5.2') num2 = Decimal('2.3')  
  
print(num1 + num2) print(num1 - num2) print(num1 * num2) print(num1 / num2) print(num1 // num2) print(num1 ** num2) выводит:
```

7.5 2.9 11.96 2.260869565217391304347826087 2 44.34122533787992500412791298 Можно совершать арифметические операции над Decimal и целыми числами (миксовать Decimal и int), но не рекомендуется смешивать их с float.

Приведенный ниже код:

```
from decimal import *  
  
num = Decimal('5.2')  
  
print(num + 1) print(num - 10) print(num * 2) print(num ** 4) выводит:
```

6.2 -4.8 10.4 731.1616 Математические функции Decimal числа можно передавать как аргументы функциям, ожидающим float. Они будут преобразованы во float. К примеру, модуль math, оперирующий float числами, может работать и с Decimal числами.

Приведенный ниже код:

```
from decimal import * from math import *  
  
num1 = Decimal('1.44') num2 = Decimal('0.523')  
  
print(sqrt(num1)) print(sin(num2)) print(log(num1 + num2)) выводит:
```

1.2 0.4994813555186418 0.6744739152943241 Важно понимать, что результатом работы функции модуля math являются float числа, а не Decimal.

Тип данных Decimal содержит некоторые встроенные математические методы, возвращающие значения Decimal.

Функция	Описание
---------	----------

sqrt()	вычисляет квадратный корень из Decimal числа exp() возвращает e^x для Decimal числа
--------	---

ln()	вычисляет натуральный логарифм (по основанию e)
------	--

е) Decimal числа

`log10()` вычисляет десятичный логарифм (по основанию 10) Decimal числа

Приведенный ниже код:

```
from decimal import *
```

```
num = Decimal('10.0')
```

```
print(num.sqrt()) print(num.exp()) print(num.ln()) print(num.log10())
```

 выводит:

```
3.162277660168379331998893544 22026.46579480671651695790065
```

```
2.302585092994045684017991455
```

 1 Обратите внимание на количество знаков в записи чисел. Их 28, что соответствует точности десятичного числа по умолчанию.

Тип данных `Decimal` также содержит полезный метод `as_tuple()` который возвращает кортеж из 3 элементов:

`sign` – знак числа (0 для положительного числа и 1 для отрицательного числа); `digits` – цифры числа; `exponent` – значение экспоненты (количество цифр после точки, умноженное на -1), Приведенный ниже код:

```
from decimal import *
```

```
num1 = Decimal('-1.4568769017') num2 = Decimal('0.523')
```

```
print(num1.as_tuple()) print(num2.as_tuple())
```

 выводит:

```
DecimalTuple(sign=1, digits=(1, 4, 5, 6, 8, 7, 6, 9, 0, 1, 7), exponent=-10) DecimalTuple(sign=0, digits=(5, 2, 3), exponent=-3)
```

 Приведенный ниже код:

```
from decimal import *
```

```
num = Decimal('-1.4568769017') num_tuple = num.as_tuple()
```

```
print(num_tuple.sign) print(num_tuple.digits) print(num_tuple.exponent)
```

 выводит:

```
1 (1, 4, 5, 6, 8, 7, 6, 9, 0, 1, 7) -10
```

 Работа с контекстом `Decimal` чисел Базовые параметры `Decimal` можно посмотреть в его контексте, выполнив функцию `getcontext()`.

Приведенный ниже код:

```
from decimal import *
```

```
print(getcontext())
```

 выводит:

```
Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[], traps=[InvalidOperation, DivisionByZero, Overflow])
```

 Мы видим здесь, что точность 28 знаков, округление к ближайшему четному, пределы по экспоненте ±

```
999999 ± 999999, capitals
```

 – это про заглавную `E` при печати, включенные ловушки – неправильная операция, деление на ноль, переполнение.

Точность чисел Контекстом в `Decimal` можно управлять, устанавливая свои значения.

Например, чтобы управлять точностью `Decimal`, необходимо изменить параметр контекста

prec (от англ. precision – точность). При этом точность вступает в силу только во время арифметических операций, а не при создании самих чисел.

Приведенный ниже код:

```
from decimal import *  
  
getcontext().prec = 3 # устанавливаем точность в 3 знака  
  
num = Decimal('3.1415')  
  
print(num) print(num * 1) print(num * 2) print(num / 2) выводит:
```

3.1415 3.14 6.28 1.57 Обратите внимание на то, что точность вступает в силу только во время арифметических операций, а не при создании самих чисел.

Округление чисел Округляют числа Decimal с помощью метода `quantize()`. Этот метод в качестве первого аргумента принимает объект Decimal, указывающий на формат округления.

Приведенный ниже код:

```
from decimal import *  
  
getcontext().prec = 4 # устанавливаем точность числа  
  
num = Decimal('3.1415926535')  
  
print(num.quantize(Decimal('1.000'))) # округление до 3 цифр в дробной части  
print(num.quantize(Decimal('1.00'))) # округление до 2 цифр в дробной части  
print(num.quantize(Decimal('1.0'))) # округление до 1 цифр в дробной части выводит:
```

3.142 3.14 3.1 Если точность округления установлена в 2, а формат округления `Decimal('1.00')`, то возникнет ошибка.

Приведенный ниже код:

```
from decimal import *  
  
getcontext().prec = 2 # устанавливаем точность округления  
  
num = Decimal('3.1415926535')  
  
print(num.quantize(Decimal('1.00'))) # округление до 2 цифр в дробной части приводит к  
возникновению ошибки:
```

`decimal.InvalidOperation: [<class 'decimal.InvalidOperation'>]` Чтобы избежать ее, необходимо поменять точность округления на 3 и больше.

Помимо первого параметра, метод `quantize()` принимает в качестве второго параметра стратегию округления:

`ROUND_CEILING` – округление в направлении бесконечности (`Infinity`); `ROUND_FLOOR` – округляет в направлении минус бесконечности (`- Infinity`); `ROUND_DOWN` – округление в направлении нуля; `ROUND_HALF_EVEN` – округление до ближайшего четного числа, число

6.5 6.5 округлится не до 7 7, а до 6 6; ROUND_HALF_DOWN – округление до ближайшего нуля; ROUND_UP – округление от нуля; ROUND_05UP – округление от нуля (если последняя цифра после округления до нуля была бы 0 или 5, в противном случае к нулю).

Приведенный ниже код:

```
from decimal import *  
  
num = Decimal('3.456')  
  
print(num.quantize(Decimal('1.00'), ROUND_CEILING)) print(num.quantize(Decimal('1.00'),  
ROUND_FLOOR))
```

 выводит:

3.46 3.45 Сравнение float и Decimal чисел Выбор между типами данных Decimal и float – поиск компромисса в условиях конкретной задачи.

Если нужно считать очень много (симуляции, физика, графика, игры), имеет смысл отказаться от точности Decimal в пользу скорости и компактности хранения данных float. В бизнесе и финансах считать приходится не так много, но делать это нужно предельно точно, тут имеет смысл посмотреть в сторону Decimal.

Характеристика / тип float Decimal Реализация аппаратная программная Размер 64 64 бит не ограничен Основание экспоненты 2 2 10 10 Скорость ✓ ✗ Настраиваемость □ □ Для финансов и бизнеса ✓ Для симуляций, визуализаций и игр ✓ □ Для высокоточных вычислений ✓ Примечания Примечание 1. Decimal числа можно сравнивать между собой, как обычные числа, причем в отличие от float чисел допускается и точное равенство.

```
from decimal import *  
  
num = Decimal('0.1') if num*3 == Decimal('0.3'): print('YES') else: print('NO')
```

 Примечание 2. Можно сортировать списки с Decimal числами и искать минимум и максимум среди них.

Приведенный ниже код:

```
from decimal import *  
  
s = '1.34 3.45 1.00 0.03 9.25'  
  
numbers = [Decimal(i) for i in s.split()]  
  
maximum = max(numbers) minimum = min(numbers)  
  
numbers.sort()  
  
print(maximum) print(minimum) print(numbers)
```

 выводит:

9.25 0.03 [Decimal('0.03'), Decimal('1.00'), Decimal('1.34'), Decimal('3.45'), Decimal('9.25')]

Примечание 3. Подробнее о типе данных Decimal можно почитать в официальной документации [тут](#), [тут](#) и [тут](#).

Примечание 4. Подробная статья об устройстве float чисел на хабре.

Примечание 5. О стандарте Decimal чисел от IBM можно почитать [тут](#).

Примечание 6. Чтобы не писать каждый раз название типа, можно использовать следующий код:

```
from decimal import Decimal as D
```

```
num1 = D('1.5') + D('3.2') num2 = D('1.4') * D('2.58')
```

```
print(num1) print(num2)
```

```
from decimal import *
```

```
num = Decimal(0.1) + Decimal(0.1) + Decimal(0.1) - Decimal(0.3)
```

```
if num == 0:
```

```
    print('YES')
```

```
else:
```

```
    print('NO')
```

```
print(num, type(num))
```

```
NO
```

```
2.775557561565156540423631668E-17 <class 'decimal.Decimal'>
```

```
from decimal import *
```

```
s = '0.77 4.03 9.06 3.80 7.08 5.88 0.23 4.65 2.79 0.90 4.23 2.15 3.24  
8.57 0.10 8.57 1.49 5.64 3.63 8.36 1.56 6.67 1.46 5.26 4.83 7.23 1.22  
1.02 7.82 9.97 5.40 9.79 9.82 2.78 2.96 0.07 1.72 7.24 7.84 9.23 1.71  
6.24 5.78 5.37 0.03 9.60 8.86 2.73 5.83 6.50'
```

```
f=[i for i in s.split()]
```

```
w=[Decimal(i) for i in s.split()]
```

```
print(f,max(s),max(f),max(w),sep="\n")
```

```
['0.77', '4.03', '9.06', '3.80', '7.08', '5.88', '0.23', '4.65',  
'2.79', '0.90', '4.23', '2.15', '3.24', '8.57', '0.10', '8.57',  
'1.49', '5.64', '3.63', '8.36', '1.56', '6.67', '1.46', '5.26',  
'4.83', '7.23', '1.22', '1.02', '7.82', '9.97', '5.40', '9.79',  
'9.82', '2.78', '2.96', '0.07', '1.72', '7.24', '7.84', '9.23',  
'1.71', '6.24', '5.78', '5.37', '0.03', '9.60', '8.86', '2.73',  
'5.83', '6.50']
```

```
9
```

```
9.97
```

```
9.97
```

```
from decimal import Decimal as D
```

```
s = '0.77 4.03 9.06 3.80 7.08 5.88 0.23 4.65 2.79 0.90 4.23 2.15 3.24  
8.57 0.10 8.57 1.49 5.64 3.63 8.36 1.56 6.67 1.46 5.26 4.83 7.23 1.22  
1.02 7.82 9.97 5.40 9.79 9.82 2.78 2.96 0.07 1.72 7.24 7.84 9.23 1.71  
6.24 5.78 5.37 0.03 9.60 8.86 2.73 5.83 6.50'
```

```
a = [D(i) for i in s.split()]
```

```
print(max(a),min(a),max(a) + min(a))
```

```
print(max(s),min(s))
```

```
9.97 0.03 10.00
```

```
9
```


Decimal числа, разделенные символом пробела, хранятся в строковой переменной s. Дополните приведенный код, чтобы он вывел сумму наибольшего и наименьшего Decimal числа.

```
from decimal import *
s = '0.77 4.03 9.06 3.80 7.08 5.88 0.23 4.65 2.79 0.90 4.23 2.15 3.24
8.57 0.10 8.57 1.49 5.64 3.63 8.36 1.56 6.67 1.46 5.26 4.83 7.23 1.22
1.02 7.82 9.97 5.40 9.79 9.82 2.78 2.96 0.07 1.72 7.24 7.84 9.23 1.71
6.24 5.78 5.37 0.03 9.60 8.86 2.73 5.83 6.50'
a = [Decimal(i) for i in s.split()]
print(max(a) + min(a))
```

Decimal числа, разделенные символом пробела, хранятся в строковой переменной s. Дополните приведенный код, чтобы он вывел на первой строке сумму всех чисел, а на второй строке 5 самых больших чисел в порядке убывания, разделенных символом пробела.

Для отладки кода []

```
from decimal import *
s = '9.73 8.84 8.92 9.60 9.32 8.97 8.53 1.26 6.62 9.85 1.85 1.80 0.83
6.75 9.74 9.11 9.14 5.03 5.03 1.34 3.52 8.09 7.89 8.24 8.23 5.22 0.30
2.59 1.25 6.24 2.14 7.54 5.72 2.75 2.32 2.69 9.32 8.11 4.53 0.80 0.08
9.36 5.22 4.08 3.86 5.56 1.43 8.36 6.29 5.13'
f=[Decimal(i) for i in s.split()]
print(sum(f))
print(sorted(f,reverse=True)[:5])

279.12
[Decimal('9.85'), Decimal('9.74'), Decimal('9.73'), Decimal('9.60'),
Decimal('9.36')]
```

Дополните приведенный код, чтобы он вывел сумму наибольшей и наименьшей цифры Decimal числа.

```
from decimal import *
num = Decimal(input())
d=num.as_tuple()
print(d)
d=d.digits
print(max(d)+min(d)*(abs(num)>1))

0.2456
DecimalTuple(sign=0, digits=(2, 4, 5, 6), exponent=-4)
6
```

Математическое выражение На вход программе подается Decimal число
<https://stepik.org/lesson/360941/step/13?unit=345464>

d. Напишите программу, которая вычисляет значение выражения:

```
from decimal import *
num = Decimal(input())
print(num.exp() + num.ln() + num.log10() + num.sqrt())
```

```
1.1
4.189677737079134559844013562
```

Десятичные числа хранятся в списке numbers в виде строк. Дополните приведенный код, чтобы он для каждого десятичного числа вывел его представление в виде обыкновенной дроби в формате:

десятичное число = обыкновенная дробь *Примечание.* Программа должна вывести

6.34 = 317/50 4.08 = 102/25 3.04 = 76/25 ...

Десятичные числа, разделенные символом пробела, хранятся в строковой переменной s. Дополните приведенный код, чтобы он вывел сумму наибольшего и наименьшего числа в виде обыкновенной дроби.

Для отладки кода ☐

```
from posixpath import split
from fractions import Fraction

s = '0.78 4.3 9.6 3.88 7.08 5.88 0.23 4.65 2.79 0.90 4.23 2.15 3.24
8.57 0.10 8.57 1.49 5.64 3.63 8.36 1.56 6.67 1.46 5.26 4.83 7.13 1.22
1.02 7.82 9.97 5.40 9.79 9.82 2.78 2.96 0.07 1.72 7.24 7.84 9.23 1.71
6.24 5.78 5.37 0.03 9.60 8.86 2.73 5.83 6.50 0.123 0.00021'
s_=[Fraction(i) for i in s.split()]
print(max(s_),min(s_),max(s_)+min(s_))
```

```
997/100 21/100000 997021/100000
```



```
from fractions import Fraction
```

```
num1 = Fraction(3, 4)      # 3 - числитель, 4 - знаменатель
num2 = Fraction('0.55')
num3 = Fraction('1/9')
```

```
print(num1, num2, num3, sep='\n')
```

```
3/4
11/20
1/9
```

Сократите дробь Даны два натуральных числа  m и 

n. Напишите программу, которая сокращает дробь   n m и выводит ее.

Формат входных данных На вход программе подается два натуральных числа, числитель и знаменатель дроби, каждое на отдельной строке.

Формат выходных данных Программа должна вывести ответ на задачу.

Тестовые данные Sample Input 1:

3 6 Sample Output 1:

1/2

```
from fractions import Fraction
n=int(input())
m=int(input())
print(Fraction(n,m))

9
99
1/11
```

Операции над дробями Даны две дроби в формате a/b. Напишите программу, которая вычисляет и выводит их сумму, разность, произведение и частное.

Формат входных данных На вход программе подаются две ненулевые дроби, каждая на отдельной строке.

Формат выходных данных Программа должна вывести сумму, разность, произведение и частное двух дробей.

Примечание. Обратите внимание на третий тест: исходные дроби сокращать не нужно, а результат нужно.

Тестовые данные Sample Input 1:

2/3 3/7

```
from fractions import Fraction
n=input()
m=input()
print(f"{n} + {m} = {Fraction(n)+Fraction(m)}")
print(f"{n} - {m} = {Fraction(n)-Fraction(m)}")
print(f"{n} * {m} = {Fraction(n)*Fraction(m)}")
print(f"{n} / {m} = {Fraction(n)/Fraction(m)}")

1/8
4/6
1/8 + 4/6 = 19/24
1/8 - 4/6 = -13/24
1/8 * 4/6 = 1/12
1/8 / 4/6 = 3/16
```

Сумма дробей 1 На вход программе подается натуральное число 

n. Напишите программу, которая вычисляет и выводит рациональное число, равное значению выражения

Формат входных данных На вход программе подается натуральное число 

n.

Формат выходных данных Программа должна вывести ответ на задачу.

Примечание 1. Результирующая дробь должна быть несократимой.

Примечание 2. Подробнее о ряде обратных квадратов можно почитать тут.

```
# put your python code here
from fractions import Fraction
n=int(input())
s=[Fraction(1,i**2) for i in range(1,n+1)]
print(sum(s))

4
205/144
```

Сумма дробей 2 На вход программе подается натуральное число 

n. Напишите программу, которая вычисляет и выводит рациональное число, равное значению выражения

Формат входных данных На вход программе подается натуральное число 

n.

Формат выходных данных Программа должна вывести ответ на задачу.

Примечание 1. Результирующая дробь должна быть несократимой.

Примечание 2. Для вычисления факториала можно использовать функцию factorial из модуля math

```
# put your python code here
from fractions import Fraction
def vic(i):
    q=1
    for i in range(1,i+1):
        q=q*i
    return q

n=int(input())
s=[Fraction(1,vic(i)) for i in range(1,n+1)]
print(sum(s))


3
5/3
```

Юный математик Дима учится в седьмом классе и сейчас они проходят обыкновенные дроби с натуральными числителем и знаменателем. Вчера на уроке математики Дима

узнал, что дробь называется правильной, если ее числитель меньше знаменателя, и несократимой, если нет равной ей дроби с меньшими натуральными числителем и знаменателем.

Дима очень любит математику, поэтому дома он долго экспериментировал, придумывая и решая разные задачки с правильными несократимыми дробями. Одну из этих задач Дима предлагает решить вам с помощью компьютера.

На вход программе подается натуральное число 

n. Напишите программу, которая находит наибольшую правильную несократимую дробь с суммой числителя и знаменателя равной 

n.

Формат входных данных На вход программе подается натуральное число 

n.

Формат выходных данных Программа должна вывести ответ на задачу.

Примечание. Возможно вам потребуется функция gcd(), которая позволяет находить наибольший общий делитель (НОД) двух чисел. Функция реализована в модуле math.

Тестовые данные  Sample Input 1:

10 Sample Output 1:

3/7


```
from fractions import Fraction
from math import gcd

n = int(input())

for i in range((n-1)//2, 0, -1):
    if gcd(i, n-i) == 1:
        print(Fraction(i, n-i))
        break
```

```
10
(4, 6)
```

Упорядоченные дроби На вход программе подается натуральное число 

n. Напишите программу, которая выводит в порядке возрастания все несократимые дроби, заключённые между 0 и 1, знаменатель которых не превосходит 

n.

Формат входных данных На вход программе подается натуральное число  , 

1 n, n > 1.

Формат выходных данных Программа должна вывести ответ на задачу.

Примечание. Возможно вам потребуется функция `gcd()`, которая позволяет находить наибольший общий делитель (НОД) двух чисел. Функция реализована в модуле `math`.

Тестовые данные ☐ Sample Input 1:

5 Sample Output 1:

1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5

```
from math import gcd
from fractions import Fraction
n = int(input())
res = []
while n != 1:
    for i in range(1, n):
        if gcd(i, n) == 1:
            res.append(f'{i}/{n}')
    n -= 1
answer = sorted(map(Fraction, res))
print(*answer, sep='\n')

z1, z2 = complex(input()), complex(input())
print(z1, '+', z2, '=', z1 + z2)
print(z1, '-', z2, '=', z1 - z2)
print(z1, '*', z2, '=', z1 * z2)
```

Функции

Тема урока: необязательные и именованные аргументы Позиционные аргументы
Необязательные аргументы Именованные аргументы Аннотация. Урок посвящен необязательным и именованным аргументам.

Позиционные аргументы Все ранее написанные нами функции имели позиционные аргументы. Такие аргументы передаются без указания имен. Они называются позиционными, потому что именно по позиции, расположению аргумента, функция понимает, какому параметру он соответствует.

Рассмотрим следующий код:

```
def diff(x, y): return x - y
```

`res = diff(10, 3)` # используем позиционные аргументы `print(res)` Такой код выведет число 7

1. При вызове функции `diff()` первому параметру `x` будет соответствовать первый переданный аргумент, 10, а второму параметру `y` — второй аргумент, 3

В Python можно использовать не только позиционные, но и именованные аргументы.

Именованные аргументы Аргументы, передаваемые с именами, называются именованными. При вызове функции можно использовать имена параметров из ее определения. Исключение составляют списки аргументов неопределенной длины, где используются аргументы со звездочкой, но об этом в следующем уроке. Все функции из предыдущих уроков можно вызывать, передавая им именованные аргументы.

Рассмотрим следующий код:

```
def diff(x, y): return x - y
```

```
res = diff(x=10, y=3) # используем именованные аргументы print(res) Такой код по-прежнему выведет число 7
```

1. При вызове функции `diff()` мы явно указываем, что параметру `x` соответствует аргумент 10, а параметру `y` — аргумент 3

Использование именованных аргументов позволяет нарушать их позиционный порядок при вызове функции. Порядок упоминания именованных аргументов не имеет значения!

Мы можем вызвать функцию `diff()` так:

```
res = diff(y=3, x=10) и получить тот же результат 7
```

Возможность использования именованных аргументов — еще один повод давать параметрам значащие, а не однобуквенные имена.

Когда стоит применять именованные аргументы. Каких-то строгих правил на этот счёт не существует. Однако широко практикуется такой подход: если функция принимает больше трёх аргументов, нужно хотя бы часть из них указать по имени. Особенно важно именовать значения аргументов, когда они относятся к одному типу, ведь без имен очень трудно понять, что делает функция с подобным вызовом.

Рассмотрим определение функции `make_circle()` для рисования круга:

```
def make_circle(x, y, radius, line_width, fill): # тело функции Вызвать такую функцию можно так:
```

```
make_circle(200, 300, 17, 2.5, True) Тут непросто понять, какие параметры круга задают числа 200 200, 300 300 или 17
```

Сравните:

```
make_circle(x=200, y=300, radius=17, line_width=2.5, fill=True) Такой код читать значительно проще!
```

В соответствии с PEP 8 при указании значений именованных аргументов при вызове функции знак равенства не окружается пробелами.

Когда значение аргументов очевидно, можно их не именовать. Да, очевидность относительна, но обычно легко понять, что скрывается за значениями при вызове функции `point3d(7, 50, 13)` или `rgb(7, 255, 45)`. В первом случае переданные аргументы — координаты точки в трехмерном пространстве, во втором — значения компонент `red`, `green`, `blue` некоторого цвета. Тут можно ориентироваться только на здравый смысл, жестких правил нет.

Мы уже сталкивались с именованными аргументами, когда вызывали функцию `print()`.

Приведенный ниже код:

```
print('aaaa', 'bbbb', sep='*', end='###') print('cccc', 'dddd', sep='()') print('eeee', 'ffff', sep='123', end='python')
```

использует именованные аргументы `sep` и `end` и выводит:

aaaa*bbbb###cccc()dddd eeee123ffffpython Используя именованные аргументы `sep` и `end` можно не беспокоиться, какой из них указать первым.

Напомним, что значение по умолчанию для `sep` — ' ' (символ пробела), а для `end` — '\n' (символ перевода строки).

Комбинирование позиционных и именованных аргументов Мы можем вызывать функции, используя именованные и позиционные аргументы одновременно. Но позиционные значения должны быть указаны до любых именованных!

Для функции `diff()` код:

```
res = diff(10, y=3)
```

 # используем позиционный и именованный аргумент работает как полагается, при этом параметру `x` соответствует значение 10

Приведенный ниже код:

```
res = diff(x=10, 3)
```

 # используем позиционный и именованный аргумент приводит к возникновению ошибки `SyntaxError: positional argument follows keyword argument`.

Необязательные аргументы Бывает, что какой-то параметр функции часто принимает одно и то же значение. Например, для функции `print()` создатели языка Python установили значения параметров `sep` и `end` равными символу пробела и символу перевода строки, поскольку эти значения используют наиболее часто.

Другим примером служит функция `int()`, преобразующая строку в число. Она принимает два аргумента:

первый аргумент: строка, которую нужно преобразовать в число; второй аргумент: основание системы счисления. Это позволяет ей считывать числа в различных системах счисления.

Приведенный ниже код, преобразует двоичное число 101 101:

```
num = int('101', 2)
```

 # аргумент 2 указывает на то, что число 101 записано в двоичной системе
В переменной `num` будет храниться число 5, так как $10_2 = 5$, $10_2 = 5$.

Но чаще всего эта функция используется для считывания из строки чисел, записанных в десятичной системе счисления. Утомительно каждый раз писать 10 вторым аргументом. В таких ситуациях Python позволяет задавать некоторым параметрам значения по умолчанию. У функции `int()` второй параметр по умолчанию равен 10, и потому можно вызывать эту функцию с одним аргументом. Значение второго подставится автоматически.

Чтобы задать значение параметра по умолчанию, в списке параметров функции достаточно после имени переменной написать знак равенства и нужное значение.

Параметры со значением по умолчанию идут последними, ведь иначе интерпретатор не смог бы понять, какой из аргументов указан, а какой пропущен, и для него нужно использовать значение по умолчанию.

Рассмотрим все то же определение функции `make_circle()`, которая рисует круг:

```
def make_circle(x, y, radius, line_width, fill): # тело функции
```

Поскольку обычно нам нужно рисовать круг с шириной линии, равной 1 с заливкой, то логично установить данные значения в качестве значений по умолчанию:

```
def make_circle(x, y, radius, line_width=1, fill=True): # тело функции
```

Теперь для того, чтобы нарисовать стандартный круг, то есть круг имеющий ширину линии, равную 1 с заливкой, мы вызываем функцию так:

```
make_circle(100, 50, 20) или так:
```

```
make_circle(x=100, y=50, radius=20)
```

Если вам хочется поменять ширину линии и заливку, то вы легко можете это сделать:

```
make_circle(x=100, y=50, radius=20, fill=False) # line_width=1, fill=False
make_circle(x=100, y=50, radius=20, line_width=3) # fill=True, line_width=3
make_circle(x=100, y=50, radius=20, line_width=5, fill=False) # line_width=5, fill=False
```

В соответствии с стандартом PEP 8 и при объявлении аргументов по умолчанию пробел вокруг знака равенства не ставят.

Изменяемые типы в качестве значений по умолчанию При использовании изменяемых типов данных в качестве значения параметра по умолчанию можно столкнуться с неожиданными результатами работы функции.

Рассмотрим определение функции `append()`, где в качестве значения по умолчанию используется изменяемый тип данных (список, тип `list`):

```
def append(element, seq=[]): seq.append(element) return seq
```

Вызывая функцию `append()` следующим образом:

```
print(append(10, [1, 2, 3])) print(append(5, [1])) print(append(1, [])) print(append(3, [4, 5]))
```

получим ожидаемый вывод:

```
[1, 2, 3, 10] [1, 5] [1] [4, 5, 3]
```

А если вызовем функцию `append()` так:

```
print(append(10)) print(append(5)) print(append(1))
```

получим не совсем ожидаемый вывод:

```
[10] [10, 5] [10, 5, 1]
```

Что происходит? Значение по умолчанию для параметра создается единожды при определении функции (обычно при загрузке модуля) и становится атрибутом (свойством) функции. Поэтому, если значение по умолчанию изменяемый объект, то его изменение повлияет на каждый следующий вызов функции.

Чтобы посмотреть значения по умолчанию, можно использовать атрибут **defaults**.

Приведенный ниже код:

```
def append(element, seq=[]): seq.append(element) return seq
```

```
print('Значение по умолчанию', append.__defaults__)
```

выводит:

Значение по умолчанию ([,]) Приведенный ниже код:

```
def append(element, seq=[]): seq.append(element) return seq

print('Значение по умолчанию', append.__defaults__) print(append(10)) print('Значение по умолчанию', append.__defaults__) print(append(5)) print('Значение по умолчанию', append.__defaults__) print(append(1)) print('Значение по умолчанию', append.__defaults__)
```

выводит:

Значение по умолчанию ([,]) [10] Значение по умолчанию ([10,]) [10, 5] Значение по умолчанию ([10, 5,]) [10, 5, 1] Значение по умолчанию ([10, 5, 1,]) Для решения проблемы можно использовать константу None в качестве значения параметра по умолчанию, а в теле функции устанавливать нужное значение:

```
def append(element, seq=None): if seq is None: seq = [] seq.append(element) return seq
```

Вызывая функцию append() следующим образом:

```
print(append(10)) print(append(5)) print(append(1))
```

получим ожидаемый вывод:

[10] [5] [1] Подход, основанный на значении None, общепринятый в Python.

Примечания Примечание 1. При написании функций стоит указывать более важные параметры первыми.

Примечание 2. Именованные аргументы часто используют вместе со значениями по умолчанию.

Примечание 3. Именованные и позиционные аргументы не всегда хорошо ладят друг с другом. При вызове функции позиционные аргументы должны обязательно идти перед именованными аргументами.

Примечание 4. Параметр в программировании — принятый функцией аргумент. Термин «аргумент» подразумевает, что конкретно и какой конкретной функции было передано, а параметр — в каком качестве функция применила это принятое. То есть вызывающий код передает аргумент в параметр, который определен в описании (заголовке) функции.

Parameter → Placeholder (заполнитель принадлежит имени функции и используется в теле функции); Argument → Actual value (фактическое значение, которое передается при вызове функции). Подробнее можно почитать тут.

Примечание 5. Отличная статья на хабре про аргументы функций в Python.

Напишите функцию count_args(), которая принимает произвольное количество аргументов и возвращает количество переданных в нее аргументов.

Примечание 1. Обратите внимание, что функция должна принимать не список, а именно произвольное количество аргументов.

Примечание 2. Следующий программный код:

1. Новый пункт
2. Новый пункт

Напишите функцию `matrix()`, которая создает, заполняет и возвращает матрицу заданного размера. При этом (в зависимости от переданных аргументов) она должна вести себя так:

`matrix()` — возвращает матрицу 1×1 , в которой единственное число равно нулю;
`matrix(n)` — возвращает матрицу $n \times n$, заполненную нулями; `matrix(n, m)` — возвращает матрицу из n строк и m столбцов, заполненную нулями; `matrix(n, m, value)` — возвращает матрицу из n строк и m столбцов, в которой каждый элемент равен числу `value`. При создании функции пользуйтесь аргументами по умолчанию.

Примечание 1. Приведенный ниже код:

```
print(matrix()) # матрица 1 × 1 из 0 print(matrix(3)) # матрица 3 × 3 из 0 print(matrix(2, 5)) #  
матрица 2 × 5 из 0 print(matrix(3, 4, 9)) # матрица 3 × 4 из 9
```

 должен вывести:

```
[[0]] [[0, 0, 0], [0, 0, 0], [0, 0, 0]] [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0]] [[9, 9, 9, 9], [9, 9, 9, 9], [9, 9, 9, 9]]
```

Примечание 2. Вызывать функцию `matrix()` не нужно, требуется только реализовать ее.

```
def matrix(n=1,m=1,value=0):  
    if n != 1 and m==1 :  
        m=n  
    d=[[value]*m for i in range(n)]  
    return d  
print(matrix(3,4,9))  
[[9, 9, 9, 9], [9, 9, 9, 9], [9, 9, 9, 9]]  
  
def count_args(*kward):  
    return len(kward)
```

Тема урока: функции с переменным количеством аргументов

Аргументы `*args` Аргументы `**kwargs` Keyword-only аргументы Аннотация. Урок посвящен функциям с переменным количеством аргументов.

Переменное количество аргументов Вспомним функцию `print()`, которой мы пользуемся почти в каждой задаче.

Приведенный ниже код:

```
print('a')  
print('a', 'b')  
print('a', 'b', 'c')  
print('a', 'b', 'c', 'd')  
  
a  
a b
```

```
a b c
a b c d
```

Функция `print()` принимает столько аргументов, сколько ей передано. Причем функция `print()` делает полезную работу, даже когда вызывается вообще без аргументов. В этом случае она переносит каретку печати на новую строку.

Было бы здорово научиться писать свои собственные функции, способные принимать переменное количество аргументов. Для этого потребуется специальный, совсем не сложный во всех смыслах, синтаксис.

Рассмотрим определение функции `my_func()`:

```
def my_func(*arg):
    print(type(arg))
    print(arg)

my_func()
my_func(1, 2, 3)
my_func('a', 'b')

<class 'tuple'>
()
<class 'tuple'>
(1, 2, 3)
<class 'tuple'>
('a', 'b')
```

В заголовке функции `my_func()` указан всего один параметр `args`, но со звездочкой перед ним. Звездочка в определении функции означает, что переменная (параметр) `args` получит в виде кортежа все аргументы, переданные в функцию при ее вызове от текущей позиции и до конца.

При описании функции можно использовать только один параметр помеченный звездочкой, причем располагаться он должен в конце списка параметров, иначе последующим параметрам не достанется значений.

```
def my_func(*args, num):
    print(args)
    print(num)
print(my_func(3))
```

```
-----
-----
TypeError                                Traceback (most recent call
last)
<ipython-input-17-d224c7915962> in <cell line: 4>()
      2     print(args)
      3     print(num)
----> 4 print(my_func(3))
```

```
TypeError: my_func() missing 1 required keyword-only argument: 'num'
```

не является рабочим, так как параметр со звездочкой указан раньше позиционного num.

```
def my_func(a,*arg):
    print(type(arg))
    print(arg)

my_func(5)
my_func(1, 2, 3)
my_func('a', 'b')

<class 'tuple'>
()
<class 'tuple'>
(2, 3)
<class 'tuple'>
('b',)

def my_func(num, *args):
    print(args)
    print(num)

my_func(17, 'Python', 2, 'C#')

('Python', 2, 'C#')
17
```

Обратите внимание: функция my_func() принимает несколько аргументов, но как минимум один аргумент должен быть передан обязательно. Этот первый аргумент станет значением переменной num, а остальные аргументы сохранятся в переменной args. Подобным образом можно делать любое количество обязательных аргументов.

Параметр args в определении функции пишется после позиционных параметров перед первым параметром со значением по умолчанию.

Передача аргументов в форме списка и кортежа

Иногда хочется сначала сформировать набор аргументов, а потом передать их функции. Тут поможет оператор распаковки коллекций, который также обозначается звездочкой *.

Вспомним, что встроенная функция sum() принимает на вход коллекцию чисел (список, кортеж, и т.д.).

Приведенный ниже код:

```
sum1 = sum([1, 2, 3, 4])      # считаем сумму чисел в списке
sum2 = sum((10, 20, 30, 40)) # считаем сумму чисел в кортеже
```

```
print(sum1, sum2)
10 100
```

Однако функция `sum()` не может принимать переменное количество аргументов.

Приведенный ниже код:

```
sum1 = sum(1, 2, 3, 4)
print(sum1)
-----
-----
TypeError                                Traceback (most recent call
last)
<ipython-input-20-2f7c38a7e045> in <cell line: 1>()
----> 1 sum1 = sum(1, 2, 3, 4)
      2
      3 print(sum1)

TypeError: sum() takes at most 2 arguments (4 given)
```

приводит к возникновению ошибки:

TypeError: sum expected at most 2 arguments, got 4

```
a=sum([1,2,3])
print(a)
6
```

Напишем свою версию функции `sum()`, функцию `my_sum()`, которая принимает переменное количество аргументов и вычисляет их сумму

```
def my_sum(a):
    return sum(a)

print(my_sum([1,2,3]))
6

def my_sum(*arg):
    return sum(arg)

print(my_sum())
print(my_sum(1))
print(my_sum(1, 2))
```

```
print(my_sum(1, 2, 3))
print(my_sum(1, 2, 3, 4))

0
1
3
6
10
```

Мы также можем вызывать нашу функцию `my_sum()`, передавая ей списки или кортежи, предварительно распаковав их.

Приведенный ниже код:

```
print(my_sum(*[1, 2, 3, 4, 5])) # распаковка списка
print(my_sum(*(1, 2, 3)))      # распаковка кортежа

15
6
```

Более того, часть аргументов можно передавать непосредственно и даже коллекции подставлять не только по одной.

Приведенный ниже код:

```
def my_sum(*args):
    return sum(arg)

print(my_sum(1, 2, *[3, 4, 5], *(7, 8, 9), 10))

49
```

По соглашению, параметр, получающий подобный кортеж значений, принято называть `args` (от слова *arguments*). Старайтесь придерживаться этого соглашения.

Получение именованных аргументов в виде словаря

Позиционные аргументы можно получать в виде `*args`, причём произвольное их количество. Такая возможность существует и для именованных аргументов. Только именованные аргументы получаются в виде словаря, что позволяет сохранить имена аргументов в ключах.

Рассмотрим определение функции `my_func()`:

```
def my_func(**kwargs):
    print(type(kwargs))
    print(kwargs)

my_func()
```

```

my_func(a=1, b=2)
my_func(name='Timur', job='Teacher')

<class 'dict'>
{}
<class 'dict'>
{'a': 1, 'b': 2}
<class 'dict'>
{'name': 'Timur', 'job': 'Teacher'}

```

По соглашению параметр, получающий подобный словарь, принято называть kwargs (от словосочетания keyword arguments). Старайтесь придерживаться этого соглашения.

Параметр `**kwargs` пишется в самом конце, после последнего аргумента со значением по умолчанию. При этом функция может содержать и `*args` и `**kwargs` параметры.

Рассмотрим определение функции, которая принимает все виды аргументов.

```

def my_func(a, b, *args, name='Gvido', age=17, **kwargs):
    print(a, b)
    print(args)
    print(name, age)
    print(kwargs)

my_func(1, 2, 3, 4, name='Timur', age=28, job='Teacher',
        language='Python')
my_func(1, 2, name='Timur', age=28, job='Teacher', language='Python')
my_func(1, 2, 3, 4, job='Teacher', language='Python')

1 2
(3, 4)
Timur 28
{'job': 'Teacher', 'language': 'Python'}
1 2
()
Timur 28
{'job': 'Teacher', 'language': 'Python'}
1 2
(3, 4)
Gvido 17
{'job': 'Teacher', 'language': 'Python'}

```

Не нужно пугаться, в реальном коде функции редко используют все эти возможности одновременно. Но понимать, как каждая отдельная форма объявления аргументов работает, и как такие формы сочетать — очень важно.

Для лучшего понимания, поэкспериментируйте с передачей аргументов. Правила использования аргументов довольно сложно описывать, но на практике мы редко сталкиваемся с проблемами.

Передача именованных аргументов в форме словаря

Как и в случае позиционных аргументов, именованные можно передавать в функцию "пачкой" в виде словаря. Для этого нужно перед словарём поставить две звёздочки.

Рассмотрим определение функции `my_func()`:

```
def my_func(**kwargs):
    print(type(kwargs))
    print(kwargs)

info = {'name': 'Timur', 'age': '28', 'job': 'teacher'}

my_func(**info)

<class 'dict'>
{'name': 'Timur', 'age': '28', 'job': 'teacher'}
```

Рассмотрим еще один пример определения функции `print_info()`, распечатающей информацию о пользователе.

```
def print_info(name, surname, age, city, *children,
**additional_info):
    print('Имя:', name)
    print('Фамилия:', surname)
    print('Возраст:', age)
    print('Город проживания:', city)
    if len(children) > 0:
        print('Дети:', ', '.join(children))
    if len(additional_info) > 0:
        print(additional_info)

children = ['Бодхи Рансом Грин', 'Ноа Шэннон Грин', 'Джорни Ривер Грин']
additional_info = {'height': 163, 'job': 'actress'}

print_info('Меган', 'Фокс', 34, 'Ок-Ридж', *children,
**additional_info)

Имя: Меган
Фамилия: Фокс
Возраст: 34
Город проживания: Ок-Ридж
Дети: Бодхи Рансом Грин, Ноа Шэннон Грин, Джорни Ривер Грин
{'height': 163, 'job': 'actress'}
```

При подстановке аргументов "разворачивающиеся" наборы аргументов вроде `*positional` и `**named` можно указывать вперемешку с аргументами соответствующего типа: `*positional` с позиционными, а `**named` — с именованными. И, конечно, же, все именованные аргументы должны идти после всех позиционных!

Keyword-only аргументы

В Python 3 добавили возможность пометить именованные аргументы функции так, чтобы вызвать функцию можно было, только передав эти аргументы по именам. Такие аргументы называются keyword-only и их нельзя передать в функцию в виде позиционных.

Рассмотрим определение функции `make_circle()`:

```
def make_circle(x, y, radius, *, line_width=1, fill=True):
```

Здесь `*` выступает разделителем: отделяет обычные аргументы (их можно указывать по имени и позиционно) от строго именованных.

Приведенный ниже код работает как и полагается:

```
def make_circle(x, y, radius, *, line_width=1, fill=True):  
    pass  
  
make_circle(10, 20, 5) # x=10,  
y=20, radius=5, line_width=1, fill=True  
make_circle(x=10, y=20, radius=7) # x=10,  
y=20, radius=7, line_width=1, fill=True  
make_circle(10, 20, radius=10, line_width=2, fill=False) # x=10,  
y=20, radius=10, line_width=2, fill=False  
make_circle(x=10, y=20, radius=17, line_width=3) # x=10,  
y=20, radius=17, line_width=3, fill=True
```

То есть аргументы `x`, `y` и `radius` могут быть переданы в качестве как позиционных, так и именованных аргументов. При этом аргументы `line_width` и `fill` могут быть переданы только как именованные аргументы.

Приведенный ниже код:

```
make_circle(10, 20, 15, 20)  
make_circle(x=10, y=20, 15, True)  
make_circle(10, 20, 10, 2, False)  
  
File "<ipython-input-40-0f4760bc0136>", line 2  
    make_circle(x=10, y=20, 15, True)  
                                ^  
SyntaxError: positional argument follows keyword argument
```

приводит к возникновению ошибок.

Этот пример неплохо демонстрирует подход к описанию аргументов. Первые три аргумента — координаты центра круга и радиус. Координаты центра и радиус присутствуют у круга всегда, поэтому обязательны и их можно не именовать. А вот `line_width` и `fill` — необязательные аргументы, ещё и получающие ничего не говорящие значения. Вполне логично ожидать, что вызов вида `make_circle(10, 20, 5, 3, False)` мало кому

понравится! Ради ясности аргументы `line_width` и `fill` и объявлены так, что могут быть указаны только явно через имя.

Мы также можем объявить функцию, у которой будут только строго именованные аргументы, для этого нужно поставить звёздочку в самом начале перечня аргументов.

```
def make_circle(*, x, y, radius, line_width=1, fill=True):
```

Теперь для вызова функции `make_circle()` нам нужно передать значения всех аргументов явно через их имя:

```
make_circle(x=10, y=20, radius=15) #  
line_width=1, fill=True  
make_circle(x=10, y=20, radius=15, line_width=4, fill=False)
```

Такой разделитель можно использовать только один раз в определении функции. Его нельзя применять в функциях с неограниченным количеством позиционных аргументов `*args`.

примечания. Примечание 1. Специальный синтаксис `*args` и `**kwargs` в определении функции позволяет передавать функции переменное количество позиционных и именованных аргументов. При этом `args` и `kwargs` просто имена. Вы не обязаны их использовать, можно выбрать любые, однако среди Python программистов приняты именно эти.

Примечание 2. Вы можете использовать одновременно `*args` и `**kwargs` в одной строке для вызова функции. В этом случае порядок имеет значение. Как и аргументы, не являющиеся аргументами по умолчанию, `*args` должны предшествовать и аргументам по умолчанию, и `**kwargs`. Правильный порядок параметров:

позиционные аргументы, `*args` аргументы, `**kwargs` аргументы.

```
def my_func(a, b, *args, **kwargs):
```

Задачи

Напишите функцию `count_args()`, которая принимает произвольное количество аргументов и возвращает количество переданных в нее аргументов.

Примечание 1. Обратите внимание, что функция должна принимать не список, а именно произвольное количество аргументов.

Примечание 2. Следующий программный код:

```
def count_args(*args):  
    return len(args)  
  
print(count_args())  
print(count_args(10))
```

```
print(count_args('stepik', 'beegreek'))
print(count_args([], (''), 'a', 12, False))

0
1
2
5
```

Напишите функцию `sq_sum()`, которая принимает произвольное количество числовых аргументов и возвращает сумму их квадратов.

Примечание 1. Обратите внимание, что функция должна принимать не список, а именно произвольное количество аргументов.

Примечание 2. Следующий программный код:

```
print(sq_sum()) print(sq_sum(2)) print(sq_sum(1.5, 2.5)) print(sq_sum(1, 2, 3)) print(sq_sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
```

```
def sq_sum(*kvarg):
    d=[]
    for i in kvarg:
        d.append(i**2)
    return sum(d)
```

Напишите функцию `mean()`, которая принимает произвольное количество аргументов и возвращает среднее арифметическое переданных в нее числовых (`int` или `float`) аргументов.

Примечание 1. Обратите внимание, что функция должна принимать не список, а именно произвольное количество аргументов.

Примечание 2. Функция должна игнорировать аргументы всех типов, кроме `int` или `float`.

Примечание 3. Следующий программный код:

```
print(mean()) print(mean(7)) print(mean(1.5, True, ['stepik'], 'beegreek', 2.5, (1, 2)))
print(mean(True, ['stepik'], 'beegreek', (1, 2))) print(mean(-1, 2, 3, 10, ('5'))) print(mean(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
```

должен вывести:

0.0 7.0 2.0 0.0 3.5 5.5

Примечание 4. Для проверки типа можно использовать встроенную функцию `type()`.

Примечание 5. Вызывать функцию `mean()` не нужно, требуется только реализовать.

```
def mean(*args):
    d = [i for i in args if type(i) == int or type(i) == float]
    if len(d) == 0:
        return 0.0
    return sum(d) / len(d)
```

Напишите функцию `greet()`, которая принимает произвольное количество аргументов строк имен (как минимум одно) и возвращает приветствие в соответствии с образцом.

Примечание 1. Обратите внимание, что функция должна принимать не список, а именно произвольное количество аргументов.

Примечание 2. Следующий программный код:

```
print(greet('Timur')) print(greet('Timur', 'Roman')) print(greet('Timur', 'Roman', 'Ruslan'))
```

должен выводить:

Hello, Timur! Hello, Timur and Roman! Hello, Timur and Roman and Ruslan! Примечание 3. Функция `greet()` должна принимать как минимум один обязательный аргумент!

Примечание 4. Вызывать функцию `greet()` не нужно, требуется только реализовать.

```
def greet(name,*karg):
    if len(karg) == 0:
        d="Hello,"+' '+name+'!'
        return d
    else:
        s="Hello, "+' and '.join((name,) + karg)+ '!'
        return s

def greet(name, *args):
    names = (name,) + args
    return f'Hello, ' + f' and '.join(names) + f'!'

print(greet('Timur'))
print(greet('Timur', 'Roman'))
print(greet('Timur', 'Roman', 'Ruslan'))
```

```
Hello, Timur!
Hello, Timur and Roman!
Hello, Timur and Roman and Ruslan!
```

Напишите функцию `print_products()`, которая принимает произвольное количество аргументов и выводит список продуктов (любая непустая строка) по образцу: <номер продукта> <название продукта> (нумерация продуктов начинается с единицы). Если среди переданных аргументов нет ни одного продукта, необходимо вывести текст Нет продуктов.

Примечание 1. Обратите внимание, что функция должна принимать не список, а именно произвольное количество аргументов.

Примечание 2. Числа, списки, кортежи, словари, множества и другие нестроковые объекты продуктами не являются и их нужно игнорировать.

Примечание 3. Следующий программный код:

```
print_products('Бананы', [1, 2], ('Stepik',), 'Яблоки', '',  
'Макароны', 5, True)  
должен выводить:
```

- 1) Бананы
- 2) Яблоки
- 3) Макароны

Следующий программный код:

```
print_products([4], {}, 1, 2, {'Beegeek'}, '')  
должен выводить:
```

Нет продуктов

Примечание 4. Обратите внимание: функция `print_products()` должна выводить (печатать) нужное значение, а не возвращать его.

Примечание 5. Вызывать функцию `print_products()` не нужно, требуется только реализовать.

```
def print_products(*prod):  
    d=[i for i in prod if type(i)==str and len(i) !=0 ]  
    if len(d)!=0:  
        for i,j in enumerate(d):  
            print(i+1,")", " ", j, sep="")  
  
    else:  
        return ("Нет продуктов")  
    return ""
```

Напишите функцию `info_kwargs()`, которая принимает произвольное количество именованных аргументов и печатает именованные аргументы в соответствии с образцом: `:`, при этом имена аргументов следуют в алфавитном порядке (по возрастанию).

Примечание 1. Обратите внимание, что функция должна принимать не список, а именно произвольное количество именованных аргументов.

Примечание 2. Следующий программный код:

`info_kwargs(first_name='Timur', last_name='Guev', age=28, job='teacher')` должен выводить:

`age: 28 first_name: Timur job: teacher last_name: Guev` Примечание 3. Вызывать функцию `info_kwargs()` не нужно, требуется только реализовать.

```
def info_kwargs(**arg):  
    for i,j in sorted(arg.items()):  
        print(i,":", " ", j, sep="")  
    return ""
```

Тема урока: парадигмы программирования Парадигмы программирования Императивное программирование Структурное программирование Объектно-ориентированное

программирование Логическое программирование Функциональное программирование
Аннотация. Урок посвящен основным парадигмам программирования.

Парадигмы программирования Парадигма программирования (подход к программированию) — совокупность идей и понятий, определяющих стиль написания компьютерных программ.

Парадигма – устоявшаяся система научных взглядов, в рамках которой ведутся исследования (Т. Кун).

Парадигма программирования определяется:

вычислительной моделью; базовой программной единицей (-ами); методами разделения абстракций. Язык программирования не обязательно использует единственную парадигму. Существуют мультипарадигменные языки. Создатели таких языков считают, что ни одна парадигма не может быть одинаково эффективной для всех задач, и следует позволять программисту выбирать лучшую для решения каждой.

Язык программирования Python – мультипарадигменный.

Основные парадигмы программирования:

императивное программирование; структурное программирование; объектно-ориентированное программирование; функциональное программирование; логическое программирование. Вычислительная техника создавалась для решения математических задач — расчета баллистических траекторий, численного решения уравнений и т.д. Для этого же предназначены первые языки программирования, такие как Fortran, Алгол, реализованные в парадигме императивного программирования.

Императивное программирование Императивное программирование (ИП) характеризуется тем, что:

в исходном коде программы записаны инструкции (команды); инструкции должны выполняться последовательно; данные, полученные при выполнении инструкции, могут записываться в память; данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями. Императивная программа похожа на приказы (англ. imperative — приказ, повелительное наклонение), выражаемые повелительным наклонением в естественных языках. Это последовательность команд, выполняемых процессором.

При императивном подходе к составлению кода широко используется присваивание. Наличие операторов присваивания увеличивает сложность модели вычислений и создает условия для специфических ошибок императивных программ.

Основные механизмы управления:

последовательное исполнение команд; использование именованных переменных; использование оператора присваивания; использование ветвления (оператор if); использование безусловного перехода (оператор goto 😊). Ключевой идеей императивного программирования является работа с переменными, как с временным хранением данных в оперативной памяти.

Структурное программирование Структурная парадигма программирования нацелена на сокращение времени разработки и упрощение поддержки программ за счёт использования блочных операторов и подпрограмм. Отличительная черта структурных программ — отказ от оператора безусловного перехода (goto []), который широко использовался в 1970-х годах.

Основные механизмы управления:

последовательное исполнение команд; использование именованных переменных; использование оператора присваивания; использование ветвления (оператор if); использование циклов; использование подпрограмм (функций). В структурном программировании программа по возможности разбивается на маленькие подпрограммы (функции) с изолированным контекстом.

Парадигму структурного программирования предложил нидерландский ученый Эдсгер Дейкстра.

Объектно-ориентированное программирование В объектно-ориентированной парадигме программа разбивается на объекты – структуры данных, состоящие из полей, описывающих состояние, и методов – функций, применяемых к объектам для изменения или запроса их состояния.

Объектно-ориентированную парадигму программирования поддерживают:

Python; C#; Java; C++; JavaScript; и другие. Основные механизмы управления:

абстракция; класс; объект; полиморфизм; инкапсуляция; наследование. Логическое программирование При использовании логического программирования программа содержит описание проблемы в терминах фактов и логических формул, а решение проблемы система находит с помощью механизмов логического вывода.

В конце 60-х годов XX века Корделл Грин предложил использовать резолюцию как основу логического программирования. Алан Колмеро создал язык логического программирования Prolog в 1971 году. Логическое программирование пережило пик популярности в середине 80-х годов XX века, когда было положено в основу проекта разработки программного и аппаратного обеспечения вычислительных систем пятого поколения.

Важное его преимущество — достаточно высокий уровень машинной независимости, а также возможность откатов, возвращения к предыдущей подцели при отрицательном результате анализа одного из вариантов в процессе поиска решения.

Один из концептуальных недостатков логического подхода — специфичность класса решаемых задач.

Недостаток практического характера — сложность эффективной реализации для принятия решений в реальном времени, скажем, для систем жизнеобеспечения.

Функциональное программирование Основной инструмент функционального программирования (ФП) — математические функции.

Математические функции выражают связь между исходными данными и итогом процесса. Процесс вычисления также имеет вход и выход, поэтому функция — вполне подходящее и

адекватное средство описания вычислений. Именно этот простой принцип положен в основу функциональной парадигмы программирования.

Функциональное программирование (ФП) — декларативная парадигма программирования.

Функциональная программа — набор определений функций. Функции определяются через другие функции или рекурсивно через самих себя. При выполнении программы функции получают аргументы, вычисляют и возвращают результат, при необходимости вычисляя значения других функций.

Как преимущества, так и недостатки данной парадигмы определяет модель вычислений без состояний. Если императивная программа на любом этапе исполнения имеет состояние, то есть совокупность значений всех переменных, и производит побочные эффекты, то чисто функциональная программа ни целиком, ни частями состояния не имеет и побочных эффектов не производит. То, что в императивных языках делается путем присваивания значений переменным, в функциональных достигается передачей выражений в параметры функций. В результате чисто функциональная программа не может изменять имеющиеся данные, а может лишь порождать новые копированием и/или расширением старых. Следствие того же — отказ от циклов в пользу рекурсии.

Сильные стороны функционального программирования:

повышение надёжности кода; удобство организации модульного тестирования; возможности оптимизации при компиляции; возможности параллелизма. Недостатки: отсутствие присваиваний и замена их на порождение новых данных приводят к необходимости постоянного выделения и автоматического освобождения памяти, поэтому в системе исполнения функциональной программы обязательным компонентом становится высокоэффективный сборщик мусора.

Основные идеи функционального программирования:

неизменяемые переменные — в функциональном программировании можно определить переменную, но изменить ее значение нельзя; чистая функция — это функция, результат работы которой предсказуем. При вызове с одними и теми же аргументами, такая функция всегда вернет одно и то же значение. Про такие функции говорят, что они не вызывают побочных эффектов; функции высшего порядка — могут принимать другие функции в качестве аргумента или возвращать их; рекурсия — поддерживается многими языками программирования, а для функционального программирования обязательна. Дело в том, что в языках ФП отсутствуют циклы, поэтому для повторения операций служит рекурсия. Использование рекурсии в языках ФП оптимизировано, и происходит быстрее, чем в языках императивного программирования; лямбда-выражения — способ определения анонимных функциональных объектов. Примечания Примечание 1. Термин «парадигма программирования» впервые применил в 1978 году Роберт Флорд. В своей лекции при получении премии Тьюринга он отметил, что в программировании можно наблюдать явление, подобное парадигмам Куна, но, в отличие от них, парадигмы программирования не взаимоисключающие: если прогресс искусства программирования в целом требует постоянного изобретения и усовершенствования парадигм, то совершенствование искусства программиста требует расширения репертуара парадигм.

Таким образом, по мнению Роберта Флойда, в отличие от парадигм в научном мире, описанных Куном, парадигмы программирования могут сочетаться, обогащая инструментарий программиста.

Примечание 2. В основе императивных, структурных, объектно-ориентированных языков программирования лежит машина Тьюринга, разработанная Аланом Тьюрингом.

Примечание 3. В основе функциональных языков программирования лежит модель лямбда-исчислений, разработанная Алонзо Чёрчем.

Тема урока: функции как объекты

