**Тема урока: введение в списки**

1. Создание списков
2. Пустые списки
3. Встроенная функция list()
4. Вывод списков

**Аннотация.** Списки как сохранение последовательностей и аналог массивов.

**Списки**

В предыдущих уроках мы работали с последовательностями чисел, символов, строк, но не сохраняли всю последовательность в памяти компьютера, а обрабатывали ее поэлементно, считывая раз за разом новый элемент. Однако во многих задачах требуется **сохранять всю последовательность**. Например, классическая задача [сортировки](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8) (упорядочения) некоторой последовательности требует сохранения всех данных в памяти компьютера. Увы, не сохранив, их невозможно отсортировать. И тут на помощь приходит структура данных, которая в большинстве языков программирования называется [массивом](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)). В Python она называется **списком**.

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

Список представляет собой последовательность элементов, пронумерованных от 0, как символы в строке.

**Создание списка**

Чтобы создать список, нужно перечислить его элементы через запятую в квадратных скобках:

numbers = [2, 4, 6, 8, 10]

languages = ['Python', 'C#', 'C++', 'Java']

Список numbers состоит из 5 элементов, и каждый из них — целое число.

* numbers[0] == 2;
* numbers[1] == 4;
* numbers[2] == 6;
* numbers[3] == 8;
* numbers[4] == 10.

Список languages состоит из 4 элементов, каждый из которых — **строка.**

* languages[0] == 'Python';
* languages[1] == 'C#';
* languages[2] == 'C++';
* languages[3] == 'Java'.

    Значения, заключенные в квадратные скобки и отделенные запятыми, называются **элементами списка**.

Список может содержать значения **разных типов данных**:

info = ['Timur', 1992, 61.5]

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

* info[0] == 'Timur';
* info[1] == 1992;
* info[2] == 61.5.

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

**Пустой список**

Создать пустой список можно двумя способами:

1. Использовать пустые квадратные скобки [];
2. Использовать встроенную функцию, которая называется list.

Следующие две строки кода создают пустой список:

mylist = [] # пустой список

mylist = list() # тоже пустой список

**Вывод списка**

Для вывода всего списка можно применить функцию print():

numbers = [2, 4, 6, 8, 10]

languages = ['Python', 'C#', 'C++', 'Java']

print(numbers)

print(languages)

Функция print() выводит на экран элементы списка, в квадратных скобках, разделенные запятыми:

[2, 4, 6, 8, 10]

['Python', 'C#', 'C++', 'Java']

Обратите внимание, что вывод списка содержит квадратные скобки. Позже мы научимся выводить элементы списка в более удобном виде с помощью циклов или с помощью распаковки.

**Встроенная функция list**

Python имеет встроенную функцию list(), которая помимо создания пустого списка может преобразовывать некоторые типы объектов в списки.

Например, мы знаем, что функция range() создает последовательность целых чисел в заданном диапазоне. Для преобразования этой последовательности в список, мы пишем следующий код:

numbers = list(range(5))

Во время исполнения этого кода происходит следующее:

1. Вызывается функция range(), в которую в качестве аргумента передается число 5;
2. Эта функция возвращает последовательность чисел 0, 1, 2, 3, 4;
3. Последовательность чисел 0, 1, 2, 3, 4 передается в качестве аргумента в функцию list();
4. Функция list() возвращает список [0, 1, 2, 3, 4];
5. Список [0, 1, 2, 3, 4] присваивается переменной numbers.

Вот еще один пример:

even\_numbers = list(range(0, 10, 2)) # список содержит четные числа 0, 2, 4, 6, 8

odd\_numbers = list(range(1, 10, 2)) # список содержит нечетные числа 1, 3, 5, 7, 9

Точно так же с помощью функции list() мы можем создать список из символов строки. Для преобразования строки в список мы пишем следующий код:

s = 'abcde'

chars = list(s) # список содержит символы 'a', 'b', 'c', 'd', 'e'

Во время исполнения этого кода происходит следующее:

1. Вызывается функция list(), в которую в качестве аргумента передается строка 'abcde';
2. Функция list() возвращает список ['a', 'b', 'c', 'd', 'e'];
3. Список ['a', 'b', 'c', 'd', 'e'] присваивается переменной chars.

**Примечания**

**Примечание 1.** Как уже было сказано, списки в Python аналогичны массивам в других языках программирования. Однако разница между списками и массивами все же существует. Элементы массива всегда имеют одинаковый тип данных и располагаются в памяти компьютера непрерывным блоком, а элементы списка могут быть разбросаны по памяти как угодно и могут иметь разный тип данных.

**Примечание 2.** Обратите внимание, при выводе содержимого списка с помощью функции print(), все строковые элементы списка обрамляются **одинарными кавычками**. Если требуется осуществить вывод в двойных кавычках, нужно самостоятельно писать код вывода.

## Список чисел

На вход программе подается одно число �*n*. Напишите программу, которая выводит список [1, 2, 3, ..., n].

**Формат входных данных**  
На вход программе подается одно натуральное число.

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

n=int(input())

s=list(range(1,n+1))

print(s)

## Список букв

На вход программе подается одно число �*n*. Напишите программу, которая выводит список, состоящий из �*n* букв английского алфавита ['a', 'b', 'c', ...] в нижнем регистре.

**Формат входных данных**  
На вход программе подается натуральное число �,1≤�≤26*n*,1≤*n*≤26.

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

n=int(input())

abc=list('abcdefghijklmnopqrstuvwxyz')

print(abc[:n])

n = int(input()) s = "" for i in range(n): s += chr(ord("a") + i) print(list(s))

n = int(input()) s = '' for i in range(n): s += chr(97 + i) print(list(s))

## Основы работы со списками

Работа со списками очень сильно напоминает работу со строками, поскольку и списки, и строки содержат отдельные элементы. Однако элементы списка могут иметь произвольный тип, а элементами строк всегда являются символы. Многое из того, что мы делали со строками, доступно и при работе со списками.

### Функция len()

**Длиной списка** называется количество его элементов. Для того, чтобы посчитать длину списка мы используем встроенную функцию len() (от слова length – длина).

Следующий программный код:

numbers = [2, 4, 6, 8, 10]

languages = ['Python', 'C#', 'C++', 'Java']

print(len(numbers)) # выводим длину списка numbers

print(len(languages)) # выводим длину списка languages

print(len(['apple', 'banana', 'cherry'])) # выводим длину списка, состоящего из 3 элементов

выведет:

5

4

3

### Оператор принадлежности in

Оператор in позволяет проверить, содержит ли список некоторый элемент.

Рассмотрим следующий код:

numbers = [2, 4, 6, 8, 10]

if 2 in numbers:

print('Список numbers содержит число 2')

else:

print('Список numbers не содержит число 2')

Такой код проверяет, содержит ли список numbers число 2 и выводит соответствующий текст:

Список 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('Список numbers не содержит нулей')

### Индексация

При работе со строками мы использовали **индексацию**, то есть обращение к конкретному символу строки по его индексу. Аналогично, можно индексировать и списки.

Для индексации списков в Python используются квадратные скобки [], в которых указывается индекс (номер) нужного элемента в списке:

Пусть numbers = [2, 4, 6, 8, 10].

Таблица ниже показывает, как работает индексация:

| **Выражение** | **Результат** | **Пояснение** |
| --- | --- | --- |
| numbers[0] | 2 | первый элемент списка |
| numbers[1] | 4 | второй элемент списка |
| numbers[2] | 6 | третий элемент списка |
| numbers[3] | 8 | четвертый элемент списка |
| numbers[4] | 10 | пятый элемент списка |

    Обратите внимание: первый элемент списка numbers[0], а не numbers[1]. Программисты же всё считают с нуля.

Так же, как и в строках, для нумерации с конца разрешены отрицательные индексы.

| **Выражение** | **Результат** | **Пояснение** |
| --- | --- | --- |
| numbers[-1] | 10 | пятый элемент списка |
| numbers[-2] | 8 | четвертый элемент списка |
| numbers[-3] | 6 | третий элемент списка |
| numbers[-4] | 4 | второй элемент списка |
| numbers[-5] | 2 | первый элемент списка |

Как и в строках, попытка обратиться к элементу списка по несуществующему индексу:

print(numbers[17])

вызовет ошибку:

IndexError: index out of range

### Срезы

Рассмотрим список numbers = [2, 4, 6, 8, 10].

С помощью среза мы можем получить несколько элементов списка, 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**). Разрезая списки, мы создаем новые списки, по сути, подсписки исходного.

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

     Срез numbers[:] возвращает копию исходного списка.

Как и в строках, мы можем использовать отрицательные индексы в срезах списков.

### Использование срезов для изменения элементов в заданном диапазоне

Для изменения целого диапазона элементов списка можно использовать срезы. Например, если мы хотим перевести на русский язык названия фруктов 'banana', 'cherry', 'kiwi', то это можно сделать с помощью среза.

Следующий программный код:

fruits = ['apple', 'apricot', 'banana', 'cherry', 'kiwi', 'lemon', 'mango']

fruits[2:5] = ['банан', 'вишня', 'киви']

print(fruits)

выводит:

['apple', 'apricot', 'банан', 'вишня', 'киви', 'lemon', 'mango']

### Операция конкатенации + и умножения на число \*

Мы можем применять операторы + и \* для списков подобно тому, как мы это делали со строками.

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 списков, состоящих строго из повторяющихся элементов, умножение на число — самый короткий и правильный метод.

Мы также можем использовать расширенные операторы += и \*= при работе со списками.

Следующий программный код:

a = [1, 2, 3, 4]

b = [7, 8]

a += b # добавляем к списку a список b

b \*= 5 # повторяем список b 5 раз

print(a)

print(b)

выводит:

[1, 2, 3, 4, 7, 8]

[7, 8, 7, 8, 7, 8, 7, 8, 7, 8]

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

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

Следующий программный код:

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print('Сумма всех элементов списка =', sum(numbers))

выводит:

Сумма всех элементов списка = 55

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

Следующий программный код:

numbers = [3, 4, 10, 3333, 12, -7, -5, 4]

print('Минимальный элемент =', min(numbers))

print('Максимальный элемент =', max(numbers))

выводит:

Минимальный элемент = -7

Максимальный элемент = 3333

## Отличие списков от строк

Несмотря на всю схожесть списков и строк, есть одно очень важное отличие: строки — **неизменяемые**объекты, а списки – **изменяемые**.

 Следующий программный код:

s = 'abcdefg'

s[1] = 'x' # пытаемся изменить 2 символ (по индексу 1) строки

приводит к ошибке:

object does not support item assignment

Следующий программный код:

numbers = [1, 2, 3, 4, 5, 6, 7]

numbers[1] = 101 # изменяем 2 элемент (по индексу 1) списка

print(numbers)

выводит:

[1, 101, 3, 4, 5, 6, 7]

**Запомни:** изменять отдельные символы строк нельзя, однако можно изменять отдельные элементы списков. Для этого используем индексатор и оператор присваивания.

## Добавление элементов

Мы научились создавать статические списки, то есть списки, элементы которых известны на этапе создания. Следующий шаг – научиться добавлять элементы в уже существующие списки.

### Метод append()

Для добавления нового элемента **в конец списка** используется метод append().

Следующий программный код:

numbers = [1, 1, 2, 3, 5, 8, 13] # создаем список

numbers.append(21) # добавляем число 21 в конец списка

numbers.append(34) # добавляем число 34 в конец списка

print(numbers)

выведет:

[1, 1, 2, 3, 5, 8, 13, 21, 34]

Обратите внимание, для того чтобы использовать метод append(), нужно, чтобы список был создан (при этом он может быть пустым).

Следующий программный код:

numbers = [] # создаем пустой список

numbers.append(1)

numbers.append(2)

numbers.append(3)

print(numbers)

выведет:

[1, 2, 3]

**Важно:** мы не можем использовать индексаторы для установки значений элементов списка, если список пустой. Следующий программный код:

numbers = [] # создаем пустой список

numbers[0] = 1

numbers[1] = 2

numbers[2] = 3

print(numbers)

приводит к ошибке:

IndexError: list assignment index out of range

### Метод extend()

Можно также расширить список другим списком путем вызова метода extend().

Следующий программный код:

numbers = [0, 2, 4, 6, 8, 10]

odds = [1, 3, 5, 7]

numbers.extend(odds)

print(numbers)

выведет:

[0, 2, 4, 6, 8, 10, 1, 3, 5, 7]

Метод extend() как бы расширяет один список, добавляя к нему элементы другого списка.

Отличие между методами append() и extend() проявляется при добавлении строки к списку.

Следующий программный код:

words1 = ['iq option', 'stepik', 'beegeek']

words2 = ['iq option', 'stepik', 'beegeek']

words1.append('python')

words2.extend('python')

print(words1)

print(words2)

выведет:

['iq option', 'stepik', 'beegeek', 'python']

['iq option', 'stepik', 'beegeek', 'p', 'y', 't', 'h', 'o', 'n']

Метод append() добавляет строку 'python' целиком к списку, а метод extend() разбивает строку 'python' на  символы 'p', 'y', 't', 'h', 'o', 'n' и их добавляет в качестве элементов списка.

## Удаление элементов

С помощью оператора del можно удалять элементы списка по определенному индексу.

Следующий программный код:

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

del numbers[5] # удаляем элемент имеющий индекс 5

print(numbers)

выведет:

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

Элемент под указанным индексом удаляется, а список перестраивается.

Обратите внимание на синтаксис удаления, так как он отличается от обычного вызова метода. При удалении элементов не надо передавать аргумент внутри круглых скобок.

Оператор del работает и со срезами: мы можем удалить целый диапазон элементов списка.

Следующий программный код:

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

del numbers[2:7] # удаляем элементы с 2 по 6 включительно

print(numbers)

выведет:

[1, 2, 8, 9]

Мы можем удалить все элементы на четных позициях (0, 2, 4, ...) исходного списка.

Следующий программный код:

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

del numbers[::2]

print(numbers)

 выведет:

[2, 4, 6, 8]

**Все сразу 1 🌶️**

Дополните приведенный код, чтобы он:

1. Вывел длину списка;
2. Вывел последний элемент списка;
3. Вывел список в обратном порядке (вспоминаем срезы);
4. Вывел YES, если список содержит числа 55 и 1717, и NO в противном случае;
5. Вывел список с удаленными первым и последним элементами.

**Примечание.** Каждый вывод осуществлять с новой строки.

numbers = [2, 6, 3, 14, 10, 4, 11, 16, 12, 5, 4, 16, 1, 0, 8, 16, 10, 10, 8, 5, 1, 11, 10, 10, 12, 0, 0, 6, 14, 8, 2, 12, 14, 5, 6, 12, 1, 2, 10, 14, 9, 1, 15, 1, 2, 14, 16, 6, 7, 5]

print(len(numbers))

print(numbers[-1])

print(numbers[::-1])

if 5 and 17 in numbers:

print("YES")

else:

print("NO")

print(numbers[1:-1])

## Алфавит

Напишите программу, выводящую следующий список:

['a', 'bb', 'ccc', 'dddd', 'eeeee', 'ffffff', ...]

**Формат выходных данных**  
Программа должна вывести указанный список.

**Примечание 1.** Последний элемент списка должен состоять из 2626 символов z.

**Примечание 2**. Английский алфавит (для копирования) 😇:

abcdefghijklmnopqrstuvwxyz

s=[]

for i in range(26):

s.append(chr(ord('a')+i)\*(i+1))

print(s)

## Список делителей

На вход программе подается натуральное число �*n*. Напишите программу, которая создает список, состоящий из делителей введенного числа.

**Формат входных данных**  
На вход программе подается натуральное число �*n*.

**Формат выходных данных**  
Программа должна вывести список, состоящий из делителей введенного числа.

n=int(input())

s=[]

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

if n%i==0:

s.append(i)

print(s)

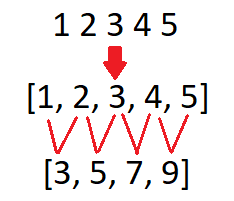
## Суммы двух

На вход программе подается натуральное число �*n*, где �≥2*n*≥2. Затем поступают �*n* целых чисел. Напишите программу, которая создает из указанных чисел список, состоящий из сумм соседних чисел (00 и 11, 11 и 22, 22 и 33 и т.д.).

**Формат входных данных**  
На вход программе подаются натуральное число �*n*, а затем �*n* целых чисел, каждое на отдельной строке.

**Формат выходных данных**  
Программа должна вывести список, состоящий из сумм соседних чисел.

**Примечание.** Для большей ясности разберем 11-й тест:



n=int(input())

s1=[]

s2=[]

for i in range(n):

x=int(input())

s1.append(x)

for i in range(len(s1)-1):

s2.append(s1[i]+s1[i+1])

print(s2)

## Удалите нечетные индексы

На вход программе подается натуральное число �*n*, а затем �*n* целых чисел. Напишите программу, которая создает из указанных чисел список, затем удаляет все элементы стоящие по нечетным индексам, а затем выводит полученный список.

**Формат входных данных**  
На вход программе подаются натуральное число �*n*, а затем �*n* целых чисел, каждое на отдельной строке.

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

**Примечание.** Используйте оператор del.

n=int(input())

s=[]

for i in range(n):

x=int(input())

s.append(x)

del s[1::2]

print(s)

a = [int(i) for i in [input() for \_ in range(int(input()))]] del a[1::2] print(a)

## k-ая буква слова 🌶️🌶️

На вход программе подается натуральное число �*n* и �*n* строк, а затем число �*k*. Напишите программу, которая выводит �*k*-ую букву из введенных строк на одной строке без пробелов.

**Формат входных данных**  
На вход программе подается натуральное число �*n*,  далее �*n* строк, каждая на отдельной строке. В конце вводится натуральное число �*k* – номер буквы (нумерация начинается с единицы).

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

**Примечание.** Если некоторые строки слишком короткие, и в них нет символа с заданным номером, то такие строки при выводе нужно игнорировать.

n=int(input())

s=[]

str=""

for i in range(n):

x=input()

s.append(x)

k=int(input())

for i in range(len(s)):

m=s[i]

if k<=len(m):

y=m[k-1]

str+=y

print(str)

## Символы всех строк

На вход программе подается натуральное число �*n*, а затем �*n* строк. Напишите программу, которая создает список из символов всех строк, а затем выводит его.

**Формат входных данных**  
На вход программе подаются натуральное число �*n*, а затем �*n* строк, каждая на отдельной строке.

**Формат выходных данных**  
Программа должна вывести список состоящий из символов всех введенных строк.

**Примечание.** В результирующем списке могут содержаться одинаковые символы.

n=int(input())

s=[]

for i in range(n):

x=input()

s.extend(x)

print(s)

**Тема урока: вывод элементов списка**

1. Вывод списка с помощью for
2. Вывод списка с помощью распаковки
3. Вывод строки с помощью распаковки

**Аннотация.** Вывод элементов списка и строки.

При выводе содержимого списка с помощью функции print()  вывод элементов осуществляется в квадратных скобках, причем все элементы разделены запятой. Такой вывод не всегда удобен и предпочтителен, поэтому нужно уметь выводить элементы списка нужным нам способом.

**Вывод с помощью цикла for**

Для вывода элементов списка **каждого 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])

Мы передаем в функцию range() длину списка len(numbers). В нашем случае длина списка numbers, равна 11. Таким образом вызов функции range(len(numbers)) имеет вид range(11) и переменная цикла i последовательно перебирает все значения от 0 до 10. Это означает, что выражение numbers[i] последовательно вернет все элементы списка numbers. Такой способ итерации списка удобен, когда нам нужен не только сам элемент numbers[i], но и его индекс 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)

Этот цикл пройдет по списку numbers, придавая переменной цикла num значение каждого элемента списка (!) в отличие от предыдущего цикла, в котором переменная цикла «бегала» по индексам списка.

Если требуется выводить  элементы списка на одной строке, через пробел, то мы можем использовать необязательный параметр end функции print():

numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for num in numbers:

print(num, end=' ')

**Вывод с помощью распаковки списка**

В Python есть удобный способ вывода элементов списка без использования цикла for.

**Вариант 1.** Вывод элементов списка через один символ пробела:

numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(\*numbers)

Такой код выведет:

0 1 2 3 4 5 6 7 8 9 10

**Вариант 2.** Вывод элементов списка, каждого на отдельной строке

numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

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

Такой код выведет:

0

1

2

3

4

5

6

7

8

9

10

Поскольку строки содержат символы, подобно тому, как списки содержат элементы, то мы можем использовать распаковку строк точно так же, как и распаковку списков.

Такой код:

s = 'Python'

print(\*s)

print()

print(\*s, sep='\n')

выведет:

P y t h o n

P

y

t

h

o

n

## Значение функции

На вход программе подается натуральное число �*n*, а затем �*n* целых чисел. Напишите программу, которая для каждого введенного числа �*x* выводит значение функции �(�)=�2+2�+1*f*(*x*)=*x*2+2*x*+1, каждое на отдельной строке.

**Формат входных данных**  
На вход программе подаются натуральное число �*n*, а затем �*n* целых чисел, каждое на отдельной строке.

**Формат выходных данных**  
Программа должна вывести сначала введенные числа, затем пустую строку, а затем соответствующие значения функции.

**Примечание.** Для первого теста имеем:�(1)=12+2⋅1+1=4,�(2)=22+2⋅2+1=9,�(3)=32+2⋅3+1=16,…*f*(1)=12+2⋅1+1=4,*f*(2)=22+2⋅2+1=9,*f*(3)=32+2⋅3+1=16,…

n=int(input())

s=[]

p=[]

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

s.append(int(input()))

print(\*s,sep='\n')

print()

for j in s:

p.append((j+1)\*\*2)

print(\*p,sep='\n')

## Remove outliers

При анализе данных, собранных в рамках научного эксперимента, бывает полезно удалить самое большое и самое маленькое значение.

На вход программе подается натуральное число �*n*, а затем �*n* **различных** натуральных чисел. Напишите программу, которая удаляет наименьшее и наибольшее значение из указанных чисел, а затем выводит оставшиеся числа каждое на отдельной строке, не меняя их порядок.

**Формат входных данных**  
На вход программе подаются натуральное число �*n*, а затем �*n* различных натуральных чисел, каждое на отдельной строке.

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

n=int(input())

s=[]

for i in range(n):

s.append(int(input()))

a=min(s)

b=max(s)

s.remove(a)

s.remove(b)

print(\*s,sep='\n')

## Без дубликатов

На вход программе подается натуральное число �*n*, а затем �*n* строк. Напишите программу, которая выводит только уникальные строки, в том же порядке, в котором они были введены.

**Формат входных данных**  
На вход программе подаются натуральное число �*n*, а затем �*n* строк, каждая на отдельной строке.

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

**Примечание.** Считайте, что все строки состоят из строчных символов.

n=int(input())

s=[]

for i in range(n):

x=input()

if x not in s:

s.append(x)

print(\*s, sep='\n')

## Google search - 1

На вход программе подается натуральное число �*n*, затем �*n* строк, затем еще одна строка — поисковый запрос. Напишите программу, которая выводит все введенные строки, в которых встречается поисковый запрос.

**Формат входных данных**  
На вход программе подаются натуральное число �*n* — количество строк, затем сами строки в указанном количестве, затем один поисковый запрос.

**Формат выходных данных**  
Программа должна вывести все введенные строки, в которых встречается поисковый запрос.

**Примечание.** Поиск не должен быть чувствителен к регистру символов.

n=int(input())

s=[]

for i in range(n):

x=input()

s.append(x)

y=input()

for x in s:

if y.lower() in x.lower():

print(x)

## Google search - 2 🌶️🌶️

На вход программе подается натуральное число �*n*, затем �*n* строк, затем число �*k* — количество поисковых запросов, затем �*k* строк — поисковые запросы. Напишите программу, которая выводит все введенные строки, в которых встречаются одновременно **все** поисковые запросы.

**Формат входных данных**  
На вход программе подаются натуральное число �*n* — количество строк, затем сами строки в указанном количестве, затем число �*k*, затем сами поисковые запросы.

**Формат выходных данных**  
Программа должна вывести все введенные строки, в которых встречаются **все** поисковые запросы.

**Примечание.** Поиск не должен быть чувствителен к регистру символов.

n=int(input())

s=[]

for i in range(n):

x=input()

s.append(x)

k=int(input())

p=[]

for j in range(k):

y=input()

p.append(y)

c=[]

for l in s:

a=0

for g in p:

if g.lower() in l.lower():

a+=1

if a>=len(p):

print(l)

## Negatives, Zeros and Positives

На вход программе подается натуральное число �*n*, а затем �*n* целых чисел. Напишите программу, которая сначала выводит все отрицательные числа, затем нули, а затем все положительные числа, каждое на отдельной строке. Числа должны быть выведены в том же порядке, в котором они были введены.

**Формат входных данных**  
На вход программе подаются натуральное число �*n*, а затем �*n* целых чисел, каждое на отдельной строке.

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

n=int(input())

s=[]

a=[]

b=[]

for i in range(n):

x=int(input())

if x<0:

s.append(x)

elif x>0:

b.append(x)

else:

a.append(x)

print(\*(s+a+b),sep='\n')

**Тема урока: строковые методы**

1. Метод split()
2. Метод join()

**Аннотация.** Строковые методы split() и join().

В предыдущем модуле мы детально изучили основные строковые методы, однако обошли стороной два важных: split() и join(), имеющих отношение к спискам. Они как бы противоположны по смыслу: метод split() разбивает строку по произвольному разделителю на список слов, а метод join() собирает строку из списка слов через заданный разделитель.

**Метод split()**

Метод split() разбивает строку на слова, используя в качестве разделителя последовательность пробельных символов.

Следующий программный код:

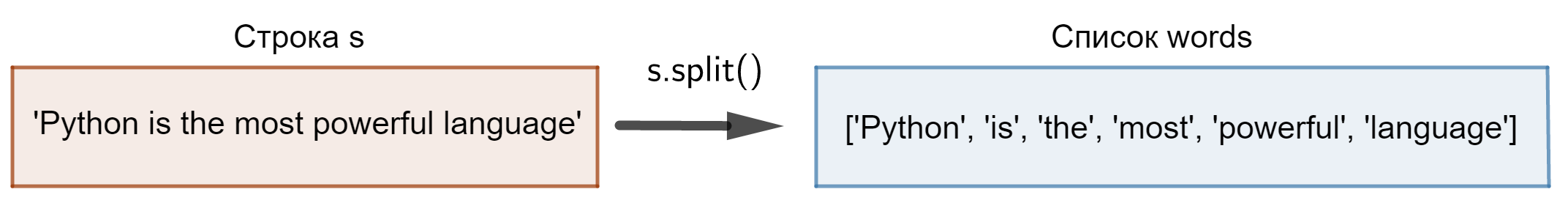
s = 'Python is the most powerful language'

words = s.split()

print(words)

выведет:

['Python', 'is', 'the', 'most', 'powerful', 'language']



Таким образом, вызов метода split() разбивает строку на слова и возвращает список, содержащий все слова.

Рассмотрим следующий программный код:

numbers = input().split()

Если при запуске этой программы ввести строку 1 2 3 4 5, то список numbers будет следующим  ['1', '2', '3', '4', '5']. Обратите внимание, что список будет состоять из строк, а не из чисел. Если требуется получить именно список чисел, то затем нужно элементы списка по одному преобразовать в числа:

numbers = input().split()

for i in range(len(numbers)):

numbers[i] = int(numbers[i])

**Необязательный параметр**

У метода split() есть необязательный параметр, который определяет, какой набор символов будет использоваться в качестве разделителя между элементами списка. Например, вызов метода split('.') вернет список, полученный разделением исходной строки по символу '.'.

Следующий программный код:

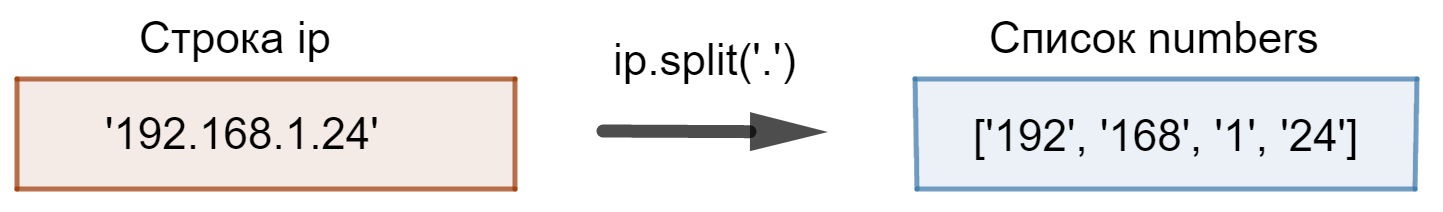
ip = '192.168.1.24'

numbers = ip.split('.') # указываем явно разделитель

print(numbers)

выведет список:

['192', '168', '1', '24']

****

**Метод join()**

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

 Следующий программный код:

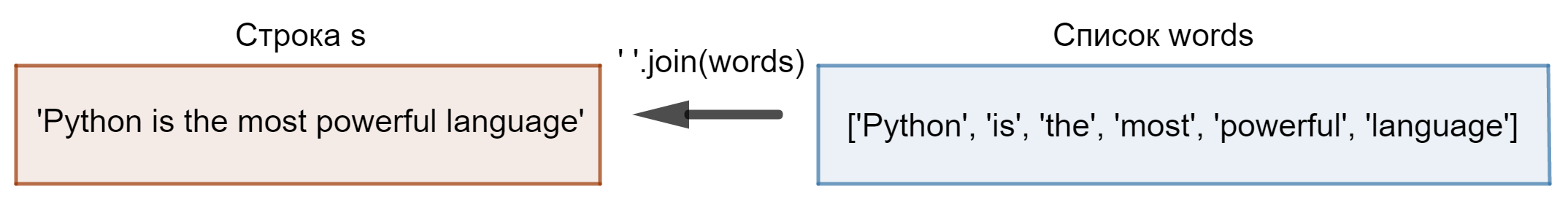
words = ['Python', 'is', 'the', 'most', 'powerful', 'language']

s = ' '.join(words)

print(s)

выведет:

Python is the most powerful language



Обратите внимание, все слова разделены одним пробелом, поскольку метод join() вызывался на строке, состоящей из одного символа пробела ' '.

Рассмотрим еще пару примеров:

words = ['Мы', 'учим', 'язык', 'Python']

print('\*'.join(words))

print('-'.join(words))

print('?'.join(words))

print('!'.join(words))

print('\*\*\*\*\*'.join(words))

print('abc'.join(words))

print('123'.join(words))

Результатом выполнения такого кода будет:

Мы\*учим\*язык\*Python

Мы-учим-язык-Python

Мы?учим?язык?Python

Мы!учим!язык!Python

Мы\*\*\*\*\*учим\*\*\*\*\*язык\*\*\*\*\*Python

МыabcучимabcязыкabcPython

Мы123учим123язык123Python

**Запомни:** Строковый метод split() служит для преобразования строки в список, а метод join() — для преобразования списка в строку.

**Примечания**

**Примечание 1.** Существует большая разница между результатами вызова методов s.split() и s.split(''). Разница в поведении проявляется, когда строка содержит несколько пробелов между словами.

Следующий программный код:

s = 'Python is the most powerful language'

words1 = s.split()

words2 = s.split(' ')

print(words1)

print(words2)

 выведет списки:

['Python', 'is', 'the', 'most', 'powerful', 'language']

['Python', '', '', '', 'is', '', '', 'the', '', 'most', '', 'powerful', '', 'language']

**Примечание 2.** Методы split() и join() являются строковыми методами. Следующий код приводит к ошибке:

print([1, 2].split())

print([1, 2].join([3, 4, 5]))

**Примечание 3.** Строковый метод join() работает только со списком строк. Следующий код приводит к ошибке:

numbers = [1, 2, 3, 4] # список чисел

s = '\*'.join(numbers)

print(s)

## Построчный вывод

На вход программе подается строка текста. Напишите программу, которая выводит слова введенной строки в столбик.

**Формат входных данных**  
На вход программе подается строка текста.

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

s=input()

w=s.split()

print(\*w,sep='\n')

## Инициалы

На вход программе подается строка текста, содержащая имя, отчество и фамилию человека. Напишите программу, которая выводит инициалы человека.

**Формат входных данных**  
На вход программе подается строка текста, содержащая имя, отчество и фамилию человека.

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

s=input().split()

for i in s:

print(i[0],end=".")

## Windows OS

В операционной системе Windows полное имя файла состоит из буквы диска, после которого ставится двоеточие и символ  "\",  затем через такой же символ перечисляются подкаталоги (папки), в которых находится файл, в конце пишется имя файла (C:\Windows\System32\calc.exe).

На вход программе подается одна строка с корректным именем файла в операционной системе Windows. Напишите программу, которая разбирает строку на части, разделенные символом "\". Каждую часть вывести в отдельной строке.

**Формат входных данных**  
На вход программе подается одна строка.

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

**Примечание.** В Python символ \ обычно используется для создания специальных символьных последовательностей, которые представляют собой управляющие символы или экранированные последовательности. Например, \n представляет символ новой строки, \t представляет символ табуляции и т.д. Однако если символ \ используется как часть строки, его следует экранировать, т.е. использовать два обратных слэша \\.

s=input()

print(\*s.split('\\'),sep='\n')

## Диаграмма

На вход программе подается строка текста, содержащая целые числа. Напишите программу, которая по заданным числам строит столбчатую диаграмму.

**Формат входных данных**  
На вход программе подается строка текста, содержащая целые числа, разделенных символом пробела.

**Формат выходных данных**  
Программа должна вывести столбчатую диаграмму.

s=input()

w=s.split()

for i in w:

print('+'\*int(i))

## Корректный ip-адрес

На вход программе подается строка текста, содержащая 44 целых неотрицательных числа, разделенных точкой. Напишите программу, которая определяет, является ли введенная строка текста корректным ip-адресом.

**Формат входных данных**  
На вход программе подается строка текста, содержащая 44 целых числа, разделенных точкой.

**Формат выходных данных**  
Программа должна вывести «ДА», если введеная строка является корректным ip-адресом, и «НЕТ» — в противном случае.

**Примечание.** ip-адрес является корректным, если все 44 числа находятся в диапазоне от 00 до 255255 включительно.

s=input().split('.')

for i in s:

if int(i)<0 or int(i)>=256:

print("НЕТ")

break

else:

print("ДА")

## Добавь разделитель

На вход программе подается строка текста и строка-разделитель. Напишите программу, которая вставляет указанный разделитель между каждым символом введенной строки текста.

**Формат входных данных**  
На вход программе подается строка текста и строка-разделитель, каждая на отдельной строке

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

s=input()

n=input()

print(n.join(s))

## Количество совпадающих пар

На вход программе подается строка текста, содержащая целые числа. Из данной строки формируется список чисел. Напишите программу, которая подсчитывает, сколько в полученном списке пар элементов, равных друг другу. Считается, что любые два элемента, равные друг другу образуют одну пару, которую необходимо посчитать.

**Формат входных данных**  
На вход программе подается строка текста, содержащая целые числа, отделенные символом пробела.

**Формат выходных данных**  
Программа должна вывести одно число – количество пар элементов, равных друг другу.

s=input().split()

a=0

for i in range(len(s)):

for j in range(i+1,len(s)):

if s[j]==s[i]:

a+=1

print(a)

**Тема урока: методы списков**

1. Метод insert()
2. Метод index()
3. Метод remove()
4. Метод pop()
5. Метод reverse()
6. Метод count()
7. Метод clear()
8. Метод copy()
9. Метод sort()

**Аннотация.** Другие методы списков.

Мы уже познакомились с двумя списочными методами append() и extend(). Первый добавляет в конец списка один новый элемент, а второй расширяет список другим списком. К спискам в Python применимы и другие удобные методы, с которыми мы познакомимся в этом уроке.

**Метод insert()**

Метод insert() позволяет вставлять значение в список в заданной позиции. В него передается два аргумента:

1. index: индекс, задающий место вставки значения;
2. value: значение, которое требуется вставить.

Когда значение вставляется в список, список расширяется в размере, чтобы разместить новое значение. Значение, которое ранее находилось в заданной индексной позиции, и все элементы после него сдвигаются на одну позицию к концу списка.

Следующий программный код:

names = ['Gvido', 'Roman' , 'Timur']

print(names)

names.insert(0, 'Anders')

print(names)

names.insert(3, 'Josef')

print(names)

выведет:

['Gvido', 'Roman' , 'Timur']

['Anders', 'Gvido', 'Roman' , 'Timur']

['Anders', 'Gvido', 'Roman' , 'Josef', 'Timur']

При указании недопустимого индекса во время выполнения программы ошибки не происходит. Если задан индекс за пределами конца списка, то значение будет добавлено в конец списка. Если применен отрицательный индекс, который указывает на недопустимую позицию, то значение будет вставлено в начало списка.

**Метод index()**

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

1. value: значение, индекс которого требуется найти.

Если элемент в списке 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: 'Anders' is not in list

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 списке')

**Метод remove()**

Метод remove() удаляет первый элемент, значение которого равняется переданному в метод значению. В метод передается один параметр:

1. value: значение, которое требуется удалить.

Метод уменьшает размер списка на один элемент. Все элементы после удаленного элемента смещаются на одну позицию к началу списка. Если элемент в списке не найден, то во время выполнения происходит ошибка.

Следующий программный код:

food = ['Рис', 'Курица', 'Рыба', 'Брокколи', 'Рис']

print(food)

food.remove('Рис')

print(food)

выведет:

['Рис', 'Курица', 'Рыба', 'Брокколи', 'Рис']

['Курица', 'Рыба', 'Брокколи', 'Рис']

**Важно:** метод remove() удаляет только первый элемент с указанным значением. Все последующие его вхождения остаются в списке. Чтобы удалить все вхождения, нужно использовать цикл while в связке с оператором принадлежности in и методом remove().

**Метод pop()**

Метод pop() удаляет элемент по указанному индексу и возвращает его. В метод pop() передается один **необязательный** аргумент:

1. index: индекс элемента, который требуется удалить.

Если индекс не указан, то метод удаляет и возвращает последний элемент списка. Если список пуст или указан индекс за пределами диапазона, то во время выполнения происходит ошибка.

Следующий программный код:

names = ['Gvido', 'Roman', 'Timur']

item = names.pop(1)

print(item)

print(names)

выведет:

Roman

['Gvido', 'Timur']

**Метод count()**

Метод count() возвращает количество элементов в списке, значения которых равны переданному в метод параметру.

Таким образом, в метод передается один параметр:

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

Если значение в списке не найдено, то метод возвращает 00.

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

**Метод reverse()**

Метод reverse()инвертирует порядок следования значений в списке, то есть меняет его на противоположный.

Следующий программный код:

names = ['Gvido', 'Roman' , 'Timur']

names.reverse()

print(names)

выведет:

['Timur', 'Roman', 'Gvido']

Существует большая разница между вызовом метода names.reverse() и использованием среза names[::-1]. Метод reverse() меняет порядок элементов на обратный **в текущем списке**, а срез создает копию списка, в котором элементы следуют в обратном порядке.

**Метод clear()**

Метод clear() удаляет все элементы из списка.

Следующий программный код:

names = ['Gvido', 'Roman' , 'Timur']

names.clear()

print(names)

выведет:

[]

**Метод copy()**

Метод copy() создает поверхностную копию списка.

Следующий программный код:

names = ['Gvido', 'Roman' , 'Timur']

names\_copy = names.copy() # создаем поверхностную копию списка names

print(names)

print(names\_copy)

выведет:

['Gvido', 'Roman', 'Timur']

['Gvido', 'Roman', 'Timur']

Аналогичного результата можно достичь с помощью срезов или функции list():

names = ['Gvido', 'Roman' , 'Timur']

names\_copy1 = list(names) # создаем поверхностную копию с помощью функции list()

names\_copy2 = names[:] # создаем поверхностную копию с помощью среза от начала до конца

Подробнее про поверхностные копии мы поговорим в курсе для продвинутых.

**Примечания**

**Примечание.** Существует большая разница в работе строковых и списочных методов. Строковые методы не изменяют содержимого объекта, к которому они применяются, а возвращают новое значение. Списочные методы (кроме методов index(), count(), copy()), напротив, меняют содержимое объекта, к которому применяются.

Например, когда мы применяем метод lower() к строке original\_string, то создаётся новая строка lowercase\_string, которая не имеет ничего общего со строкой original\_string. Более точно будет сказать, что строковый метод вернул новую строку, не изменяя старую:

original\_string = "HELLO WORLD"

lowercase\_string = original\_string.lower() # тут возвращается новая, модифицированная строка

print(original\_string) # HELLO WORLD

print(lowercase\_string) # hello world

Как мы видим, сама строка original\_string не изменилась. Мы просто использовали её для создания строки lowercase\_string через метод lower(). А lowercase\_string действительно создана на основе строки original\_string, где каждый символ в нижнем регистре.

Совершенно иная ситуация возникает при работе со списками. Например, при добавлении элемента в список через метод append() мы изменяем наш список, не создавая новый. Именно поэтому использование метода append() ничего не возвращает.

original\_list = [1, 2, 4, 7]

modified\_list = original\_list.append(11)  # тут ничего не возвращается

print(original\_list)  # [1, 2, 4, 7, 11]

print(modified\_list)  # None

Как мы видим, original\_list изменился. Элемент 1111 был добавлен в конец. А вот наша попытка присвоить списку modified\_list метод append() не увенчалась успехом. Был возвращён объект None (подробнее этот объект будет рассмотрен в курсе для продвинутых). Вкратце, объект None используется для обозначения отсутствия значения или буквально "ничего", т.е. пустоты.

**Все сразу 2 🌶️**

Дополните приведенный код, чтобы он:

1. Заменил второй элемент списка на 17;
2. Добавил числа 4, 5 и 6 в конец списка;
3. Удалил первый элемент списка;
4. Удвоил список;
5. Вставил число 25 по индексу 3;
6. Вывел список, с помощью функции print()

numbers = [8, 9, 10, 11]

numbers[1]=17

numbers.extend([4,5,6])

del numbers[0]

numbers\*=2

numbers.insert(3,25)

print(numbers)

## Переставить min и max

На вход программе подается строка текста, содержащая **различные** натуральные числа. Из данной строки формируется список чисел. Напишите программу, которая меняет местами минимальный и максимальный элемент этого списка.

**Формат входных данных**  
На вход программе подается строка текста, содержащая различные натуральные числа, разделенные символом пробела.

**Формат выходных данных**  
Программа должна вывести строку текста в соответствии с условием задачи.

**Примечание.** Используйте подходящие встроенные функции и списочные методы.

s=input().split()

n=[]

for i in s:

n.append(int(i))

a=n.index(max(n))

b=n.index(min(n))

n[a],n[b]=n[b],n[a]

print(\*n)

## Количество артиклей

На вход программе подается строка, содержащая английский текст. Напишите программу, которая подсчитывает общее количество артиклей: 'a', 'an', 'the'.

**Формат входных данных**  
На вход программе подается строка, содержащая английский текст. Слова текста разделены символом пробела.

**Формат выходных данных**  
Программа должна вывести общее количество артиклей 'a', 'an', 'the' вместе с поясняющим текстом.

**Примечание.** Артикли могут начинаться с заглавной буквы 'A', 'An', 'The'.

s=input().lower().split()

n=s.count('a')

m=s.count('an')

p=s.count('the')

print('Общее количество артиклей:',n+m+p)

## Взлом Братства Стали 🌶️

Немалоизвестный в пустошах Мохаве Курьер забрел в Хидден-Вэли – секретный бункер Братства Стали и любезно соглашается помочь им в решении их проблем. Одной из такой проблем являлся странный компьютерный вирус, который проявлялся в виде появления комментариев к программам на терминалах Братства Стали. Известно, что программисты Братства никогда не оставляют комментарии к коду и пишут программы на Python, поэтому удаление всех этих комментариев никак не навредит им. Помогите писцу Ибсену удалить все комментарии из программы.  
  
**Формат входных данных**  
На первой строке вводится символ решётки и сразу же натуральное число �*n* — количество строк в программе, не считая первой. Далее следует �*n* строк кода.  
  
**Формат выходных данных**  
Нужно вывести те же строки, но удалить комментарии и символы пустого пространства в конце строк. Пустую строку вместо первой строки ввода выводить не надо.

**Примечание.** Обращайте внимание на лишние пробелы в конце строк. Проверяющая система не пропустит ваше решение с ними.

n=input()

for \_ in range(int(n[1:])):

s = input()

if '#' in s:

s = s[:s.find('#')]

print(s.rstrip())

## Метод sort()

В Python списки имеют встроенный метод sort(), который сортирует элементы списка по возрастанию или убыванию.

Следующий программный код:

a = [1, 7, -3, 9, 0, -67, 34, 12, 45, 1000, 6, 8, -2, 99]

a.sort()

print('Отсортированный список:', a)

выведет:

Отсортированный список: [-67, -3, -2, 0, 1, 6, 7, 8, 9, 12, 34, 45, 99, 1000]

По умолчанию метод sort() сортирует список по возрастанию. Если требуется отсортировать список по убыванию, необходимо явно указать параметр reverse = True.

Следующий программный код:

a = [1, 7, -3, 9, 0, -67, 34, 12, 45, 1000, 6, 8, -2, 99]

a.sort(reverse=True) # сортируем по убыванию

print('Отсортированный список:', a)

выведет:

Отсортированный список: [1000, 99, 45, 34, 12, 9, 8, 7, 6, 1, 0, -2, -3, -67]

## Примечания

**Примечание 1.** С помощью метода sort() можно сортировать списки содержащие не только числа, но и строки. В таком случае элементы списка сортируются в соответствии с [лексикографическим порядком](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D0%BA%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA).

Следующий программный код:

a = ['бета', 'альфа', 'дельта', 'гамма']

a.sort()

print('Отсортированный список:', a)

выведет:

Отсортированный список: ['альфа', 'бета', 'гамма', 'дельта']

**Примечание 2.** Метод sort() использует алгоритм [Timsort](https://ru.wikipedia.org/wiki/Timsort" \t "_blank).

## Сортировка чисел

На вход программе подается строка текста, содержащая целые числа. Из данной строки формируется список чисел. Напишите программу, которая сортирует и выводит данный список сначала по возрастанию, а затем по убыванию.

**Формат входных данных**  
На вход программе подается строка текста, содержащая целые числа, разделенные символом пробела.

**Формат выходных данных**  
Программа должна вывести две строки текста в соответствии с условием задачи.

s=input().split()

for i in range(len(s)):

s[i]=int(s[i])

s.sort()

print(\*s)

s.sort(reverse=True)

print(\*s)

**Тема урока: списочные выражения**

1. Списочные выражения
2. Решение задач

**Аннотация.** Списочные выражения. Создание списков без явного использования циклов и вызова списочного метода append().

**Создание списков**

Для того, чтобы создать список состоящий из 1010 нулей мы можем использовать следующий код:

zeros = []

for i in range(10):

zeros.append(0)

В Python, однако есть более простой и компактный способ для создания такого списка. Мы можем использовать оператор умножения списка на число:

zeros = [0] \* 10

Для создания списков, заполненных по более сложным правилам нам приходится явно использовать цикл for.

Например, для создания списка целых чисел от 00 до 99, мы вынуждены писать такой код:

numbers = []

for i in range(10):

numbers.append(i)

Такой код хоть и не является сложным, однако достаточно громоздок.

**Списочные выражения**

В Python есть механизм для создания списков из неповторяющихся элементов. Такой механизм называется — **списочное выражение** (list comprehension).

Предыдущий код можно записать следующим образом:

numbers = [i for i in range(10)]

Общий вид списочного выражения следующий:

[выражение for переменная in последовательность]

где переменная — имя некоторой переменной, последовательность — последовательность значений, которые она принимает (список, строка или объект, полученный при помощи функции range), выражение — некоторое выражение, как правило, зависящее от использованной в списочном выражении переменной, которым будут заполнены элементы списка.

**Примеры использования списочных выражений**

**1. Создать список, заполненный 10 нулями можно и при помощи списочного выражения:**

zeros = [0 for i in range(10)]

**2. Создать список, заполненный квадратами целых чисел от 0 до 9 можно так:**

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

**3. Создать список, заполненный кубами целых чисел от 10 до 20 можно так:**

cubes = [i \*\* 3 for i in range(10, 21)]

**4. Создать список, заполненный символами строки:**

chars = [c for c in 'abcdefg']

print(chars)

**Считывание входных данных**

При решении многих задач из предыдущих уроков мы считывали начальные данные (строки, числа) и заполняли ими список. С помощью списочных выражений процесс заполнения списка можно заметно сократить.

Например, если сначала вводится число n – количество строк, а затем сами строки, то создать список можно так:

n = int(input())

lines = [input() for \_ in range(n)]

Можно опустить описание переменной n:

lines = [input() for \_ in range(int(input()))]

Если требуется считать список чисел, то необходимо добавить преобразование типов:

numbers = [int(input()) for \_ in range(int(input()))]

Обратите внимание, мы используем символ \_ в качестве имени переменной цикла, поскольку она не используется.

Списочные выражения часто используются для инициализации списков. В Python не принято создавать пустые списки, а затем заполнять их значениями, если можно этого избежать.

**Условия в списочном выражении**

В списочных выражениях можно использовать условный оператор. Например, если требуется создать список четных чисел от 0 до 20, то мы можем написать такой код:

evens = [i for i in range(21) if i % 2 == 0]

**Важно:** для того, чтобы получить список, состоящий из четных чисел, лучше использовать функцию range(0, 21, 2). Предыдущий пример приведен для демонстрации возможности использования условий в списочных выражениях.

**Вложенные циклы**

В списочном выражении можно использовать вложенные циклы.

Следующий программный код:

numbers = [i \* j for i in range(1, 5) for j in range(2)]

print(numbers)

выведет список:

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

Такой код равнозначен следующему:

numbers = []

for i in range(1, 5):

for j in range(2):

numbers.append(i \* j)

print(numbers)

**Подводя итог**

Пусть word = 'Hello', numbers = [1, 14, 5, 9, 12], words = ['one', 'two', 'three', 'four', 'five', 'six'].

| **Списочное выражение** | **Результирующий список** |
| --- | --- |
| [0 for i in range(10)] | [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] |
| [i \*\* 2 for i in range(1, 8)] | [1, 4, 9, 16, 25, 36, 49] |
| [i \* 10 for i in numbers] | [10, 140, 50, 90, 120] |
| [c \* 2 for c in word] | ['HH', 'ee', 'll', 'll', 'oo'] |
| [m[0] for m in words] | ['o', 't', 't', 'f', 'f', 's'] |
| [i for i in numbers if i < 10] | [1, 5, 9] |
| [m[0] for m in words if len(m) == 3] | ['o', 't', 's'] |

## Списочное выражение 1

На вход программе подается натуральное число �*n*. Напишите программу, использующую **списочное выражение**, которая создает список, содержащий квадраты чисел от 11 до �*n* (включительно), а затем выводит его элементы построчно, то есть каждый на отдельной строке.

**Формат входных данных**  
На вход программе подается натуральное число.

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

**Примечание.** Для вывода элементов списка используйте цикл for.

n=int(input())

s = [i \*\* 2 for i in range(1,n+1)]

print(\*s,sep='\n')

## Списочное выражение 2

На вход программе подается строка текста, содержащая целые числа. Напишите программу, **использующую списочное выражение**, которая выведет кубы указанных чисел также на одной строке.

**Формат входных данных**  
На вход программе подается строка текста, содержащая целые числа, разделенные символом пробела.

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

**Примечание 1.** Для вывода элементов списка используйте цикл for.

**Примечание 2.** Используйте метод split().

str=input().split()

cubes = [int(i) \*\* 3 for i in str]

print(\*cubes)

## В одну строку 1

На вход программе подается строка текста, содержащая слова. Напишите программу, которая выводит слова введенной строки в столбик.

**Формат входных данных**  
На вход программе подается строка текста, содержащая слова, разделенные символом пробела.

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

**Примечание.** Программу можно написать в одну строку кода.

str=input().split()

n=[i for i in str]

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

## В одну строку 2

На вход программе подается строка текста. Напишите программу, **использующую списочное выражение**, которая выводит все цифровые символы данной строки.

**Формат входных данных**  
На вход программе подается строка текста.

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

**Примечание.** Программу можно написать в одну строку кода.

str=input()

n=[i for i in str if i.isdigit()]

print(\*n, sep='')

## В одну строку 3

На вход программе подается строка текста, содержащая целые числа. Напишите программу, **использующую списочное выражение**, которая выводит не оканчивающиеся на цифру 44 квадраты четных чисел.

**Формат входных данных**  
На вход программе подается строка текста, содержащая целые числа, разделенные символом пробела.

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

**Примечание 1.** Программу можно написать в одну строку кода.

**Примечание 2**. На цифру 44 не должны оканчиваться именно квадраты чисел, а не сами числа.

str=input().split()

n=[int(i)\*\*2 for i in str if int(i)%2==0 and int(i)\*\*2%10!=4]

print(\*n)

**Тема урока: сортировка списков**

1. Задача сортировки;
2. Сортировка пузырьком;
3. Сортировка выбором;
4. Сортировка простыми вставками.

**Аннотация.**  Задачи и способы (алгоритмы) сортировки списков.

**Задача сортировки**

Задача сортировки списка заключается в перестановке его элементов так, чтобы они были упорядочены по возрастанию или убыванию. Это одна из основных задач программирования. Мы сталкиваемся с ней очень часто: при записи фамилий учеников в классном журнале, при подведении итогов соревнований и т.д.

**Алгоритмы сортировки**

**Алгоритм сортировки** — это алгоритм упорядочивания элементов в списке. Алгоритмы сортировки оцениваются по скорости выполнения и эффективности использования памяти:

* время — основной параметр, характеризующий быстродействие алгоритма;
* память — ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных.

    Алгоритмы сортировки, не потребляющие дополнительной памяти, относят к **сортировкам на месте**.

**Основные алгоритмы сортировки**

**Медленные:**

1. Пузырьковая сортировка (Bubble sort);
2. Сортировка выбором (Selection sort);
3. Сортировка простыми вставками (Insertion sort).

**Быстрые:**

1. Сортировка Шелла (Shell sort);
2. Быстрая сортировка (Quick sort);
3. Сортировка слиянием (Merge sort);
4. Пирамидальная сортировка (Heap sort);
5. Сортировка TimSort (используется в Java и Python).

Большинство алгоритмов сортировки, в частности, указанные выше, основаны на сравнении двух элементов списка. Существуют однако алгоритмы не основанные на сравнениях. Такие алгоритмы как правило используют наперед заданные условия относительно элементов списка. Например, элементами списка являются натуральные или целые числа в некотором диапазоне, элементами являются строки и т.д.

К алгоритмам не основанным на сравнениях можно отнести следующие:

1. Сортировка подсчетом (Counting sort);
2. Блочная сортировка (Bucket sort);
3. Поразрядная сортировка (Radix sort).

В рамках курса мы рассмотрим несложные алгоритмы пузырьковой сортировки, сортировки выбором и сортировки простыми вставками.

**Примечания**

**Примечание 1.** Подробнее об алгоритмах сортировки можно почитать [тут](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8).

**Примечание 2.** Мы называем некоторые алгоритмы сортировки **медленными**, поскольку они тратят много времени на сортировку больших списков. Например, если список содержит порядка миллиона элементов, то такие алгоритмы тратят часы, а то и дни на выполнение сортировки, в то время как быстрые алгоритмы справляются с задачей за секунды.

**Примечание 3.** Наглядную работу алгоритмов сортировки на разных входных данных можно посмотреть <https://www.toptal.com/developers/sorting-algorithms>

**Сортировка пузырьком**

Алгоритм сортировки пузырьком состоит из повторяющихся проходов по сортируемому списку. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по списку повторяются �−1*n*−1 раз, где �*n* – длина списка. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент списка ставится на свое место в конце списка рядом с предыдущим «наибольшим элементом».

Наибольший элемент каждый раз «всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма.

Алгоритм пузырьковой сортировки считается учебным и практически не применяется вне учебной литературы, а на практике применяются более эффективные.

Рассмотрим работу алгоритма на примере сортировки списка a = [5, 1, 4, 2, 8] по возрастанию.

**Первый проход:**

1. [**5, 1**, 4, 2, 8] → [**1, 5**, 4, 2, 8]: меняем местами первый и второй элементы, так как 5>15>1;
2. [1,**5, 4**, 2, 8] → [1, **4, 5**, 2, 8]: меняем местами второй и третий элементы, так как 5>45>4;
3. [1, 4, **5, 2**, 8] → [1, 4, **2, 5**, 8]: меняем местами третий и четвертый элементы, так как 5>25>2;
4. [1, 4, 2, **5, 8**] → [1, 4, 2, **5, 8**]: не меняем четвертый и пятый элементы местами, так как 5<85<8;
5. Самый большой элемент встал («всплыл») на свое место.

**Второй проход:**

1. [**1, 4**, 2, 5, **8**] → [**1, 4**, 2, 5, **8**]: не меняем первый и второй элементы местами, так как 1<41<4;
2. [1, **4, 2**, 5, **8**] → [1, **2, 4**, 5, **8**]: меняем местами второй и третий элементы, так как 4>24>2;
3. [1, 2, **4, 5**, **8**] → [1, 2, **4, 5, 8**]: не меняем местами третий и четвертый элементы, так как 4 <54 <5;
4. Второй по величине элемент встал («всплыл») на свое место.

Теперь список полностью отсортирован, но алгоритму это неизвестно и он работает дальше.

**Третий проход:**

1. [**1, 2**, 4,**5, 8**] → [**1, 2**, 4,**5, 8**]: не меняем первый и второй элементы местами, так как 1<21<2;
2. [1, **2, 4**,**5, 8**] → [1, **2, 4**,**5, 8**]: не меняем второй и третий элементы местами, так как 2 <42 <4;
3. Третий по величине элемент встал («всплыл») на свое место. (на котором и был)

**Четвертый проход:**

1. [**1, 2**,**4**,**5, 8**] → [**1, 2**,**4**,**5, 8**]:
2. Четвертый по величине элемент встал («всплыл») на свое место.

Теперь список отсортирован и алгоритм может быть завершен.

**Визуализация алгоритма**

**Реализация алгоритма**

Пусть требуется отсортировать по возрастанию список чисел: a = [1, 7, -3, 9, 0, -67, 34, 12, 45, 1000, 6,  8, -2, 99].

Следующий программный код реализует алгоритм пузырьковой сортировки:

a = [1, 7, -3, 9, 0, -67, 34, 12, 45, 1000, 6, 8, -2, 99]

n = len(a)

for i in range(n - 1):

for j in range(n - 1 - i):

if a[j] > a[j + 1]: # если порядок элементов пары неправильный

a[j], a[j + 1] = a[j + 1], a[j] # меняем элементы пары местами

print('Отсортированный список:', a)

Результатом выполнения такого кода будет:

Отсортированный список: [-67, -3, -2, 0, 1, 6, 7, 8, 9, 12, 34, 45, 99, 1000]

**Оптимизация алгоритма**

Алгоритм пузырьковой сортировки можно немного ускорить. Если на одном из очередных проходов окажется, что обмены больше не нужны, то это означает, что все элементы списка находятся на своих местах, то есть список отсортирован. Для реализации такого ускорения нужно воспользоваться сигнальной меткой, то есть флажком и оператором прерывания break.

Оптимизируйте приведенный код, реализующий алгоритм пузырьковой сортировки.

a = [17, 24, 91, 96, 67, -27, 79, -71, -71, 58, 48, 88, 88, -16, -78, 96, -76, 56, 92, 1, 32, -17, 36, 88, -61, -97, -37, -84, 50, 47, 94, -6, 52, -76, 93, 14, -32, 98, -65, -16, -9, -68, -20, -40, -71, 93, -91, 44, 25, 79, 97, 0, -94, 7, -47, -96, -55, -58, -78, -78, -79, 75, 44, -56, -41, 38, 16, 70, 17, -17, -24, -83, -74, -73, 11, -26, 63, -75, -19, -13, -51, -74, 21, -8, 21, -68, -66, -84, -95, 78, 69, -29, 39, 38, -55, 7, -11, -26, -62, -84]

n = len(a)

m=0

for i in range(n - 1):

for j in range(n - i - 1):

if a[j] > a[j + 1]:

m+=1

a[j], a[j + 1] = a[j + 1], a[j]

if m==0:

break

print(a)

## Сортировка выбором

Сортировка выбором улучшает пузырьковую сортировку, совершая всего **один обмен за каждый проход по списку**. Для этого алгоритм ищет максимальный элемент и помещает его на соответствующую позицию. Как и для пузырьковой сортировки, после первого прохода самый большой элемент находится на правильном месте. После второго прохода на своё место становится следующий максимальный элемент. Проходы по списку повторяются �−1*n*−1 раз, где �*n* – длина списка, поскольку последний из них автоматически оказывается на своем месте.

Алгоритм сортировки выбором также считается учебным и практически не применяется вне учебной литературы. На практике используют более эффективные алгоритмы.

Рассмотрим работу алгоритма на примере сортировки списка a = [5, 1, 8, 2, 4] по возрастанию.

**Первый проход:**

Находим максимальный элемент 8 в неотсортированной части списка и меняем его с последним элементом списка:

[5, 1, 4, 2, **8**].

**Второй проход:**

Находим максимальный элемент 5 в неотсортированной части списка и меняем его с предпоследним элементом списка:

[2, 1, 4, **5**, **8**].

**Третий проход:**

Находим максимальный элемент 4 в неотсортированной части списка и меняем его с пред-предпоследним элементом списка:

[2, 1, **4**, **5**, **8**].

**Четвертый проход:**

Находим максимальный элемент 2 в неотсортированной части списка и меняем его с вторым элементом списка:

[1, **2**, **4**, **5**, **8**].

Теперь список отсортирован и алгоритм может быть завершен.

    Вместо максимального элемента можно искать минимальный.

## Визуализация алгоритма

Выберите в выпадающем списке алгоритм: **Selection sort**.

Отсортируйте список по возрастанию, реализовав алгоритм сортировки выбором.

a = [78, -32, 5, 39, 58, -5, -63, 57, 72, 9, 53, -1, 63, -97, -21, -94, -47, 57, -8, 60, -23, -72, -22, -79, 90, 96, -41, -71, -48, 84, 89, -96, 41, -16, 94, -60, -64, -39, 60, -14, -62, -19, -3, 32, 98, 14, 43, 3, -56, 71, -71, -67, 80, 27, 92, 92, -64, 0, -77, 2, -26, 41, 3, -31, 48, 39, 20, -30, 35, 32, -58, 2, 63, 64, 66, 62, 82, -62, 9, -52, 35, -61, 87, 78, 93, -42, 87, -72, -10, -36, 61, -16, 59, 59, 22, -24, -67, 76, -94, 59]

n = len(a)

a.append(a.pop(a.index(max(a))))

for j in range(1, n):

a.insert(-j, a.pop(a.index(max(a[:-j]))))

print(a)

## Сортировка простыми вставками

Алгоритм сортировки простыми вставками делит список на 22 части — отсортированную и неотсортированную. Из неотсортированной части извлекается очередной элемент и вставляется на нужную позицию в отсортированной части, в результате чего отсортированная часть списка увеличивается, а неотсортированная уменьшается. Так происходит, пока не исчерпан набор входных данных  и не отсортированы все элементы.

Сортировка простыми вставками наиболее эффективна, когда список уже частично отсортирован и элементов массива немного. Если элементов в списке меньше 1010, то этот алгоритм — один из самых быстрых.

Рассмотрим его работу на примере сортировки списка a = [5, 1, 8, 2, 4] по возрастанию.

**Первый проход:**

Делим список на две части: отсортированную [5] и неотсортированную [1, 8, 2, 4].

Извлекаем первый элемент 1 из неотсортированной части списка и находим ему место в отсортированной части:

[**1, 5**, 8, 2, 4].

**Второй проход:**

Делим список на две части: отсортированную [1, 5] и неотсортированную [8, 2, 4].

Извлекаем первый элемент 8 из неотсортированной части списка и находим ему место в отсортированной части:

[**1, 5, 8**, 2, 4].

**Третий проход:**

Делим список на две части: отсортированную [1, 5, 8] и неотсортированную [2, 4].

Извлекаем первый элемент 2 из неотсортированной части списка и находим ему место в отсортированной части:

[**1, 2, 5, 8**, 4].

**Четвертый проход:**

Делим список на две части: отсортированную [1, 2, 5, 8] и неотсортированную [4].

Извлекаем первый элемент 4 из неотсортированной части списка и находим ему место в отсортированной части:

[**1, 2, 4, 5, 8**].

Теперь список отсортирован, и алгоритм может быть завершен.

## Визуализация алгоритма

Выберите в выпадающем списке алгоритм: **Insert sort**.

## Сумма чисел

На вход программе подается строка текста, содержащая натуральные числа. Напишите программу, которая вставляет между каждым числом знак +, а затем вычисляет сумму полученных чисел.

**Формат входных данных**  
На вход программе подается строка текста, содержащая натуральные числа, разделенные символом пробела.

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

**Примечание.** Строковый метод join() работает только со списком строк.

a=[int(i) for i in input().split()]

b=sum(a)

print(\*a,sep='+',end='=')

print(b)

seq = input().split()

expression = "+".join(seq)

sm = 0

for el in seq:

sm += int(el)

expression += f"={sm}"

print(expression)

**Валидный номер 🌶️🌶️**

На вход программе подается строка текста. Напишите программу, которая определяет, является ли введенная строка корректным телефонным номером. Строка текста является корректным телефонным номером, если она имеет формат:

* abc-def-hijk или
* 7-abc-def-hijk,

где a, b, c, d, e, f, h, i, j, k – цифры от 00 до 99.

**Формат входных данных**  
На вход программе подается строка текста.

**Формат выходных данных**  
Программа должна вывести «YES», если строка является корректным телефонным номером и «NO» в противном случае.

**Примечание.** Телефонный номер должен содержать только цифры и символ -, а количество цифр в каждой группе должно быть правильным.

n = input().split("-")

c = [len(i) for i in n]

if c == [3, 3, 4] and ''.join(n).isdigit():

print("YES")

elif c == [1, 3, 3, 4] and ''.join(n).isdigit() and n[0] == '7':

print("YES")

else:

print("NO")

## Самый длинный

На вход программе подается строка текста. Напишите программу, **использующую списочное выражение**, которая находит длину самого длинного слова.

**Формат входных данных**  
На вход программе подается строка текста.

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

n=[len(i) for i in input().split()]

print(max(n))

**Молодежный жаргон**

На вход программе подается строка текста. Напишите программу, **использующую списочное выражение**, которая преобразует каждое слово введенного текста в "молодежный жаргон" по следующему правилу:

1. первая буква каждого слова удаляется и ставится в конец слова;
2. затем в конец слова добавляется слог "ки".

**Формат входных данных**  
На вход программе подается строка текста на русском языке.

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

print(\*[i[1:] + i[0] + "ки"for i in input().split()])