**Тема урока: строки**

1. Индексация строк
2. Итерирование строк
3. Решение задач

**Аннотация.** Строковый тип данных. Вспомним основные операции над строками, научимся работать с отдельными символами, а также перебирать (итерировать) символы строк.

**Повторение материала**

Строки в Python используются когда надо работать с текстовыми данными.

**Создание строки.** Для создания строк, мы используем парные кавычки '' или "":

s1 = 'Python'

s2 = "Pascal"

**Считывание строки.** Для считывания текстовых данных в строковую переменную, мы используем функцию input():

s = input() # считали текст

num = int(input()) # считали текст и преобразовали его в целое число

**Пустая строка.** Для создания пустой строки, мы пишем s = '' или s = "". Пустая строка – это аналог числа 0.

**Длина строки.** Для определения длины строки (количества символов), мы используем встроенную функцию len():

s = 'Hello'

n = len(s) # значение переменной равно 5

print(n)

**Конкатенация и умножение на число.** Операторы + и \* можно использовать для строк. Оператор + сцепляет две и более строк. Это называется конкатенацией строк. Оператор \* повторяет строку указанное количество раз.

| **Выражение** | **Результат** |
| --- | --- |
| 'AB' + 'cd' | 'ABcd' |
| 'A' + '7' + 'B' | 'A7B' |
| 'Hi'\* 4 | 'HiHiHiHi' |

**Оператор принадлежности in.** С помощью оператора in, мы можем проверять, находится ли одна строка в составе другой. То есть, является ли одна строка подстрокой другой:

s = 'All you need is love'

if 'love' in s:

print('❤️')

else:

print('💔')

Так как строка s содержит подстроку 'love', то будет выведен смайлик ❤️.

   В Python можно использовать смайлики [emoji](https://pypi.org/project/emoji/" \t "_blank)👍

**Индексация строк**

Очень часто бывает необходимо обратиться к конкретному символу в строке. Для этого в Python используются квадратные скобки [], в которых указывается индекс (номер) нужного символа в строке.

Пусть s = 'Python'. Таблица ниже, показывает как работает индексация:

| **Выражение** | **Результат** | **Пояснение** |
| --- | --- | --- |
| s[0] | P | первый символ строки |
| s[1] | y | второй символ строки |
| s[2] | t | третий символ строки |
| s[3] | h | четвертый символ строки |
| s[4] | o | пятый символ строки |
| s[5] | n | шестой символ строки |

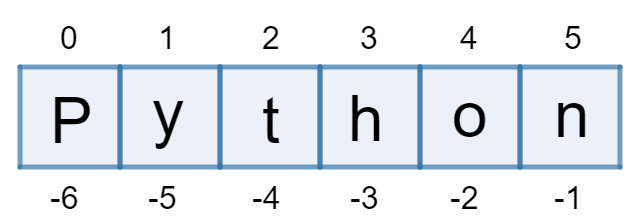
Обратите внимание первый символ строки равен s[0], а не s[1]. В Python индексация начинается с 0, по аналогии с функцией range(n), которая генерировала последовательность натуральных чисел от 0 до n - 1.

В отличие от многих языков программирования, в Python есть возможность работы с отрицательными индексами. Если первый символ строки имеет индекс 0, то последнему элементу присваивается индекс -1.

| **Выражение** | **Результат** | **Пояснение** |
| --- | --- | --- |
| s[-6] | P | первый символ строки |
| s[-5] | y | второй символ строки |
| s[-4] | t | третий символ строки |
| s[-3] | h | четвертый символ строки |
| s[-2] | o | пятый символ строки |
| s[-1] | n | шестой символ строки |

Таким образом, получаем

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Положительные индексы** | 0 | 1 | 2 | 3 | 4 | 5 |
| **Строка** | P | y | t | h | o | n |
| **Отрицательные индексы** | -6 | -5 | -4 | -3 | -2 | -1 |



Частая ошибка у начинающих программистов — обращение по несуществующему индексу в строке.

Например, если s = 'Python', и мы попытаемся обратиться к s[17], то мы получим ошибку:

IndexError: string index out of range

Ошибка возникает, поскольку строка содержит всего 66 символов.

Обратите внимание: если длина строки s равна len(s), то при положительной нумерации слева направо, последний элемент имеет индекс равный len(s) - 1, а при отрицательной индексации справа налево, первый элемент имеет индекс равный -len(s).

**Итерирование строк**

Очень часто нужно просканировать всю строку целиком, обрабатывая каждый ее символ. Для этого удобно использовать цикл for. Напишем программу, которая выводит каждый символ строки на отдельной строке:

s = 'abcdef'

for i in range(len(s)):

print(s[i])

Результатом выполнения такой программы будут строки:

a

b

c

d

e

f

Мы передаем в функцию range() длину строки len(s). В нашем случае длина строки s, равна 66. Таким образом, вызов функции range(len(s)) имеет вид range(6) и переменная цикла i последовательно перебирает все значения от 00 до 55. Это означает, что выражение s[i] последовательно вернет все символы строки s. Такой способ итерации строки удобен, когда нам нужен не только сам элемент s[i], но и его индекс i.

Если нам не нужен индекс самого символа, то мы можем использовать более короткий способ итерации:

s = 'abcdef'

for c in s:

print(c)

Этот цикл пройдет по строке s, придавая переменной цикла c значение каждого символа (!) в отличие от предыдущего цикла, в котором переменная цикла «бегала» по индексам строки.

Обратите внимание на обозначение переменных цикла. В первом цикле мы используем имя i, что соответствует стандартной идеологии наименования переменных цикла. Во втором цикле, мы назвали переменную буквой c – первая буква слова char (символ).

## Одинаковые соседи

На вход программе подается одна строка. Напишите программу, которая определяет сколько в ней одинаковых соседних символов.

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

**Формат выходных данных**  
Программа должна вывести количество одинаковых соседних символов.

n=input()

a=0

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

if n[i]==n[i+1]:

a+=1

print(a)

## Гласные и согласные

На вход программе подается одна строка с буквами русского языка. Напишите программу, которая определяет количество гласных и согласных звуков.

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

**Формат выходных данных**  
Программа должна вывести количество гласных и согласных звуков.

**Примечание.** В русском языке 99 букв, передающих гласные звуки, и 2121 буква, передающая согласные звуки (букву ё игнорируем):

ауоыиэяюе

бвгджзйклмнпрстфхцчшщ

n=input()

a=0

b=0

for i in range(len(n)):

if n[i] in "ауоыиэяюеАУОЫИЭЯЮЁЕ":

a+=1

if n[i] in "бвгджзйклмнпрстфхцчшщБВГДЖЗЙКЛМНПРСТФХЦЧШЩ":

b+=1

print("Количество гласных букв равно", a)

print("Количество согласных букв равно", b)

## Decimal to Binary

На вход программе подается натуральное число, записанное в десятичной системе счисления. Напишите программу, которая переводит данное число в [двоичную систему счисления](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F).

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

**Формат выходных данных**  
Программа должна вывести число записанное в двоичной системе счисления.

n=int(input())

d=''

while n>0:

d=str(n%2)+d

n//=2

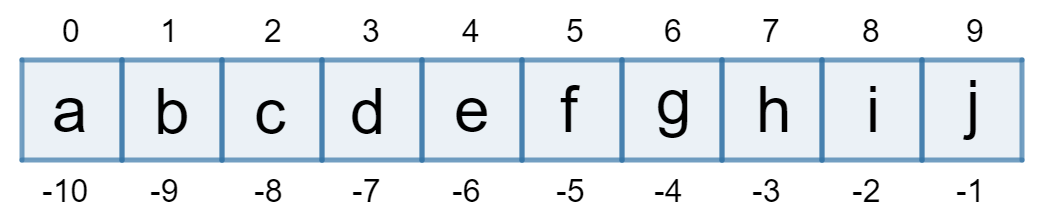
print(d)

## Срезы строк

В предыдущем уроке мы научились работать с конкретными символами строки с помощью индексов []. Иногда нужно бывает работать с целыми частями строки, в таком случае мы используем **срезы (slices)**. Срезы похожи на комбинацию индексации и функции range().

Рассмотрим строку s = 'abcdefghij'.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Положительные индексы** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **Строка** | a | b | c | d | e | f | g | h | i | j |
| **Отрицательные индексы** | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |



С помощью среза мы можем получить несколько символов исходной строки, создав диапазон индексов, разделенных двоеточием s[x:y].

Следующий программный код:

print(s[2:5])

print(s[0:6])

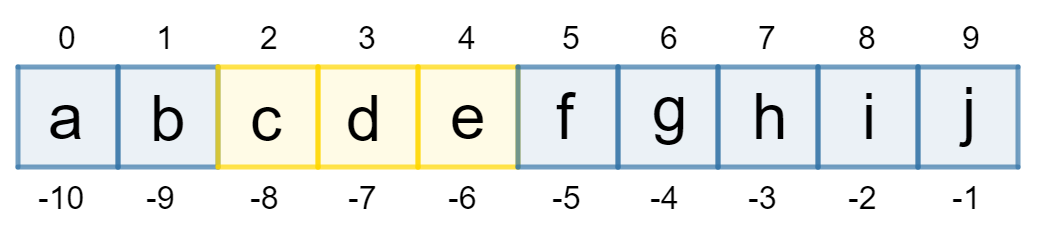
print(s[2:7])

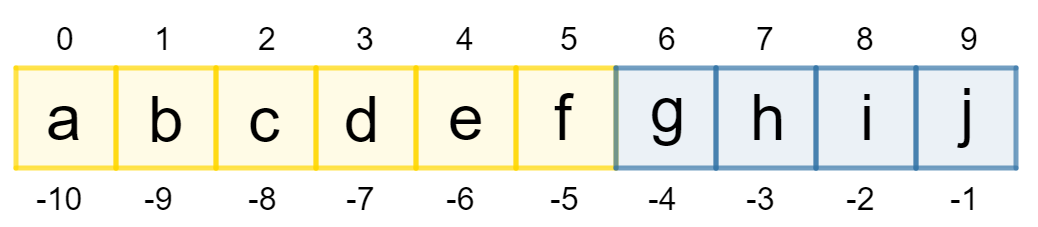
выводит:

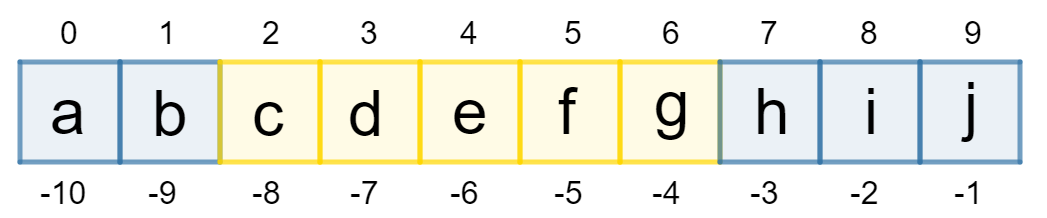
cde

abcdef

cdefg







При построении среза s[x:y] первое число – это то место, где начинается срез (**включительно**), а второе – это место, где заканчивается срез (**невключительно**). Разрезая строки, мы создаем подстроку, которая по сути является строкой внутри другой строки.

### Срез до конца, от начала

Если опустить второй параметр в срезе s[x:] (но поставить двоеточие), то срез берется до конца строки. Аналогично если опустить первый параметр s[:y], то можно взять срез от начала строки. Срез s[:] совпадает с самой строкой s.

Следующий программный код:

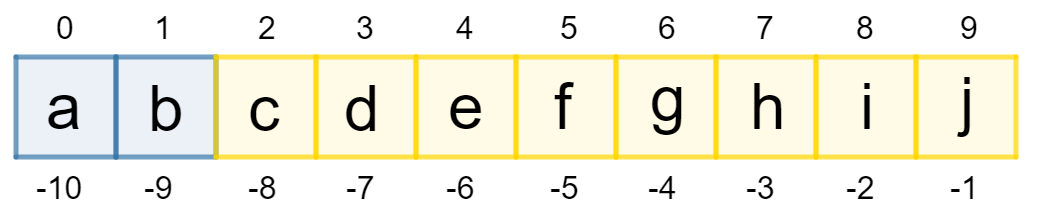
print(s[2:])

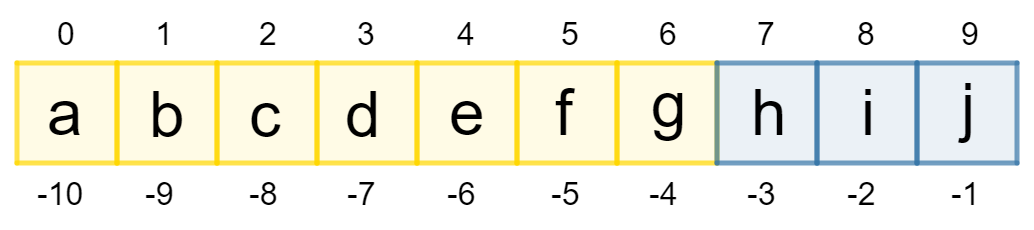
print(s[:7])

выводит:

cdefghij

abcdefg





     Срез s[:] возвращает исходную строку.

### Отрицательные индексы в срезе

Мы также можем использовать отрицательные индексы для создания срезов. Как уже говорилось ранее, отрицательные индексы строки начинаются с -1 и отсчитываются до достижения начала строки. При использовании отрицательных индексов ***первый параметр среза должен быть меньше второго, либо должен быть пропущен***.

Следующий программный код:

print(s[-9:-4])

print(s[-3:])

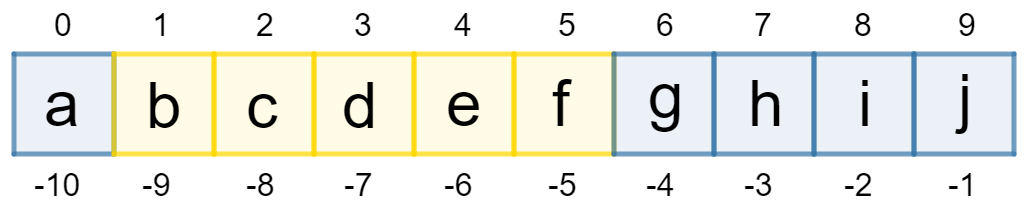
print(s[:-3])

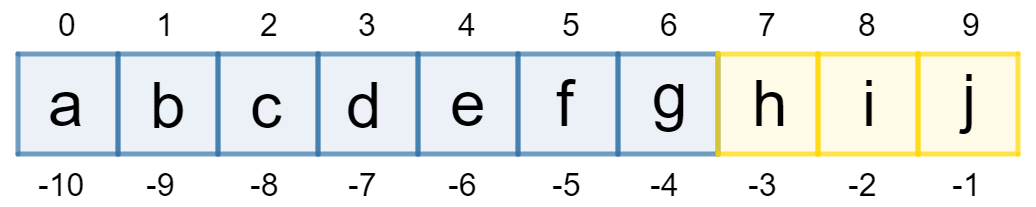
выводит:

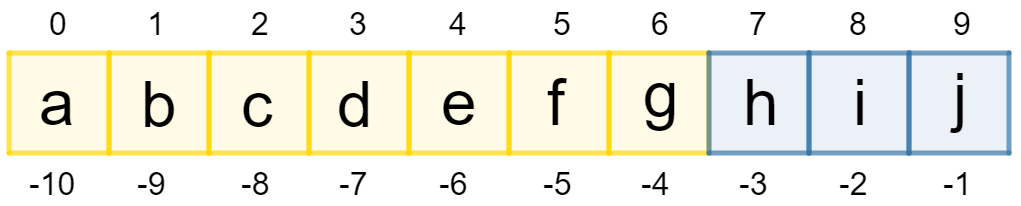
bcdef

hij

abcdefg



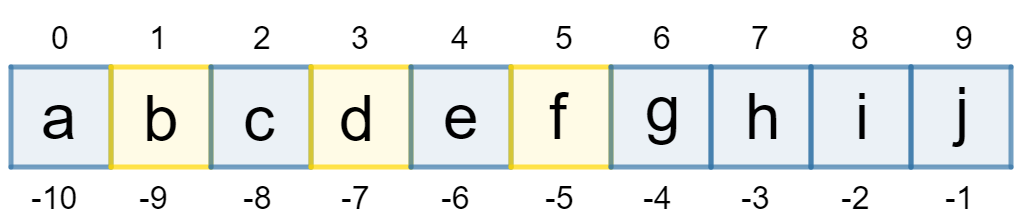




    Удалить из строки последний символ можно при помощи среза s[:-1].

### Шаг среза

Мы можем передать в срез третий необязательный параметр, который отвечает за шаг среза. К примеру, срез s[1:7:2] создаст строку bdf, состоящую из каждого второго символа (индексы 1,3,51,3,5, правая граница не включена в срез).



### Отрицательный шаг среза

Если в качестве шага среза указать **отрицательное число**, то символы будут идти в обратном порядке.

Следующий программный код:

print(s[::-1])

выводит:

jihgfedcba

Следующий программный код:

print(s[1:7:2])

print(s[3::2])

print(s[:7:3])

print(s[::2])

print(s[::-1])

print(s[::-2])

выводит:

bdf

dfhj

adg

acegi

jihgfedcba

jhfdb

### Подводя итог

| s = 'abcdefghij' | |
| --- | --- |
| **Программный код** | **Результат** | **Пояснение** |
| s[2:5] | cde | строка состоящая из символов с индексами 2,3,42,3,4 |
| s[:5] | abcde | первые пять символов строки |
| s[5:] | fghij | строка состоящая из символов с индексами от 5 до конца |
| s[-2:] | ij | последние два символа строки |
| s[:] | abcdefghij | вся строка целиком |
| s[1:7:2] | bdf | строка состоящая из каждого второго символа с индексами от 11 до 66 |
| s[::-1] | jihgfedcba | строка в обратном порядке, так как шаг отрицательный |

## Изменение символа строки по индексу

Предположим, у нас есть строка s = 'abcdefghij' и мы хотим заменить символ с индексом 44 на 'X'. Можно попытаться написать код:

s[4] = 'X'

Однако такой код не работает. В Python строки являются **неизменяемыми**, то есть мы не можем менять их содержимое с помощью индексатора.

Если мы хотим поменять какой-либо символ строки s, мы должны создать новую строку. Следующий код использует срезы и решает поставленную задачу:

s = s[:4] + 'X' + s[5:]

Мы создаем два среза: от начала строки до 55-го символа (не включительно) и с 66-го символа (включительно) по конец строки, а между ними вставляем нужный нам символ, который встанет на 55-ю позицию (44 индекс).

## Примечания

**Примечание 1.** Синтаксис срезов строк очень похож на синтаксис функции range().

**Примечание 2.** Если первый параметр среза больше второго, то срез создает пустую строку.

## Палиндром

На вход программе подается одно слово, записанное в нижнем регистре. Напишите программу, которая определяет, является ли оно палиндромом.

**Формат входных данных**  
На вход программе подается одно слово в нижнем регистре.

**Формат выходных данных**  
Программа должна вывести «YES», если слово является палиндромом, и «NO» в противном случае.

**Примечание.** Палиндром считается слово, которое читается одинаково в обоих направлениях. Например, слово «потоп» является палиндромом.

n=input()

s=n[::-1]

if n==s:

print("YES")

else:

print("NO")

**Делаем срезы 1**

На вход программе подается одна строка. Напишите программу, которая выводит:

1. общее количество символов в строке;
2. исходную строку, повторенную 33 раза;
3. первый символ строки;
4. первые три символа строки;
5. последние три символа строки;
6. строку в обратном порядке;
7. строку с удаленным первым и последним символом.

**Формат входных данных**  
На вход программе подается одна строка, длина которой больше 33 символов.

**Формат выходных данных**  
Программа должна вывести данные в соответствии с условием. Каждое значение выводится на отдельной строке.

n=input()

a=len(n)

print(a)

print(n\*3)

print(n[:1])

print(n[:3])

print(n[-3:])

print(n[::-1])

print(n[1:-1])

**Делаем срезы 2**

На вход программе подается одна строка. Напишите программу, которая выводит:

1. третий символ этой строки;
2. предпоследний символ этой строки;
3. первые пять символов этой строки;
4. всю строку, кроме последних двух символов;
5. все символы с четными индексами;
6. все символы с нечетными индексами;
7. все символы в обратном порядке;
8. все символы строки через один в обратном порядке, начиная с последнего.

**Формат входных данных**  
На вход программе подается одна строка, длина которой больше 5 символов.

**Формат выходных данных**  
Программа должна вывести данные в соответствии с условием. Каждое значение выводится на отдельной строке.

n=input()

print(n[2:3])

print(n[-2:-1])

print(n[:5])

print(n[:-2])

print(n[::2])

print(n[1::2])

print(n[::-1])

print(n[::-2])

## Две половинки

На вход программе подается строка текста. Напишите программу, которая разрежет ее на две равные части, переставит их местами и выведет на экран.

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

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

**Примечание.** Если длина строки нечетная, то длина первой части должна быть на один символ больше.

n=input()

a=len(n)//2+len(n)%2

print(n[a:]+n[:a])

## Методы и функции

Мы уже знакомы с некоторыми встроенными функциями: min(), max(), len(), int(), float() и т.д. Метод — специализированная функция, тесно связанная с объектом. Как и функция, метод вызывается для выполнения отдельной задачи, но он вызывается для определенного объекта и “знает” о своем целевом объекте во время выполнения.

Таким образом: метод — функция, применяемая к объекту. В данном случае к строке. Метод вызывается в виде имя\_объекта.имя\_метода(параметры).

Например, s.find('e') — это применение к строке s метода find с одним параметром 'e'.

Методы строкового типа данных можно разделить на три группы:

1. **Конвертация регистра;**
2. Поиск и замена;
3. Классификация символов.

## Конвертация регистра

Методы в этой группе выполняют преобразование регистра для строк.

### Метод capitalize()

Метод capitalize() возвращает копию строки s, в которой первый символ имеет верхний регистр, а все остальные символы имеют нижний регистр.

Результатом выполнения следующего кода:

s = 'foO BaR BAZ quX'

print(s.capitalize())

будет:

Foo bar baz qux

Символы, не являющиеся буквами алфавита, остаются неизменными. Результатом выполнения следующего кода:

s = 'foo123#BAR#.'

print(s.capitalize())

будет:

Foo123#bar#.

### Метод swapcase()

Метод swapcase() возвращает копию строки s, в которой все символы, имеющие верхний регистр, преобразуются в символы нижнего регистра и наоборот.

 Результатом выполнения следующего кода:

s = 'FOO Bar 123 baz qUX'

print(s.swapcase())

будет:

foo bAR 123 BAZ Qux

### Метод title()

Метод title() возвращает копию строки s, в которой первый символ каждого слова переводится в верхний регистр.

Результатом выполнения следующего кода:

s = 'the sun also rises'

print(s.title())

будет:

The Sun Also Rises

Этот метод использует довольно простой алгоритм: он не пытается различить важные и неважные слова и не обрабатывает аббревиатуры и апострофы. Результатом выполнения следующего кода:

s = "what's happened to ted's IBM stock?"

print(s.title())

будет:

What'S Happened To Ted'S Ibm Stock?

### Метод lower()

Метод lower() возвращает копию строки s, в которой все символы имеют нижний регистр.

Результатом выполнения следующего кода:

s = 'FOO Bar 123 baz qUX'

print(s.lower())

будет:

foo bar 123 baz qux

### Метод upper()

Метод upper() возвращает копию строки s, в которой все символы имеют верхний регистр.

Результатом выполнения следующего кода:

s = 'FOO Bar 123 baz qUX'

print(s.upper())

будет:

FOO BAR 123 BAZ QUX

Одно очень важное замечание о методах данной категории состоит в том, что они не изменяют исходную строку. Если вы хотите изменить строку s, нужно написать код: s = s.lower(). На самом деле тут вы создаёте совсем другой объект в памяти компьютера, просто он со старым названием s.

## Примечание

Англо-русский словарик:  
capitalize — писать прописными буквами, закрепить.  
swapcase — обменять регистр. swap — гл. обмениваться, case — случай, регистр, падеж, дело, расследование...  
title — заголовок, титул.  
lower — нижний.  
upper — верхний.

## Хороший оттенок

На вход программе подается строка текста. Напишите программу, которая определяет является ли оттенок текста хорошим или нет. Текст имеет хороший оттенок, если содержит подстроку «хорош» во всевозможных регистрах.

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

**Формат выходных данных**  
Программа должна вывести «YES» если текст имеет хороший оттенок и «NO» в противном случае.

**Примечание.** Текст содержащий хорош, ХОРОШ, Хорош, хОРОШ и т.д. имеет хороший оттенок.

n=input()

if "хорош" in n.lower():

print("YES")

else:

print("NO")

## Нижний регистр

На вход программе подается строка. Напишите программу, которая подсчитывает количество буквенных символов в нижнем регистре.

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

**Формат выходных данных**  
Программа должна вывести количество буквенных символов в нижнем регистре.

n=input()

a=0

for i in range(len(n)):

if n[i]!=n[i].title():

a+=1

print(a)

## Поиск и замена

Методы поиска и замены строк внутри других строк.

Каждый метод в этой группе поддерживает необязательные аргументы <start> и <end>. Как и в строковых срезах действие метода ограничено частью исходной строки, начинающейся с позиции символа <start> и продолжающейся вплоть до позиции символа <end>, но не включающей ее. Если параметр <start> указан, а параметр <end> нет, то метод применяется к части исходной строки от <start> до конца строки. Если параметры не заданы, то подразумевается, что <start> = 0 , <end> = len(s).

### Метод count()

Метод count(<sub>, <start>, <end>) считает количество **непересекающихся** вхождений подстроки <sub> в исходную строку s.

Результатом выполнения следующего кода:

s = 'foo goo moo'

print(s.count('oo'))

print(s.count('oo', 0, 8)) # подсчет с 0 по 7 символ

будет:

3

2

### Метод startswith()

Метод startswith(<suffix>, <start>, <end>) определяет **начинается**ли исходная строка s подстрокой <suffix>. Если исходная строка начинается с подстроки <suffix>,метод возвращает значение True, а если нет, то  False.

Результатом выполнения следующего кода:

s = 'foobar'

print(s.startswith('foo'))

print(s.startswith('baz'))

будет:

True

False

### Метод endswith()

Метод endswith(<suffix>, <start>, <end>) определяет **оканчивается**ли исходная строка s подстрокой <suffix>. Метод возвращает значение True если исходная строка оканчивается на подстроку <suffix> и False в противном случае.

Результатом выполнения следующего кода:

s = 'foobar'

print(s.endswith('bar'))

print(s.endswith('baz'))

будет:

True

False

### Методы find(), rfind()

Метод find(<sub>, <start>, <end>) находит **индекс первого вхождения** подстроки <sub> в исходной строке s. Если строка s не содержит подстроки <sub>, то метод возвращает значение -1. Мы можем использовать данный метод наравне с оператором in для проверки: содержит ли заданная строка некоторую подстроку или нет.

Результатом выполнения следующего кода:

s = 'foo bar foo baz foo qux'

print(s.find('foo'))

print(s.find('bar'))

print(s.find('qu'))

print(s.find('python'))

будет:

0

4

20

-1

Метод rfind(<sub>, <start>, <end>) идентичен методу find(<sub>, <start>, <end>), за тем исключением, что он ищет первое вхождение подстроки <sub> начиная с конца строки s.

### Методы index(), rindex()

Метод index(<sub>, <start>, <end>) идентичен методу find(<sub>, <start>, <end>), за тем исключением, что он **вызывает ошибку**  ValueError: substring not found во время выполнения программы, если подстрока <sub> не найдена.

Метод rindex(<sub>, <start>, <end>) идентичен методу index(<sub>, <start>, <end>), за тем исключением, что он ищет первое вхождение подстроки <sub> начиная с конца строки s.

Методы find() и rfind() являются более безопасными чем index() и rindex(), так как не приводят к возникновению ошибки во время выполнения программы.

### Метод strip()

Метод strip() возвращает копию строки s у которой удалены все пробелы стоящие **в начале и конце** строки.

Результатом выполнения следующего кода:

s = ' foo bar foo baz foo qux '

print(s.strip())

будет:

foo bar foo baz foo qux

### Метод lstrip()

Метод lstrip() возвращает копию строки s у которой удалены все пробелы стоящие **в начале** строки.

Результатом выполнения следующего кода:

s = ' foo bar foo baz foo qux '

print(s.lstrip())

будет:

foo bar foo baz foo qux⎵ ⎵ ⎵ ⎵ ⎵ ⎵

### Метод rstrip()

Метод rstrip() возвращает копию строки s у которой удалены все пробелы стоящие **в конце** строки.

Результатом выполнения следующего кода:

s = ' foo bar foo baz foo qux '

print(s.rstrip())

будет:

⎵ ⎵ ⎵ ⎵ ⎵ ⎵foo bar foo baz foo qux

Методы strip(), lstrip(), rstrip() могут принимать на вход опциональный аргумент<chars>. Необязательный аргумент <chars>– это строка, которая определяет набор символов для удаления.

### Метод replace()

Метод replace(<old>, <new>) возвращает копию s **со всеми** вхождениями подстроки <old>, замененными на <new>.

Результатом выполнения следующего кода:

s = 'foo bar foo baz foo qux'

print(s.replace('foo', 'grault'))

будет:

grault bar grault baz grault qux

Метод replace() может принимать опциональный третий аргумент <count>,  который определяет количество замен.

Результатом выполнения следующего кода:

s = 'foo bar foo baz foo qux'

print(s.replace('foo', 'grault', 2))

будет:

grault bar grault baz foo qux

## Количество слов

На вход программе подается строка текста, состоящая из слов, разделенных ровно одним пробелом. Напишите программу, которая подсчитывает количество слов в ней.

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

**Формат выходных данных**  
Программа должна вывести количество слов.

**Примечание 1.** Строка текста не содержит пробелов в начале и конце.

**Примечание 2.** Используйте для решения задачи метод count.

n=input()

if len(n)==0:

print(0)

elif n.count(" ")==0:

print(1)

else:

print(n.count(" ")+1)

## Минутка генетики

На вход программе подается строка генетического кода, состоящая из букв А (аденин), Г (гуанин), Ц (цитозин), Т (тимин). Напишите программу, которая подсчитывает сколько аденина, гуанина, цитозина и тимина входит в данную строку генетического кода.

**Формат входных данных**  
На вход программе подается строка генетического кода, состоящая из символов А, Г, Ц, Т, а, г, ц, т.

**Формат выходных данных**  
Программа должна вывести сколько гуанина, тимина, цитозина, аденина входит в данную строку генетического кода.

**Примечание.** Строка не содержит символов, кроме как А, Г, Ц, Т, а, г, ц, т.

n=input()

print("Аденин:", n.count("А")+n.count("а"))

print("Гуанин:", n.count("Г")+n.count("г"))

print("Цитозин:", n.count("Ц")+n.count("ц"))

print("Тимин:", n.count("Т")+n.count("т"))

## Очень странные дела

Джим Хоппер с помощью радиоприемника пытается получить сообщение Оди. На приемник ему поступает �*n* различных последовательностей кода Морзе. Декодировав их, он получает последовательности из цифр и букв строчного латинского алфавита. При этом только **в сообщениях Оди** содержится число 1111, причем минимум 33 раза. Помогите определить Джиму количество сообщений от Оди.

**Формат входных данных**  
В первой строке подаётся число �*n* – количество сообщений, в последующих �*n* строках вводятся строки, содержащие латинские строчные буквы и цифры.

**Формат выходных данных**  
Программа должна вывести количество строк в которых содержится число 1111 минимум 33 раза.

**Примечание.** В сообщениях Оди числа 1111 необязательно должны быть разделены другими символами, нужно подсчитать вхождения последовательности символов "1111", т.е. например в строке "111111" содержится одна такая последовательность, в то время как в "11111111" их уже две.

n=int(input())

s=0

for \_ in range(n):

a=input()

if a.count("11")>=3:

s+=1

print(s)

## Количество цифр

На вход программе подается строка текста. Напишите программу, которая подсчитывает количество цифр в данной строке.

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

**Формат выходных данных**  
Программа должна вывести количество цифр в данной строке.

n=input()

s=0

for i in range(10):

s+=n.count(str(i))

print(s)

## .com or .ru

На вход программе подается строка текста. Напишите программу, которая проверяет, что строка заканчивается подстрокой .com или .ru.

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

**Формат выходных данных**  
Программа должна вывести «YES» если введенная строка заканчивается подстрокой .com или .ru и «NO» в противном случае.

n=input()

f="NO"

if n.endswith((".com", ".ru")):

f="YES"

print(f)

## Самый частотный символ

На вход программе подается строка текста. Напишите программу, которая выводит на экран символ, который появляется наиболее часто.

**Формат входных данных**  
На вход программе подается строка текста. Текст может содержать строчные и заглавные буквы английского и русского алфавита, а также цифры.

**Формат выходных данных**  
Программа должна вывести символ, который появляется наиболее часто.

**Примечание 1.** Если таких символов несколько, следует вывести последний по порядку символ.

**Примечание 2.** Следует различать заглавные и строчные буквы, а также буквы русского и английского алфавита.

n=input()

a=0

b=""

for i in range(len(n)):

if n.count(n[i])>=a:

a=n.count(n[i])

b=n[i]

print(b)

## Первое и последнее вхождение

На вход программе подается строка текста. Если в этой строке буква «f» встречается только один раз, выведите её индекс. Если она встречается два и более раза, выведите индексы её первого и последнего вхождения на одной строке, разделенные символом пробела. Если буква «f» в данной строке не встречается, следует вывести «NO».

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

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

n=input()

if n.count("f")==0:

print("NO")

elif n.count("f")==1:

print(n.find("f"))

else:

print(n.find("f"), n.rfind("f"))

## Удаление фрагмента

На вход программе подается строка текста, в которой буква «h» встречается минимум два раза. Напишите программу, которая удаляет из этой строки первое и последнее вхождение буквы «h», а также все символы, находящиеся между ними.

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

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

n=input()

print(n[:n.find("h")]+n[n.rfind("h")+1:])

## Классификация символов

Методы в этой группе классифицируют строку на основе содержащихся в ней символов.

### Метод isalnum()

Метод isalnum() определяет, состоит ли исходная строка из буквенно-цифровых символов. Метод возвращает значение True если исходная строка является непустой и состоит **только** из буквенно-цифровых символов и False в противном случае.

Результатом выполнения следующего кода:

s1 = 'abc123'

s2 = 'abc$\*123'

s3 = ''

print(s1.isalnum())

print(s2.isalnum())

print(s3.isalnum())

будет:

True

False

False

### Метод isalpha()

Метод isalpha() определяет, состоит ли исходная строка из буквенных символов. Метод возвращает значение True если исходная строка является непустой и состоит **только** из буквенных символов и False в противном случае.

Результатом выполнения следующего кода:

s1 = 'ABCabc'

s2 = 'abc123'

s3 = ''

print(s1.isalpha())

print(s2.isalpha())

print(s3.isalpha())

будет:

True

False

False

### Метод isdigit()

Метод isdigit() определяет, состоит ли исходная строка **только** из цифровых символов. Метод возвращает значение True если исходная строка является непустой и состоит **только** из цифровых символов и False в противном случае.

Результатом выполнения следующего кода:

s1 = '1234567'

s2 = 'abc123'

s3 = ''

print(s1.isdigit())

print(s2.isdigit())

print(s3.isdigit())

будет:

True

False

False

### Метод islower()

Метод islower() определяет, являются ли **все** буквенные символы исходной строки строчными (имеют нижний регистр). Метод возвращает значение True если все буквенные символы исходной строки являются строчными и False в противном случае. **Все неалфавитные символы игнорируются!**

Результатом выполнения следующего кода:

s1 = 'abc'

s2 = 'abc1$d'

s3 = 'Abc1$D'

print(s1.islower())

print(s2.islower())

print(s3.islower())

будет:

True

True

False

### Метод isupper()

Метод isupper() определяет, являются ли **все** буквенные символы исходной строки заглавными (имеют верхний регистр). Метод возвращает значение True если все буквенные символы исходной строки являются заглавными и False в противном случае. **Все неалфавитные символы игнорируются!**

Результатом выполнения следующего кода:

s1 = 'ABC'

s2 = 'ABC1$D'

s3 = 'Abc1$D'

print(s1.isupper())

print(s2.isupper())

print(s3.isupper())

будет:

True

True

False

### Метод isspace()

Метод isspace() определяет, состоит ли исходная строка **только** из пробельных символов. Метод возвращает значение True если строка состоит только из пробельных символов и False в противном случае.

Результатом выполнения следующего кода:

s1 = ' '

s2 = 'abc1$d'

print(s1.isspace())

print(s2.isspace())

будет:

True

False

## Форматирование строк

Хранить строки в переменных удобно, но часто бывает необходимо **собирать строки** из других объектов (строк, чисел и т.д.) и выполнять с ними нужные манипуляции. Для этой цели можно воспользоваться механизмом **форматирования строк**.

Рассмотрим следующий код:

age = 27

txt = 'My name is Timur, I am ' + age

print(txt)

Такой код приводит к ошибке во время выполнения программы, поскольку мы пытаемся сложить число и строку. Для решения такой проблемы мы можем использовать функцию str, которая преобразует числовое значение в строку:

age = 27

txt = 'My name is Timur, I am ' + str(age)

print(txt)

Такой код работает, однако в Python предпочтительным способом форматирования считается использование метода format. Предыдущую программу можно переписать в виде:

age = 27

txt = 'My name is Timur, I am {}'.format(age)

print(txt)

Мы передаем необходимые параметры методу format, а Python форматирует указанную строку и помещает их в строку на место заполнителей {}. Мы можем создавать сколько угодно заполнителей в строке:

age = 27

name = 'Timur'

profession = 'math teacher'

txt = 'My name is {}, I am {}, I work as a {}'.format(name, age, profession)

print(txt)

Для наглядности и гибкости форматирования мы можем использовать порядковый номер в заполнителе: {0}, {1}, {2},.... Такой номер определяет позицию параметра, переданного методу format (нумерация начинается с нуля):

age = 27

name = 'Timur'

profession = 'math teacher'

txt = 'My name is {0}, I am {1}, I work as a {2}'.format(name, age, profession)

print(txt)

Параметр name встает в {0} заполнитель, параметр age встает в {1} заполнитель и т.д. Мы можем использовать одно и тоже число в нескольких заполнителях

name = 'Timur'

txt = 'My name is {0}-{0}-{0}'.format(name)

print(txt)

Результатом выполнения такого кода будет:

My name is Timur-Timur-Timur

## f-строки

Метод format хорошо справляется с задачей форматирования строк, однако если параметров много, то код может показаться немного избыточным:

first\_name = 'Timur'

last\_name = 'Guev'

age = 27

profession = 'math teacher'

affiliation = 'BeeGeek'

print('Hello, {0} {1}. You are {2}. You are a {3}. You were a member of {4}'

.format(first\_name, last\_name, age, profession, affiliation))

В Python 3.6 появилась новая разновидность строк — так называемые f-строки. Если поставить перед строкой префикс f, в заполнители можно будет включить код, например имя переменной. Предыдущий код можно записать в виде:

first\_name = 'Timur'

last\_name = 'Guev'

age = 27

profession = 'math teacher'

affiliation = 'BeeGeek'

print(f'Hello, {first\_name} {last\_name}. You are {age}. You are a {profession}. You were a member of {affiliation}')

На место заполнителя {first\_name} встает значение переменной first\_name, на место  заполнителя {last\_name} встает значение переменной last\_name и т.д.

Подробная документация с форматированием строк. https://docs.python.org/3/library/string.html#custom-string-formatting

Дополните приведенный код, используя форматирование строк с помощью метода format, так чтобы он вывел текст:

«In 2010, someone paid 10k Bitcoin for two pizzas.» (без кавычек).

s = 'In {0}, someone paid {1} {2} for two pizzas.'.format("2010","10k","Bitcoin")

print(s)

Дополните приведенный код, используя форматирование строк с помощью f-строк, так чтобы он вывел текст:

«In 2010, someone paid 10K Bitcoin for two pizzas.» (без кавычек).

year = 2010

amount = '10K'

currency = 'Bitcoin'

print(f'In {year}, someone paid {amount} {currency} for two pizzas.')

# Тема урока: представление строк в памяти компьютера, ASCII и Unicode

1. Представление строк в памяти компьютера
2. Таблица символов ASCII
3. Таблица символов Unicode
4. Функция ord()
5. Функция chr()

**Аннотация.** Представление строк в памяти компьютера.

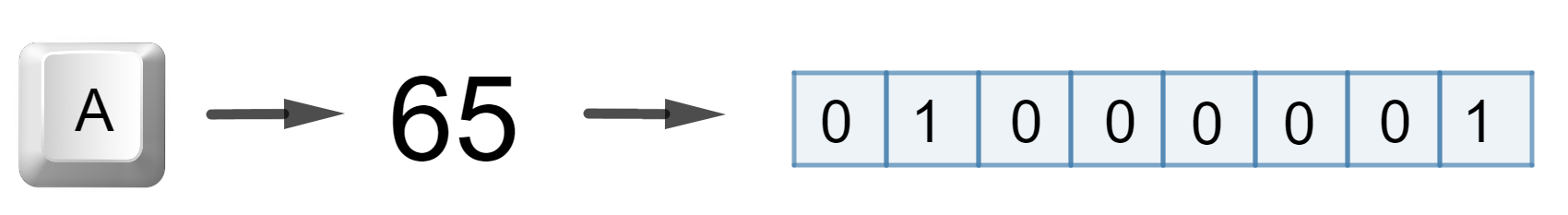
## Представление строк в памяти компьютера

Любой набор данных в [оперативной памяти](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) компьютера должен храниться в виде двоичного числа. Это относится и к строкам, которые состоят из символов (буквы, знаки препинания и т.д.). Когда символ сохраняется в памяти, он сначала преобразуется в цифровой код. И затем этот цифровой код сохраняется в памяти как двоичное число.

За прошедшие годы для представления символов в памяти компьютера были разработаны различные схемы кодирования. Исторически самой важной из этих схем кодирования является схема кодирования ASCII (American Standard Code for Information Interchange – американский стандартный код обмена информацией).

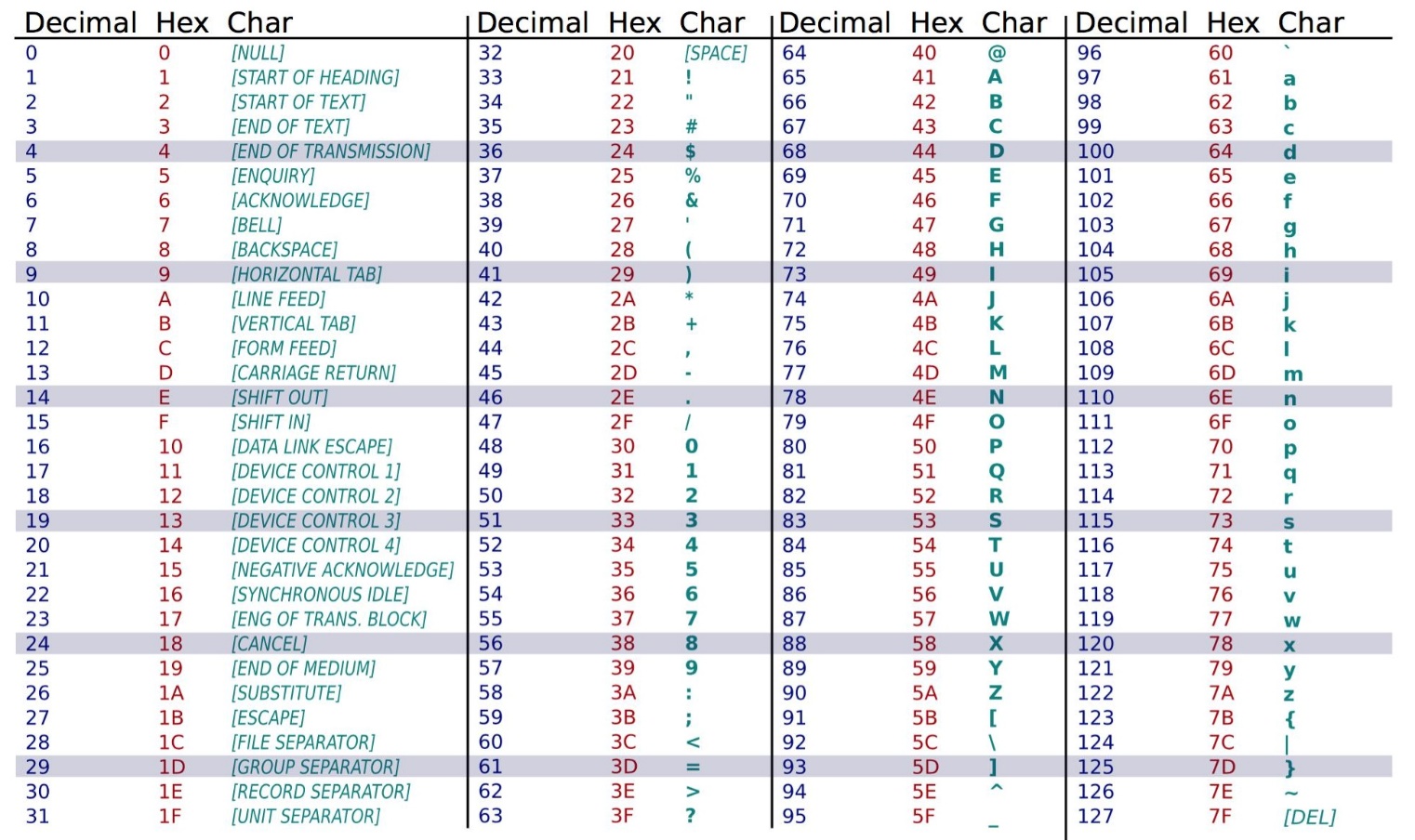
### Таблица символов ASCII

ASCII представляет собой набор из 128 цифровых кодов, которые обозначают английские буквы, различные знаки препинания и другие символы. Например, код ASCII для прописной английской буквы «А» (латинской) равняется 65. Когда на компьютерной клавиатуре вы набираете букву «А» в верхнем регистре, в памяти сохраняется число 65 (как двоичное число, разумеется).



Код ASCII для английской «В» в верхнем регистре равняется 66, для «С» в верхнем регистре – 67 и т. д. **На один символ в ASCII отводится ровно 7 бит.**  
    Аббревиатура ASCll произносится «аски».

### ASCII table



Набор символов ASCII был разработан в начале 1960-х годов и в конечном счете принят почти всеми производителями компьютеров. Однако схема кодирования ASCII имеет ограничения, потому что она определяет коды только для 128 символов. Для того чтобы это исправить, в начале 1990-х годов был разработан набор символов Юникода (Unicode). Это широкая схема кодирования, совместимая с ASCII, которая может также представлять символы многих языков мира. Сегодня Юникод быстро становится стандартным набором символов, используемым в компьютерной индустрии.

### Таблица символов Unicode

Таблица символов Юникод представляет собой набор цифровых символов, которые включают в себя знаки почти всех письменных языков мира. Стандарт предложен в 1991 году некоммерческой организацией «Консорциум Юникода». Применение этого стандарта позволяет закодировать очень большое число символов из разных систем письменности: в документах, закодированных по стандарту Юникод, могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, символы музыкальной нотации.

Стандарт состоит из двух основных частей: универсального набора символов и семейства кодировок (Unicode transformation format, UTF). Универсальный набор символов перечисляет допустимые по стандарту Юникод символы и присваивает каждому символу код в виде неотрицательного целого числа. Семейство кодировок определяет способы преобразования кодов символов для хранения на компьютере и передачи.

В Юникод все время добавляются новые символы, а сам размер этой таблицы не ограничен и будет только расти, поэтому сейчас при хранении в памяти одного юникод-символа может потребоваться от 1 до 8 байт. Отсутствие ограничений привело к тому, что стали появляться символы на все случаи жизни.  
    В Python строки хранятся в виде последовательности юникод символов.

## Примечания

**Примечание 1.** Официальный сайт таблицы символов [Unicode](https://home.unicode.org/" \t "_blank).

**Примечание 2.** Юникод — это не кодировка. Это именно таблица символов. То, как символы с соответствующими кодами будут храниться в памяти компьютера, зависит от конкретной кодировки, базирующейся на Юникоде,  например UTF-8.

**Примечание 3.** Первые 128 кодов таблицы символов Unicode совпадают с ASCII.

## ****Функция ord****

Функция ord позволяет определить код некоторого символа в таблице символов Unicode. Аргументом данной функции является одиночный символ.

Результатом выполнения следующего кода:

num1 = ord('A')

num2 = ord('B')

num3 = ord('a')

print(num1, num2, num3)

будет:

65 66 97

Обратите внимание, что функция ord принимает именно **одиночный символ**. Если попытаться передать строку, содержащую более одного символа:

num = ord('Abc')

print(num)

мы получим ошибку времени выполнения:

TypeError: ord() expected a character, but string of length 3 found

    Название функции ord происходит от английского слова order — порядок.

## ****Функция chr****

Функция chr позволяет определить по коду символа сам символ. Аргументом данной функции является численный код.

Результатом выполнения следующего кода:

chr1 = chr(65)

chr2 = chr(75)

chr3 = chr(110)

print(chr1, chr2, chr3)

будет:

A K n

     Название функции chr происходит от английского слова char — символ.

Функции ord и chr часто работают в паре. Мы можем использовать следующий код для вывода всех заглавных букв английского алфавита:

for i in range(26):

print(chr(ord('A') + i))

Вызов функции ord('A') возвращает код символа «A», который равен 65. Далее на каждой итерации цикла, к данному коду прибавляется значение переменной i = 0, 1, 2, ..., 25, а затем полученный код преобразуется в символ с помощью вызова функции chr.

## Примечания

**Примечание.** Функции ord и chr являются **взаимнообратными**. Для них выполнены равенства:

chr(ord('A')) = 'A', ord(chr(65)) = 65.

## Символы в диапазоне

На вход программе подаются два числа �*a* и �*b*. Напишите программу, которая для каждого кодового значения в диапазоне от �*a* до �*b* (включительно), выводит соответствующий ему символ из таблицы символов Unicode.

**Формат входных данных**  
На вход программе подается два натуральных числа, каждое на отдельное строке.

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

a,b=int(input()),int(input())

for i in range(a,b+1):

print(chr(i),end=" ")

## Простой шифр

На вход программе подается строка текста. Напишите программу, которая переводит каждый ее символ в соответствующий ему код из таблицы символов Unicode.

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

**Формат выходных данных**  
Программа должна вывести кодовые значения символов строки разделенных одним символом пробела.

n=input()

for i in range(len(n)):

print(ord(n[i]),end=" ")

## Шифр Цезаря 🌶️

Легион Цезаря, созданный в 2323 веке на основе Римской Империи не изменяет древним традициям и использует шифр [Цезаря](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80_%D0%A6%D0%B5%D0%B7%D0%B0%D1%80%D1%8F). Это их и подвело, ведь данный шифр очень простой. Однако в постапокалипсисе люди плохо знают все тонкости довоенного мира, поэтому ученые из НКР не могут понять, как именно нужно декодировать данные сообщения. Напишите программу для декодирования этого шифра.

**Формат входных данных**  
В первой строке дается число � (1≤ �≤ 25)*n* (1≤ *n*≤ 25) – сдвиг, во второй строке даётся закодированное сообщение в виде строки со строчными латинскими буквами.

**Формат выходных данных**  
Программа должна вывести одну строку – декодированное сообщение. Обратите внимание, что нужно декодировать сообщение, а не закодировать.

a=int(input())

b=input()

for i in range(len(b)):

n = ord(b[i])-a

if n < 97:

n = 122 - (96 - n)

print(chr(n), end="")