



# Составление алгоритмов. Применение машинной логики к задачам поиска данных. Оценка времени работы алгоритмов, эффективность кода

Турашова Анна Николаевна  
Преподаватель  
[anna1turashova@gmail.com](mailto:anna1turashova@gmail.com)  
Telegram: @anna1tur



# Строки



# Строки

Строка – неизменяемая последовательность символов. В программе конкретные значения строк заключены в кавычки или апострофы.

`n = 'Книга "Война и мир"'`

`s = "Привет!"`

Пустая строка `t = ""`

Ввод `s = input ("Введите имя: ")`

Вывод `print (s)`



# Преобразования типов

`str(x)` любое значение в строку

`int(s)` строка в целое число

`float(s)` строка в действительное число

Форматные строки

```
name = "Марина"
```

```
age = 20
```

```
print(f"Я - {name}. Мне {age} лет.")
```



# Операции со строками

Сложение (сцепление, конкатенация)

```
s1 = "Привет"
```

"Привет, Вася!"

```
s2 = "Вася"
```

```
s = s1 + ", " + s2 + "!"
```

Повторение (умножение на целое число)

```
s = "*" * 10
```

```
t = "мяу " * 3
```

Проверка входит ли одна строка в другую

```
if "*" in s:  
    print("есть звезда")
```



# Сравнения строк

Используется лексикографическое сравнение:  
из двух строк больше строка, которая стоит  
дальше в словаре.

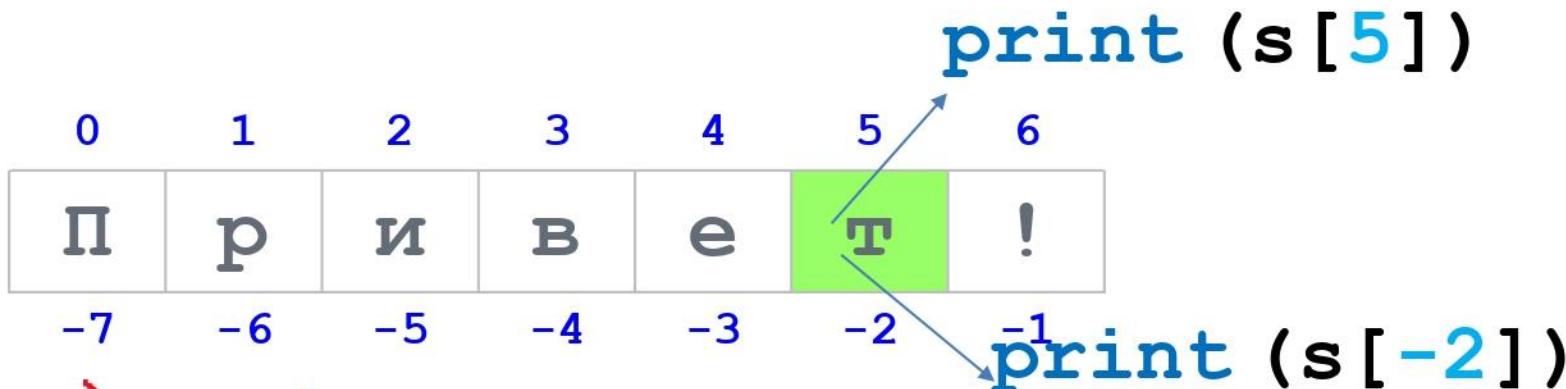
**ПАР < Пар < пАр < пар < парк**

торт ? тортик

понедельник ? вторник

# Индексация символов

Строчку можно рассматривать как список символов



~~`print(s[7])`~~ – ошибка. Индекса 7 нет.

~~`s[0] = "п"`~~ – ошибка. Строки не изменяются.

Длина строки `len(s)`.

Если `n = len(s)`, то индексы будут от 0 до `n-1`.



# Перебор символов

Строчку можно рассматривать как список символов. Символы перебираются циклом for.

Посчитаем количество гласных в строке

```
text = 'hello, my dear friends !'  
count = 0  
for letter in text:  
    if letter in 'aeiouy':  
        count += 1  
print(count)
```



# Перебор индексов

Используем функции range и len.

```
text = 'hello, my dear friends!'
count = 0
for i in range(len(text)):
    if text[i] in 'aeiouy':
        count += 1
print(count)
```



# Срезы строк

Срез – фрагмент строки.

```
s = "0123456789"
```

```
s1 = s[3:8]           # "34567"
```

```
s2 = s[:8]            # "01234567"
```

```
s3 = s[3:]             # "3456789"
```

```
s4 = s[::-2]           # "02468"
```

```
s5 = s[::-1]           # "9876543210"
```



# Удаление и вставка фрагментов

Используем срезы и сложение

```
s = "0123456789"
```

```
# удалить [3:9]
```

```
s1 = s[:3] + s[9:] # "0129"
```

```
# вставить строку после 3 символов
```

```
s2 = s[:3] + "ABC" + s[3:]
```

```
"012ABC3456789"
```



# Методы строк

Метод — это функция, применяемая к объекту, в данном случае — к строке. Вызывается так:

`строка . метод (параметры)`



# Методы поиска

Метод `find` находит номер первого вхождения подстроки в строку.

Если не найдено, результат -1.

Метод `rfind` ищет с конца.

Метод `count` считает количество вхождений.

```
s = "абракадабра"  
print(s.find("бр"))      # 1  
print(s.rfind("бр"))    # 8  
print(s.count("а"))     # 5
```



# Изменение строки

Метод `replace` заменяет все вхождения одной строки на другую.

Методы не изменяют строку, а возвращают новую!

Для изменения нужно переприсваивание.

Пример. Удалить буквы а из строки.

`s = "абракадабра"`

`s = s.replace("а", "")`



# Проверка строк

`s.startswith(s2) s.endswith(s2)`

Проверка, что `s` начинается с `s2` или оканчивается на `s2`

Пример. `'abracadabra'.startswith('abra')`

`s.isdigit() s.isalpha() s.isalnum()`

Проверка, что в строке `s` все символы — цифры, буквы, цифры или буквы соответственно

Пример. `'abc'.isalpha()`

`s.islower() s.isupper()`

Проверка, что в строке `s` не встречаются большие буквы, маленькие буквы.

Пример. `'hello!'.islower()`

`'123PYTHON'.isupper()`



# Преобразования строк

`s.lower() s.upper()`

Строка s, в которой все буквы приведены к верхнему или нижнему регистру

Пример. `'abracadabra'.upper() == 'ABRACADABRA'`

`s.capitalize()`

Строка s, в которой первая буква — заглавная

Пример. `'abracadabra'.capitalize() == 'Abracadabra'`

`s.lstrip() s.rstrip() s.strip()`

Строка s, у которой удалены символы пустого пространства (пробелы, табуляции) в начале, в конце или с обеих сторон.

Пример. `' Привет! '.strip() == 'Привет! '`



# Цепочки вызовов

```
s = s.strip().lower().replace('ё', 'е')
```

Выдаст строку s, в которой убраны начальные и конечные пробелы, все буквы сделаны маленькими, после чего убраны все точки над Ё.

' Зелёный клён ' -> 'зеленый клен' .

# Коды символов

Символы в строках имеют коды – номера в таблице Юникод

<https://unicode-table.com/ru/>

```
print ('\u2603')
```



Для того чтобы узнать код некоторого символа, существует функция `ord` (от `order` — «порядок»).

```
ord('Б') 1041
```

Зная код, всегда можно получить соответствующий ему символ. Для этого существует функция `chr` (от `character` — «символ»):

```
chr(1041) 'Б'
```

```
for i in range(26):  
    print(chr(ord('A') + i))
```



# Шифр Цезаря

Как известно, Цезарь тоже пользовался шифрованием сообщений, причем у него был свой способ. Сначала выбирается шаг шифрования (число), а затем все буквы послания заменяются на буквы, отстоящие от них в алфавите на шаг шифрования. Например, при шаге шифрования 3 (таким чаще всего пользовался Цезарь), буква А заменяется на букву Г, буква Б – на букву Д.

Обратите внимание, что алфавит «зациклен», то есть при сдвиге буквы Я на шаг 3 получится буква В.

Напишите программу, которая будет зашифровывать послание с помощью шифра Цезаря с заданным шагом шифрования.

## Формат ввода

Две строки. Первая содержит шаг шифрования, вторая – послание.

## Формат вывода

Строка с зашифрованным посланием.

## Пример 1

Ввод	Выход	Чаты
3	ГДЕ	
АБВ		

## Пример 2

Ввод	Выход
5	Те йзухк чхезе, те чхезк йхузе!
На дворе трава, на траве дрова!	



# Списки



# Списки

- **Списком** в языке Python называется упорядоченная последовательность элементов произвольных типов
- Списки имеют **произвольную** длину, которая может изменяться, также **могут изменяться значения** элементов

```
data1 = [5, 'a', ['python'], 20]
```

```
data2 = [5, 6, 7, 8]
```

```
data3 = ['red', 'green', 'black']
```

```
data4= [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```



# Создание и заполнение

Существует несколько способов создать и заполнить список

- 1) Создать пустой список

```
data = []
```

- 2) Перечисление его элементов

```
data1 = [3, 4, 5, 6, 88, 99]
```

- 3) С помощью встроенной функции list

```
data1 = list('список')
```



# Создание и заполнение

Существует несколько способов создать и заполнить список

- 1) Создать пустой список

```
data = []
```

- 2) Перечисление его элементов

```
data1 = [3, 4, 5, 6, 88, 99]
```

- 3) С помощью функции list

```
a = list('список')
```

```
b = list(range(1, 11))
```

```
c = list()
```

- 4) С помощью операций умножения и сложения

```
d = [0] * 100
```

```
a = [1, 3] + [4, 23] + [5]
```



# Генераторы списков (списочные выражения)

5) Задав генератор списка – способ построить новый список, применяя выражение к каждому элементу последовательности

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

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

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

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



# Ввод значений и заполнение случайными числами

6) Ввод значений

```
a = []
```

```
for i in range(5):  
    a.append(int(input()))
```

или

```
a = [int(input()) for i in range(5)]
```

7) Случайные числа

```
from random import randint
```

```
a = []
```

```
for i in range(5):  
    a.append(randint(1,100))
```



# Создание списка из строки (метод split)

## 8) Создание списка из строки

```
s = "One two three"
```

```
d = s.split() # ["One", "two", "three"]
```

```
# 1, 2, 3, 4
```

```
v = [int(x) for i in input().split(", ")]
```

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



# Перебор элементов списка

0	1	2	3
-4	-3	-2	-1

Общая схема (можно изменять  $a[i]$ ):

```
for i in range(N):
    ... # сделать что-то с a[i]
```

```
for i in range(N):
    a[i] += 1
```

Если не нужно изменять  $a[i]$ :

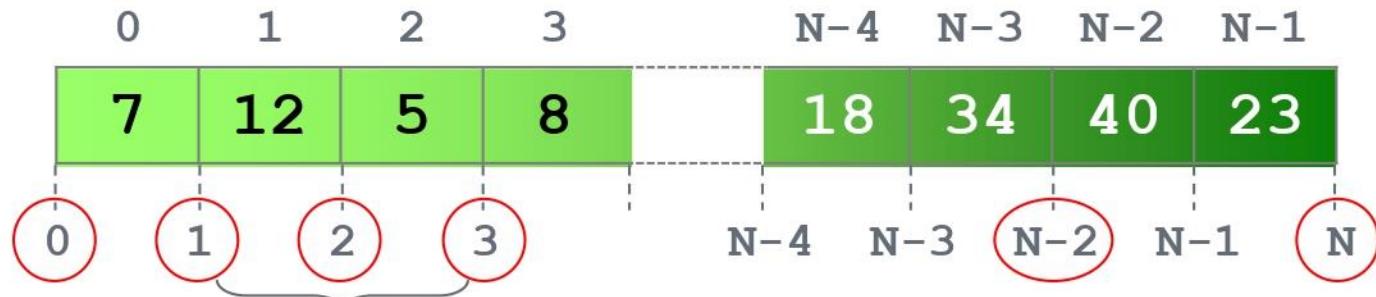
```
for x in a:
    ... # сделать что-то с x
```

$x = a[0], a[1], \dots, a[N-1]$

```
for x in a:
    print(x)
```



# Срезы списков



$A[1:3] \rightarrow [12, 5]$

$A[2:3] \rightarrow [5]$

$A[:3] \rightarrow A[0:3] \rightarrow [7, 12, 5]$

с начала

$\rightarrow [8, \dots, 18, 34]$

$A[3:] \rightarrow A[3:N] \rightarrow [8, \dots, 18, 34, 40, 23]$

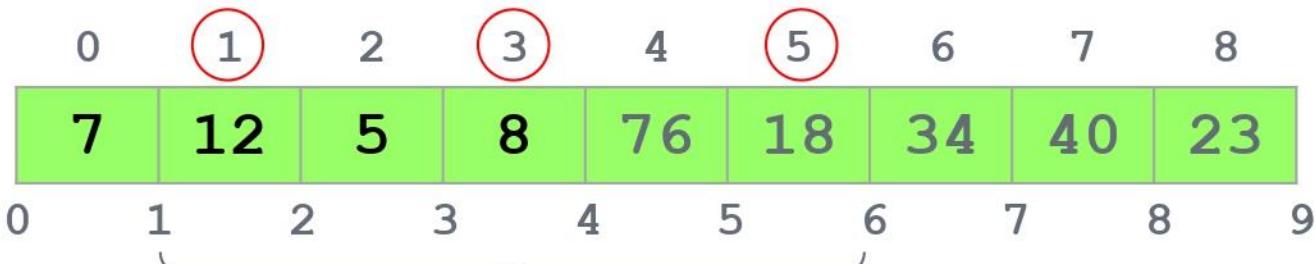
до конца

копия списка

$A[:] \rightarrow [7, 12, 5, 8, \dots, 18, 34, 40, 23]$



# Срезы списков



`A[1:6:2] → [12, 8, 18]`

`A[::-3] → [7, 8, 34]`

`A[8:2:-2] → [23, 34, 76]`

`A[::-1] → [23, 40, 34, 18, 76, 8, 5, 12, 7]`

реверс!

`A.reverse()`



# Максимальный элемент

```
M=A[0]
for i in range(1,N):
    if A[i]>M:
        M = A[i]
print ( M )
```

Варианты в стиле Python:

```
M=A[0]
for x in A:
    if x>M:
        M = x
```

```
M=max ( A )
```



Как найти его номер?



# Номер максимального элемента

```
nMax = 0
for i in range(1,N):
    if A[i] > A[nMax]
        nMax = i
print ("A[", nMax, "]=", A[nMax], sep= "")
```



# Разбор задач

Задача №63. A[0], A[2], A[4], ...

Дан массив, состоящий из целых чисел. Нумерация элементов начинается с 0. Напишите программу, которая выведет элементы массива, номера которых четны (0, 2, 4...).

## Входные данные

Сначала задано число  $N$  — количество элементов в массиве ( $1 \leq N \leq 100$ ). Далее через пробел записаны  $N$  чисел — элементы массива. Массив состоит из целых чисел.

## Выходные данные

Необходимо вывести все элементы массива с чётными номерами.

## Примеры

### входные данные

```
6
4 5 3 4 2 3
```

### выходные данные

```
4 3 2
```



# Разбор задач



## Задача №68. Количество элементов больших обоих соседей

Дан массив, состоящий из целых чисел. Напишите программу, которая в данном массиве определит количество элементов, у которых два соседних и, при этом, оба соседних элемента меньше данного.

### Входные данные

Сначала задано число  $N$  — количество элементов в массиве ( $1 \leq N \leq 100$ ).

Далее через пробел записаны  $N$  чисел — элементы массива. Массив состоит из целых чисел.

### Выходные данные

Необходимо вывести количество элементов массива, у которых два соседа и которые при этом строго больше обоих своих соседей.

### Примеры

#### входные данные

```
5
1 2 3 4 5
```

#### выходные данные

```
0
```



# Разбор задач



## Задача №70. Переставить соседние элементы

Напишите программу, которая переставляет соседние элементы массива (1-й элемент поменять с 2-м, 3-й с 4-м и т.д. Если элементов нечетное число, то последний элемент остается на своем месте).

### Входные данные

Сначала задано число  $N$  — количество элементов в массиве ( $1 \leq N \leq 35$ ).

Далее через пробел записаны  $N$  чисел — элементы массива. Массив состоит из целых чисел.

### Выходные данные

Необходимо вывести массив, полученный после перестановки элементов.

### Примеры

#### входные данные

```
6
4 5 3 4 2 3
```

#### выходные данные

```
5 4 4 3 3 2
```



## Обобщение свойств встроенных коллекций в сводной таблице:

Тип коллекции	Мутабельность	Индексированность	Уникальность	Как создаём
Список (list)	+	+	-	<code>[], list()</code>
Кортеж (tuple)	-	+	-	<code>(), tuple()</code>
Строка (string)	-	+	-	<code>''</code> <code>'''</code>
Множество (set)	+	-	+	<code>{}, set()</code>
Неизменное множество (frozenset)	-	-	+	<code>frozenset()</code>
Словарь (dict)	+ элементы - ключи + значения	-	+ элементы + ключи - значения	<code>{key: value,}, dict()</code>



# Словари



# Коллекции

## Словари:

Словари являются важнейшей структурой данных в Python-е. Они позволяют хранить данные в формате ключ-значение. Чтобы определить словарь, нужно использовать литерал фигурные скобки или просто вызвать dict. Если мы хотим, определяя словарь, сразу добавить в него данные, пишем ключ-значение через двоеточие.

```
empty_dict = {}  
empty_dict = dict()  
collections_map = {  
    'mutable': ['list', 'set', 'dict'],  
    'immutable': ['tuple', 'frozenset']}
```

Доступ к значению по ключу осуществляется за константное время, то есть не зависит от размера словаря. Это достигается с помощью алгоритма хеширования. Если пытаться получить доступ по ключу, которого не существует, Python выдаст ошибку KeyError. Однако, часто бывает полезно попытаться достать значение по ключу из словаря, а в случае отсутствия ключа вернуть какое-то стандартное значение. Для этого есть встроенный метод get.

```
print(collections_map['immutable'])  
['tuple', 'frozenset']
```



# Коллекции

## Словари:

```
print(collections_map['irresistible'])
```

KeyError

—> 1 print(collections\_map['irresistible'])

KeyError: 'irresistible'

Traceback (most recent call last) in ()

```
print(collections_map.get('irresistible', 'not found'))
```

not found

Проверка на вхождения ключа в словарь так же осуществляется за константное время и выполняется с помощью ключевого слова `in`:

```
'mutable' in collections_map
```

True



# Коллекции

## Словари:

Так как словарь является изменяемой структурой данных, мы можем добавлять и удалять элементы из него. Например, мы можем определить словарь `beatles_map`, который содержит знаменитых музыкантов и их инструменты, и добавить в него Ринго с 11 ударными, просто используя доступ по ключу. Чтобы удалить ключ и значение из словаря, можно использовать уже знакомый вам оператор `del`.

```
beatles_map = {  
    'Paul': 'Bass',  
    'John': 'Guitar',  
    'George': 'Guitar', }  
print(beatles_map)
```

```
{"Paul": "Bass", "John": "Guitar", "George": "Guitar"}
```

```
beatles_map['Ringo'] = 'Drums'  
print(beatles_map)
```

```
{"Paul": "Bass", "John": "Guitar", "George": "Guitar", "Ringo": "Drums"}
```

```
del beatles_map['John']  
print(beatles_map)  
  
{"Paul": "Bass", "George": "Guitar", "Ringo": "Drums"}
```



# Коллекции

## Словари:

Также, чтобы добавить какой-то ключ-значение в словарь, можно использовать встроенный метод `update`, который принимает словарь и дополняет им (а также обновляет в случае одинаковых ключей) исходный словарь.

```
beatles_map.update({ 'John': 'Guitar' })
print(beatles_map)
```

```
{'Paul': 'Bass', 'George': 'Guitar', 'Ringo': 'Drums', 'John': 'Guitar'}
```

Чтобы удалить ключ-значение из словаря и одновременно вернуть значение, используют метод `pop`:

```
# удаляем Ринго, нам возвращаются его ударные
print(beatles_map.pop('Ringo'))
print(beatles_map)
```

Drums

```
{'Paul': 'Bass', 'George': 'Guitar', 'John': 'Guitar'}
```



# Коллекции

## Словари:

Часто бывает необходимо не только попробовать проверить, существует ли ключ в словаре, но и в случае неудачи добавить эту новую пару ключ-значение. Для этого есть метод `setdefault`:

```
unknown_dict = {}  
print(unknown_dict.setdefault('key', 'default'))  
print(unknown_dict)
```

```
default  
{'key': 'default'}
```

Если вызвать `setdefault` и в качестве дефолтного значения передать `new_default`, вернётся значение, которое уже лежит в словаре — значение `default`:

```
print(unknown_dict.setdefault('key', 'new_default'))  
default
```



# Коллекции

## Словари:

Словари, как и все коллекции, поддерживают протокол итерации. С помощью цикла `for` можно итерироваться по ключам словаря:

```
print(collections_map)
for key in collections_map:
    print(key)

{'mutable': ['list', 'set', 'dict'], 'immutable': ['tuple', 'frozenset']}
mutable immutable
```

Если нам нужно итерироваться не по ключам, а по ключам и значениям сразу, можно использовать метод словаря `items`, который возвращает ключи и значения.

```
for key, value in collections_map.items():
    print('{} — {}'.format(key, value))
```

```
mutable — ['list', 'set', 'dict']
immutable — ['tuple', 'frozenset']
```



# Коллекции

## Словари:

Если нужно итерироваться по значениям, используйте логично метод `values`, который возвращает именно значения. Также существует симметричный метод `keys`, который возвращает итератор ключей.

```
for value in collections_map.values():
    print(value)
```

```
['list', 'set', 'dict']
['tuple', 'frozenset']
```

Важная особенность словарей в Python-е: они содержат ключи и значения в неупорядоченном виде. Однако, в Python-е существует тип `OrderedDict` (содержится в модуле `collections`), который гарантирует вам, что ключи хранятся именно в том порядке, в каком вы их добавили в словарь.

```
from collections import OrderedDict

ordered = OrderedDict()
for number in range(10):
    ordered[number] = str(number)
for key in ordered:
    print(key)
```



# Коллекции

Задача 1(сложно):

Осуществить редактор текста вводимого в консоль(нормализацию текста)(новые предложения с большой буквы, пробелы после знаков препинания)

Задача 2(средне):

Отсортировать по алфавиту любой текст содержащий знаки препинания и разный регистр  
(300 слов +)

Задача 3(просто):

Создать словарь характеристик любого объекта (описать 6-9 свойств объекта)



# Коллекции

## Задача 1:

Осуществить редактор текста вводимого в консоль(нормализацию текста)(новые предложения с большой буквы, пробелы после знаков препинания)

Для решения задач с абстрактным условием, лучше подходить поэтапно.

Первым делом стоит обратить внимание на то что дано в условии задачи, в данном случае у нас есть какой-то текст вводимый с клавиатуры.

```
text = input()
```

Для осуществления его нормализации нам понадобиться разбить его на составляющие. Стоит рассмотреть из чего он состоит - это слова и знаки препинания с пробелами. В том время как знаки препинания стоят из одного символа, а пробелы хранить и вовсе не нужно, слова состоят из нескольких символов. В связи с этим понадобиться отдельные переменные для хранения полной "карты" текста и для записи слов.

```
text_map = []
word = ''
```



# Коллекции

## Задача 1:

Далее нужно заполнить "карту", для этого циклом пройдёмся по каждому символу исходного текста, при обнаружении символа будем записывать его, исключая пробелы, преждевременно записав слово и обнулив его, в ином случае добавляя "букву" к переменной слова.

```
for symbol in text:  
    if symbol in ',.!?:'  
        if word != '': # проверка наличия слова  
            text_map.append(word) # сохранение слова  
            word = '' # обнуление слова для записи нового  
        if symbol in ',.!?' and symbol != ' ': # отсутствие пробела  
            text_map.append(symbol) # запись символа  
    else:  
        word+=symbol # добавление "буквы" к слову
```



# Коллекции

## Задача 1:

После цикла, при условии, что в конце текста не было знака препинания, но было "слово", переменная с ним не будет добавлена в "карту", проверку на это нужно сделать после цикла.

```
if word != '':
    text_map.append(word)
    word = ''
```

В случае если в конце теста был знак препинания или символ переменная word останется пустой. Далее необходимо "собрать" текст, нормализовав его. Для этого нам необходимо рассмотреть каждый элемент "карты" попутно запоминая предыдущий рассмотренный элемент для выполнения задачи по написанию каждого нового предложения с заглавной буквы. Начнём с создания переменной в которой сразу запишем первый элемент карты с заглавной буквы(если это будет знак препинания с ним ничего не произойдёт), также создадим переменную для предыдущего элемента.

```
normalize_text = text_map[0].capitalize()
prev = ''
```



# Коллекции

## Задача 1:

Создадим цикл, который будет проходить по срезу "карты", тк первый элемент уже был записан в нормализованный текст. Внутри цикла будем проверять , какой элемент встречаем, в случае если это знак препинания, просто добавляем его в нормализованный текст, в ином случае смотрим каким был предыдущий элемент , если это был знак препинания после которого нужно писать слово с заглавной буквы, добавляем пробел и слово с заглавной буквы, в ином аналогично, но слово нужно оставить в исходном виде.

```
for ele in text_map[1:]: #цикл по срезу от второго элемента
    if ele in ',.!?': # проверка на знак препинания
        normalize_text += ele
    else:
        if prev in '.!?:': # проверка на слово с заглавной
            normalize_text += ' '+ele.capitalize()
        else:
            normalize_text += ' '+ele
    prev = ele
print(normalize_text)
```

Срез необходим для того чтобы первый элемент игнорировался циклом, тк в случае слова в начале строки появиться пробел и оно будет записано с маленькой буквы.



# Коллекции

## Задача 2:

Отсортировать по алфавиту любой текст содержащий знаки препинания и разный регистр  
(300 слов +)

Составление "карты" для сортировки можно взять из предыдущего задания, убрав запись пунктуации и приводя каждую букву к нижнему регистру. При добавлении слова в "карту" проверяем, на наличие его в карте для исключения дублей

```
text = input()
text_map = []
word = ''
for symbol in text:
    if symbol in ',.!?':
        if word != '' and word not in text_map:
            text_map.append(word)
            word = ''
    else:
        word+=symbol.lower() # каждая буква к нижнему регистру
```



# Коллекции

## Задача 2:

Не забудем дописать сохранение последнего слова(помним о дублях) и отсортируем получившуюся "карту" в выводе.

```
if word != '' and word not in text_map:  
    text_map.append(word)  
    word = ''  
print(sorted(text_map))
```



# Коллекции

## Задача 3:

Создать словарь характеристик любого объекта (описать 6-9 свойств объекта)

Данное задание для демонстрации понимания конструкции словаря(ключ-значение).

Разберём усложнённый вариант его решения. Создадим список в котором будем хранить словари....  
допустим столов.

```
tables = []
```

Список будем задавать с помощью цикла `in range`, для удобства индексации и демонстрации понимания конструкции словаря зададим не просто `range`, а `range` с 1 для привидения индекса к человекочитабельному формату исчисления (с единицы) до любого числа, которое можем задавать в начале нашей программы, не зыбыв добавить к нему единицу, тк в `range` последний элемент на единицу меньше. Внутри цикла будем добавлять словари в общий список с помощью метода `append`. Самому же словарю зададим произвольные пары ключ-значения(опираясь на условие пар должно быть от 6 до 9), описывающие характеристики нашего объекта(стола), для части из значений зададим возможность ввода с клавиатуры. И предоставим возможность вывода конкретного элемента списка(стола)



# Коллекции

Задача 3:

```
amount = input('Введите количество столов')
tables = []
for id in range(1,amount+1):
    tables.append(
        {
            'id':id,
            'столешница': {
                'x': input('столешница(x): '),
                'y': input('столешница(y): '),
                'z': 3
            },
            'name': 'tabelshon',
            'material': 'red wood',
            'price': input('price: ')
        }
    )
```



# Коллекции

Задача 3:

```
output = tables[int(input(f'input table id(1-{amount}): '))-1]
print(output)
```



# Множества



# Коллекции

## Множества:

Множество в питоне — это неупорядоченный набор уникальных объектов. Множества изменяемы и чаще всего используются для удаления дубликатов и всевозможных проверок на вхождение.

Чтобы объявить пустое множество, можно воспользоваться литералом `set` или использовать фигурные скобки, чтобы объявить множество и одновременно добавить туда какие-то элементы.

```
empty_set = set()  
number_set = {1, 2, 3, 3, 4, 5}
```

```
print(number_set)  
{1, 2, 3, 4, 5}
```



# Коллекции

## Множества:

Чтобы проверить, содержится ли объект в множестве, используется уже знакомое нам ключевое слово `in`. Проверка выполняется за константное время, время выполнения операции не зависит от размера множества. Это достигается за счёт хэширования каждого элемента структуры по аналогии со словарями. По полученному от хэш-функции ключу и происходит поиск объекта. Таким образом, во множествах могут содержаться только хэшируемые объекты.

```
print(2 in number_set)
```

```
True
```



# Коллекции

## Множества:

Чтобы добавить элемент в множество, используется метод `add`. Также множества в Python поддерживают стандартные операции над множествами --- такие как объединение, разность, пересечение и симметрическая разность.

Создадим два множества с чётными и нечётными числами до десяти:

```
even_set = set()
odd_set = set()

for number in range(10):
    if number % 2:
        odd_set.add(number)
    else:
        even_set.add(number)

print(even_set)
print(odd_set)

{1, 3, 5, 7, 9}
{0, 2, 4, 6, 8}
```



# Коллекции

## Множества:

Теперь найдём объединение и пересечение этих множеств:

```
# объединение
union_set = odd_set | even_set
other_union_set= odd_set.union(even_set)
print(union_set)
print(other_union_set)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
# пересечение
intersection_set = odd_set & even_set
other_intersection_set= odd_set.intersection(even_set)
print(intersection_set)
print(other_intersection_set)
```

```
set()
```

```
set()
```



# Коллекции

## Множества:

Найдём разность и симметрическую разность двух множеств:

```
# разность
difference_set = even_set - odd_set
other_difference_set= odd_set.difference(even_set)
print(difference_set)
print(other_difference_set)
```

{0, 2, 4, 6, 8}

{1, 3, 5, 7, 9}

```
# симметрическая разность
symmetric_difference_set = even_set ^ odd_set
other_symm_difference_set= odd_set.symmetric_difference(even_set)
print(symmetric_difference_set)
print(other_symm_difference_set)
```

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}



# Коллекции

## Множества:

Множества --- изменяемая структура данных, поэтому можно как добавлять туда элементы, так и удалять. Для удаления конкретного элемента существует метод `remove`, для удаления любого элемента можно использовать `pop`. Остальные методы можно посмотреть в `help` или документации.

```
even_set.remove(2)  
print(even_set)
```

```
{0, 4, 6, 8}
```

```
print(even_set.pop())
```

```
0
```

Также в питоне существует неизменяемый аналог типа `set` --- тип `frozenset`.

```
frozen = frozenset(['Anna', 'Elsa', 'Kristoff'])  
frozen.add('Olaf')
```

```
AttributeError
```

```
Traceback (most recent call last)
```

```
<ipython-input-64-962f221e1321> in () <module>
```

```
1 frozen = frozenset(['Anna', 'Elsa', 'Kristoff'])
```

```
2
```

```
----> 3 frozen.add('Olaf')
```

```
AttributeError: 'frozenset' object has no attribute 'add'
```



# Коллекции

## Множества. Пример программы:

Задача на множества: через сколько итераций функция `random.randint(1, 10)` выдаст повтор? Будем добавлять неповторяющиеся случайные числа в множество `random_set`. Если очередное число уже есть в `random_set` --- выйдем из цикла. Затем посчитаем длину множества (и прибавим 1, тк не учли последнее число).

```
import random
random_set = set()

while True:
    new_number = random.randint(1,10)
    if new_number in random_set:
        break
    random_set.add(new_number)

print(len(random_set)+1)
```

4

Таким образом, мы получили повтор через 4 итераций.



# Проверка домашнего задания



# Домашняя работа:

Задача 1:

Испорченный гугл: Создать программу, которая будет обрабатывать русский текст, заменяя в нём символы русского языка на схожие по написанию символы английского языка(а,с,х,о,р,е), при этом сохраняя регистр.

Задача 2:

Буквоежка level-up: Отсортировать по алфавиту три(можно больше) отдельных строки убирая общие(между всеми тремя строками) дубли

(итог выводить для каждой отдельной строки)

Задача 3:

Склад: Создать список словарей для инвентаризации чего-либо

Задача 4:

Цензор: создать программу, которая будет цензуривать вводимый текст по заданному шаблону  
(не обязательно цензурировать мат! Шаблон может быть любым!(например слова-паразиты))



# Домашняя работа. Задача 1:

Для этого нам необходимо рассмотреть момент создания слова и дополнить программу словарями с заменой букв, где ключом будет символ для поиска, а значением - символ для замены. Создадим словари для всех возможных случаев.(совпадения во всех и конкретных регистрах)

```
text = input()
text_map = []
word = ''
repl_all = {'e':'e', 'o':'o', 'p':'p', 'a':'a', 'x':'x', 'c':'c'}
repl_up = {'T': 'T', 'K': 'K', 'B': 'B', 'M': 'M'}
repl_low = {'y': 'y'}
```



# Домашняя работа. Задача 1:

```
for symbol in text:  
    if symbol in ',.!?:'  
        if word != '':  
            text_map.append(word)  
            word = ''  
        if symbol in ',.!?' and symbol != ' ':  
            text_map.append(symbol)  
    else:
```



# Домашняя работа. Задача 1:

Сформируем "испорченное" слово с помощью условий и "поиска".

```
if symbol in repl_up:  
    word += repl_up[symbol]  
elif symbol in repl_low:  
    word += repl_low[symbol]  
elif symbol.lower() in repl_all:  
    if symbol in repl_all:  
        word += repl_all[symbol]  
    else:  
        word += repl_all[symbol.lower()].upper()  
else:  
    word += symbol
```



# Домашняя работа. Задача 1:

```
if word != '':
    text_map.append(word)
    word = ''
normalize_text = text_map[0].capitalize()
prev = ''
for ele in text_map[1:]:
    if ele in ',.!?':
        normalize_text += ele
    else:
        if prev in '.!?' and prev:
            normalize_text += ' ' + ele.capitalize()
            print(prev)
        else:
            normalize_text += ' ' + ele
    prev = ele
print(normalize_text)
```



# Домашняя работа. Задача 2:

Сформируем три списка и подготовим три множества.

```
text_1 = input().split()  
text_2 = input().split()  
text_3 = input().split()  
mn_1 = set()  
mn_2 = set()  
mn_3 = set()
```



# Домашняя работа. Задача 2:

Очистим элементы списка и заполним множества.

```
for ele in text_1:  
    mn_1.add(ele.strip(',.!?'))  
for ele in text_2:  
    mn_2.add(ele.strip(',.!?'))  
for ele in text_3:  
    mn_3.add(ele.strip(',.!?'))
```



# Домашняя работа. Задача 2:

Создадим множество дублей на основе повторений среди всех и каждой строки

```
mn = mn_1 & mn_2
for ele in mn_1 & mn_3:
    mn.add(ele)
for ele in mn_2 & mn_3:
    mn.add(ele)
```

Выведем результат(разность исходного множества каждой строки и множества пересечений), не забыв отсортировать его.

```
print(" ".join(sorted(mn_1-mn)))
print(" ".join(sorted(mn_2-mn)))
print(" ".join(sorted(mn_3-mn)))
```



# Домашняя работа. Задача 4:

Для этого нам необходимо рассмотреть момент записи слова и дополнить программу множеством, которое будет содержать цензурируемые слова.

```
text = input()
text_map = []
word = ''
ban_word = {'привет', 'пока'}
for symbol in text:
    if symbol in ',.!?:':
        if word != '':
            if word in ban_word:
                print('*' * len(word))
            else:
                text_map.append(word)
            word = ''
```



# Домашняя работа. Задача 4:

Проверив слово на наличие в множестве цензурируемых. И продолжим старым кодом до момента контрольной записи

```
if word in ban_word:  
    text_map.append('***')  
else:  
    text_map.append(word)  
    word = ''  
    if symbol in ',.!?' and symbol != ' ':  
        text_map.append(symbol)  
    else:  
        word += symbol  
  
if word != '':
```



# Домашняя работа. Задача 4:

Повторим проверку и закончим старым кодом.

```
if word in ban_word:  
    text_map.append('***')  
else:  
    text_map.append(word)  
word = ''  
normalize_text = text_map[0].capitalize()  
prev = ''
```



# Домашняя работа. Задача 4:

```
for ele in text_map[1:]:
    if ele in ',.!?':
        normalize_text += ele
    else:
        if prev in '.!?' and prev:
            normalize_text += ' ' + ele.capitalize()
            print(prev)
        else:
            normalize_text += ' ' + ele
    prev = ele
print(normalize_text)
```



Входит в ГК Аплана



АКАДЕМИЯ АЙТИ

Основана в 1995 г.

E-learning  
и очное  
обучение



Ежегодные награды  
Microsoft,  
Huawei, Cisco и  
другие

Направления обучения:

Информационные технологии

Информационная безопасность

ИТ-менеджмент и управление проектами

Разработка и тестирование ПО

Гос. и муниципальное управление

#### Филиалы:

Санкт-Петербург, Казань, Уфа, Челябинск,  
Хабаровск, Красноярск, Тюмень, Нижний  
Новгород, Краснодар, Волгоград, Ростов-на-Дону

Головной офис  
в Москве

Ресурсы более 400  
высококлассных  
экспертов и  
преподавателей

Разработка  
программного  
обеспечения и  
информационных  
систем

Программы по  
импортозамещению

Сеть региональных учебных центров  
по всей России

#### Крупные заказчики



100+  
сотрудников



АКАДЕМИЯ АЙТИ

# Спасибо за внимание!

Центральный офис:

Москва, Варшавское шоссе 47, корп. 4, 7 этаж

Тел: +7 (495) 150-96-00

[academy@it.ru](mailto:academy@it.ru)

[academyit.ru](http://academyit.ru)