

## Содержание

Глава 1. Язык Python и его особенности.....	2
1.1. Установка Python. Общие сведения о языке .....	2
1.2. Основы синтаксиса Python. Типы и структуры данных.....	3
Глава 2. Основные алгоритмы и их реализация на Python .....	8
2.1 Линейные алгоритмы.....	8
2.2 Условные конструкции в Python .....	10
2.3 Циклические алгоритмы.....	13
Глава 3. Функции и методы строк.....	17
Глава 4. Списки .....	20
Глава 6. Функции .....	23

## Глава 1. Язык Python и его особенности

### 1.1. Установка Python. Общие сведения о языке

Язык программирования Python 3 - это мощный инструмент для создания программ самого разнообразного назначения. Многофункциональный язык программирования Python является высокоуровневым языком программирования общего назначения с акцентом на производительность разработчика и читаемость кода.

Версии Python доступны для бесплатной загрузки по адресу <https://www.python.org/>

Для загрузки для Windows следует перейти по ссылке <https://www.python.org/downloads/windows/>

После установки в меню "Пуск" появляется раздел Python.

В стандартный комплект поставки Python входит интегрированная среда разработки IDLE. IDLE так же имеет встроенную систему отладки, позволяющую запускать программу построчно, что облегчает процесс поиска ошибок.

Независимо от того, в какой ОС Вы работаете, окно IDLE при его первом открытии будет в основном пустым, не считая текста, показанного на рисунке. Это окно называется интерактивной оболочкой.

Команды, вводимые в интерактивной оболочке, выполняются интерпретатором Python.

Например, вывести на экран текст Hello, world.

Для этого введем строку

```
print("Hello, world")
```

После нажатия клавиши Enter на экране появится следующее (рис.1):

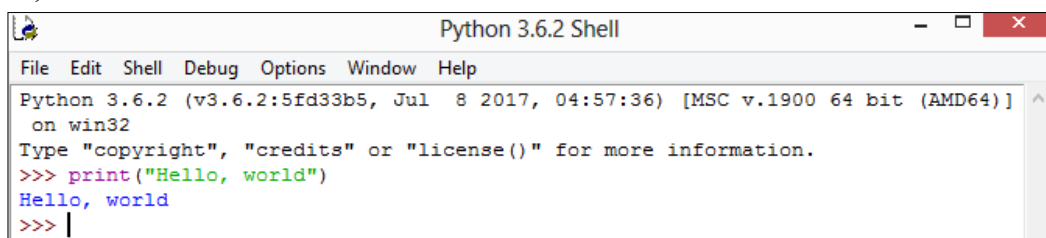


Рис.1.

Для создания нового окна в интерактивном режиме IDLE нажмите File → New file (или нажмите Ctrl+N).

В появившемся окне введите код:

```
name=input("Как Вас зовут?")  
print("Привет, ", name)
```

Первая строка напечатает вопрос "Как Вас зовут?", ожидая, пока Вы не напечатаете что-нибудь и не нажмете клавишу Enter. Введенный текст сохранится в переменную name.

Во второй строке функция print выведет "Привет," и значение переменной name.

Для запуска программы используется клавиша F5 или Run→Run Module. Результат запуска показан на рис.2.

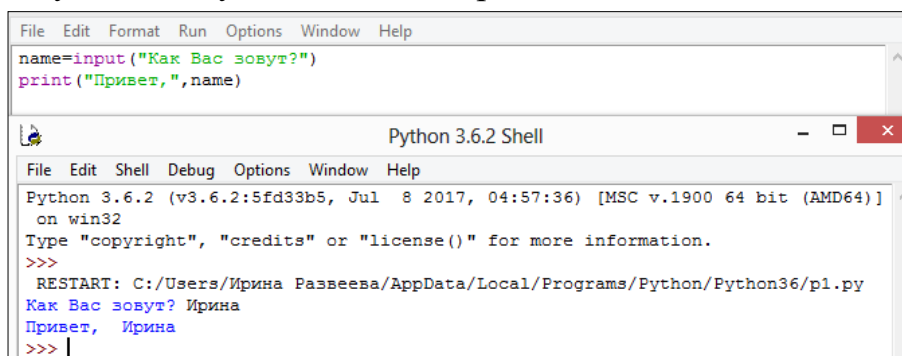


Рис.2.

## 1.2. Основы синтаксиса Python. Типы и структуры данных

### *Определение идентификатора.*

Идентификатор - имя существующего объекта в программе, которое является его уникальным признаком, позволяющим отличать его от прочих объектов. Идентификатор должен начинаться с буквы (от а до Z) или со знака подчеркивания "\_", после которых может идти произвольное количество букв, знаков подчеркивания и чисел (от 0 до 9).

В Python недопустимо использование знаков препинания или специальных символов, таких как @, \$ или % в качестве идентификаторов. Кроме того, Python чувствителен к регистру, то есть a1 и A1 это два разных имени.

Зарезервированные слова нельзя использовать в качестве имени переменной или любого другого идентификатора. Все ключевые слова Python состоят только из букв в нижнем регистре. Получить список ключевых слов возможно в интерпретаторе командой

```
help("keywords")
```

### *Обозначения отдельных блоков кода*

Одна из первых особенностей Python, которая бросается в глаза программистам, начинающим изучать этот язык программирования, это

то, что в нем не используются скобки для обозначения отдельных блоков кода. Вместо них в Python используются двоеточия и отступы.

### *Комментирование в Python*

Символ решетки (#) в Python обозначает начало комментария. Любые символы после решетки и до конца строки считаются комментариями и игнорируются интерпретатором.

Например следующий код:

```
# First comment
print "Hello, world!" # second comment
```

Выведет только Hello, world! в консоль.

### *Переменные в Python*

Переменная в языке программирования это название для зарезервированного места в памяти компьютера, предназначенное для хранения значений. В Python объявление происходит автоматически (это называется динамическая типизация), когда вы присваиваете значение переменной.

### *Операторы присваивания в Python*

Оператор	Описание
=	Присваивает значение правого операнда левому. $x = 23$
+=	Прибавит значение правого операнда к левому и присвоит эту сумму левому операнду, равносильно: $x = x + y$ .
-=	Отнимает значение правого операнда от левого и присваивает результат левому операнду, равносильно: $x = x - y$ .
*=	Умножает правый операнд с левым и присваивает результат левому операнду, равносильно: $x = x * y$ .
/=	Делит левый операнд на правый и присваивает результат левому операнду, равносильно: $x = x / y$ .
%=	Делит по модулю операнды и присваивает результат левому, равносильно: $x = x \% y$
**=	Возводит в левый операнд в степень правого и присваивает результат левому операнду, равносильно $x = x ** y$
//=	Производит целочисленное деление левого операнда на правый и присваивает результат левому операнду, равносильно $x = x // y$
x=y=z	Множественное присваивание

Например:

```
x = "Hello" # Присвоить значение Hello переменной
под названием x
y = 10 # Присвоение значения 10 переменной y
z = 5
print (x)
y+=z
print (y)
```

При выполнении данного кода выводится следующее:

```
Hello
15
```

### *Числа в Python*

Числа в Python могут быть целыми (тип int), длинными целыми (тип long), вещественными (тип float) и комплексными (тип complex).

Для преобразования чисел из вещественных в целые и наоборот в Python определены функции int () и float (). Основные операции с числами приведены в таблице 1.

Таблица 1

Операция	Описание
$x + y$	Сложение
$x - y$	Вычитание
$x * y$	Умножение
$x / y$	Деление $x$ на $y$
$x // y$	Получение целой части от деления
$x \% y$	Остаток от деления
$-x$	Смена знака числа
$\text{abs}(x)$	Модуль числа
$x ** y$	Возведение в степень

Над целыми числами также можно производить битовые операции (таблица 2).

Таблица 2

Операция	Описание
$x   y$	Побитовое или
$x \wedge y$	Побитовое исключающее или
$x \& y$	Побитовое и
$x \ll n$	Битовый сдвиг влево
$x \gg y$	Битовый сдвиг вправо
$\sim x$	Инверсия битов

Выполнить код:

```
x=int(input('введите целое число x='))
y=float(input('введите вещественное число y='))
z=x+y
print(z)
n=round(z)
print((n)) # round()- округление
print(x/y)
print(x|n)
print(~x)
print(x % y)
print(bin(x)) #преобразование целого числа в
двоичную строку
print(oct(x),hex(x)) #преобразование целого числа
в восьмеричную, шестнадцатеричную строку
x = complex(1, 2) # x-комплексное число
print(x)
```

### *Строковый тип*

Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации.

В Python можно использовать одинарные ( ' ), двойные ( " ) и тройные ( " " " или " " " " ) кавычки чтобы обозначить строчный тип данных, при этом начинаться и заканчиваться строка должна одинаковыми кавычками. Строка, занимающая несколько строк кода, должна быть обозначена тройными кавычками.

Например:

```
str1 = 'строка 1'
str2 = "строка 2"
str3 = """ строка 3
строка3 """
```

Строки в языке Python невозможно изменить – в этом случае говорят, что это `immutable` (неизменяемый) тип. Попытка изменить символ в определенной позиции или подстроку вызовет ошибку:

```
str = 'строка'
str[2] = 'a'
```

`TypeError: 'str' object does not support item assignment`

### *Логические значения в Python.*

Логические значения в Python представлены двумя величинами логическими константами True (Истина) и False (Ложь). Основные логические операции представлены в таблице 3.

Таблица 3

Операция	Описание
and	Логический оператор "И". Условие будет истинным если оба операнда истина.
or	Логический оператор "ИЛИ". Если хотя бы один из операндов истинный, то и все выражение будет истинным.
not	Логический оператор "НЕ". Изменяет логическое значение операнда на противоположное.

Выполнить код:

```
x=True  
y=False  
print(x*y)  
print(not(x))
```

### *Операторы сравнения в Python*

В следующей таблице приведены операторы сравнения в Python.

Таблица 4

Оператор	Описание
==	Проверяет равны ли оба операнда.
!=	Проверяет равны ли оба операнда.
<>	Проверяет равны ли оба операнда.
>	Проверяет больше ли значение левого операнда, чем значение правого.
<	Проверяет меньше ли значение левого операнда, чем значение правого.
>=	Проверяет больше или равно значение левого операнда, чем значение правого.
<=	Проверяет меньше или равно значение левого операнда, чем значение правого.

### *Приоритет операторов в Python*

В следующей таблице описан приоритет выполнения операторов в Python (операторы отсортированы по их приоритету от наибольшего – к меньшему:).

Таблица 5

Оператор	Описание
**	Возведение в степень
~ + -	Комплиментарный оператор
* / % //	Умножение, деление, деление по модулю, целочисленное деление.
+ -	Сложение и вычитание.
>> <<	Побитовый сдвиг вправо и побитовый сдвиг влево.
&	Бинарный "И".
^	Бинарный "Исключительное ИЛИ" и бинарный "ИЛИ"
<= < > >=	Операторы сравнения
<> == !=	Операторы равенства
= %= /= //= -= += *= **=	Операторы присваивания
is is not	Тождественные операторы
in not in	Операторы членства
not or and	Логические операторы

## Глава 2. Основные алгоритмы и их реализация на Python

### 2.1 Линейные алгоритмы

Линейный алгоритм- алгоритм, в котором вычисления выполняются строго последовательно. Далее рассмотрим типичные задачи с линейной структурой алгоритма.

**Пример 1.** Даны два целых числа. Поменять местами их значения.

*Постановка и решение задачи.* Исходными данными являются две переменные, например x и y, значения которых- целые числа. Требуется, чтобы значение x стало равно y, а значение y стало равно x. Для обмена значениями целесообразно использовать дополнительную переменную, например z.

*Текст программы на Python:*

```
x = 5 # Присвоить переменной x значение 5
y = 7 # Присвоить переменной y значение 7
z=x   # Присвоить переменной z значение x
x=y   # Присвоить переменной x значение y
y=z   # Присвоить переменной y значение z
print ('x=', x, 'y=', y) # Вывод переменных x и y на экран
```



**Пример 2.** Найти значение переменной  $y = \frac{2x}{A+B} + \sqrt{C + 5x^2 - B}$  при различных значениях A, B, C, x, вводимых с клавиатуры.

*Постановка и решение задачи.* Исходными данными являются переменные A, B, C, x, вводимые с клавиатуры. Отметим, что результатом функции input(), используемой для ввода пользователем значения с клавиатуры, является строковое значение. Преобразования введенного значения и дальнейшее использование его в качестве числа возможно при использовании функции int(), которая вернет целочисленную форму передаваемого значения.

Для удобства разобьем исходную формулу y на  $y1 = \frac{2x}{A+B}$  и  $y2 = \sqrt{C + 5x^2 - B}$ . Результирующая формула примет следующий вид:  $y = y1 + y2$ .

*Текст программы на Python:*

```
A = int(input('A=')) # Ввод переменной A с клавиатуры
B = int(input('B=')) # Ввод переменной B с клавиатуры
C = int(input('C=')) # Ввод переменной C с клавиатуры
x = int(input('x=')) # Ввод переменной x с клавиатуры
y1=2*x/(A+B)         # Вычисление y1
print ('y1=', y1)    #Вывод y1 на экран
y2=(C+5*x**2-B)**(1/2) # Вычисление y2
print ('y2=', y2)    #Вывод y2 на экран
y=y1+y2              # Вычисление результирующей переменной y
print ('y=', y)      # Вывод y на экран
```

### **Задания для самостоятельного решения.**

1. Даны три числа- a, b, c. Найти среднее арифметическое, среднее геометрическое чисел.
2. Найти значения переменной y, заданные формулой  $y = x^3 + x^2 - 3/4$  для трех различных значений x, введенных с клавиатуры.
3. Даны действительные числа A, B, C. Найти периметр треугольника.
4. Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.
5. Дан радиус шара. Найти его объем.
6. Дана сторона равностороннего треугольника. Найти площадь этого треугольника.
7. Известен объем информации в байтах. Перевести в килобайты, мегабайты, гигабайты.

## 2.2 Условные конструкции в Python

Условные конструкции в Python существуют для того, чтобы разработчик мог задавать определенное поведение программы в зависимости от заданных условий.

Оператор ветвления `if` позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования:

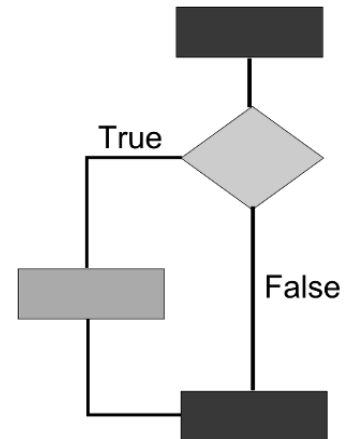
*Конструкция `if`*

Синтаксис оператора `if`:

```
if выражение:  
    инструкция_1  
    инструкция_2  
    ...  
    инструкция_n
```

Например:

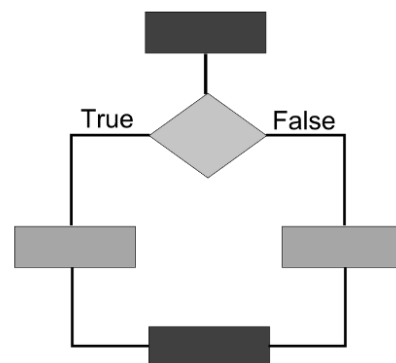
```
a = 3  
if a == 3:  
    print("Переменная a=3")
```



*Конструкция `if – else`*

Если необходимо предусмотреть альтернативный вариант выполнения программы, т.е. при истинном условии нужно выполнить один набор инструкций, при ложном – другой, то для этого используется конструкция `if – else`.

```
if выражение:  
    инструкция_1  
    инструкция_2  
    ...  
    инструкция_n  
else:  
    инструкция_a  
    инструкция_b  
    ...  
    инструкция_x
```



Например:

```
a = 3  
if a > 2:
```

```
    print("a>2")
else:
    print("a<=2")
```

### *Конструкция if – elif – else*

Для реализации выбора из нескольких альтернатив можно использовать конструкцию if – elif – else.

```
if выражение_1:
    инструкции_(блок_1)
elif выражение_2:
    инструкции_(блок_2)
elif выражение_3:
    инструкции_(блок_3)
else:
    инструкции_(блок_4)
```

Например:

```
a = int(input("введите число:"))
if a < 0:
    print("введено отрицательное число")
elif a == 0:
    print("введен 0")
else:
    print("введено положительное число ")
```

Оператор ветвления можно записать как выражение, а не как инструкцию:

```
x=40
y = x * 10 if x < 10 else x / 10
print(y)
```

**Пример 3.** Задать две целочисленные переменные a и b. Определить максимальную из них.

*Постановка и решение задачи.* Исходными данными являются две переменные, хранящие целые числа, требуется найти максимальное из этих чисел.

*Текст программы на Python:*

В первом случае оператор ветвления записан как выражение:

```
a = int(input("введите число a"))
b = int(input("введите число b"))
print(a) if a > b else print (b)
```

Во втором случае оператор ветвления записан с помощью конструкции if:

```
a = int(input("введите число a"))
b = int(input("введите число b"))
if a > b:
    print(a)
else:
    print(b)
```

**Пример 4.** Имеются 2 товара. Проверить, хватит ли 100р. на их покупку.

*Постановка и решение задачи.* Исходными данными являются две переменные tovar1 и tovar2, хранящие стоимость товаров. Необходимо определить, хватит ли суммы в 100р. на покупку данных товаров т.е. условие принимает вид  $(\text{tovar1} + \text{tovar2}) \leq 100$ .

*Текст программы на Python:*

```
tovar1 = 25
tovar2 = 50
if (tovar1 + tovar2) <= 100:
    print("Чек оплачен")
else:
    print("100 рублей недостаточно")
```

#### **Задания для самостоятельного решения.**

1. По координатам определить, в какой четверти находится точка.
2. Ввести два целых однозначных числа. Программа задаёт вопрос: "Результат умножения первого числа на второе?" Пользователь должен ввести ответ и увидеть на экране правильно он ответил или нет. Если нет – показать еще и правильный результат.
3. Даны три целых числа. Найти количество одновременно положительных и четных чисел в исходном наборе.
4. На числовой оси расположены три точки: А, В, С . Определить, какая из двух последних точек (В или С ) расположена ближе к А , и вывести эту точку и ее расстояние от точки .
5. Даны целочисленные координаты трех вершин прямоугольника, стороны которого параллельны координатным осям. Найти координаты его четвертой вершины.

## 2.3 Циклические алгоритмы

Циклы позволяют повторять некоторое действие в зависимости от соблюдения некоторого условия.

### *Оператор цикла while*

Оператор цикла `while` выполняет указанный набор инструкций до тех пор, пока условие цикла истинно. Истинность условия определяется также как и в операторе `if`. Синтаксис оператора `while` выглядит так.

```
while выражение:
    инструкция_1
    инструкция_2
    ...
    инструкция_n
```

Выполняемый набор инструкций называется телом цикла.

Например:

```
a = 0
while a < 7:
    print("a=", a)
    a += 1
```

*Пример бесконечного цикла.*

```
a = 5
while a > 0:
    print("бесконечный цикл")
```

### *Операторы break и continue*

При работе с циклами используются операторы *break* и *continue*.

Оператор *break* предназначен для досрочного прерывания работы цикла.

Например:

```
a = 1
while a >= 1:
```

```
if a == 7:
    break
print("шаг цикла= ", a)
a += 1
```

Результат выполнения данного кода:

```
шаг цикла= 1
шаг цикла= 2
шаг цикла= 3
шаг цикла= 4
шаг цикла= 5
шаг цикла= 6
```

В приведенном выше коде, выход из цикла произойдет при достижении переменной `a` значения 7. Если бы не было этого условия, то цикл выполнялся бы бесконечно.

Оператор *continue* запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

Например:

```
i = 0
while i < 10:
    i += 1
    if i == 3:
        print("Пропускаем 3")
        continue
    else:
        print("Текущее значение: ", i)
```

*Оператор цикла for*

Оператор `for` выполняет указанный набор инструкций заданное количество раз, которое определяется количеством элементов в наборе.

Например:

```
for i in range(5):
```

```
print("Hello")
```

В результате “Hello” будет выведено пять раз.

Функция `range()` является универсальной функцией для создания арифметической прогрессии. Функция `range()` может принимать от одного до трех аргументов, при этом аргументами должны быть целые числа (`int`). `range(старт, стоп, шаг)` - так выглядит стандартный вызов функции `range()` в Python. По умолчанию старт равняется нулю, шаг единице.

```
a=5
for i in range(a):
    print("шаг цикла", i)
```

Результат выполнения данного кода:

```
шаг цикла  0
шаг цикла  1
шаг цикла  2
шаг цикла  3
шаг цикла  4
```

Внутри тела цикла можно использовать операторы `break` и `continue`, принцип работы их точно такой же как и в операторе `while`.

**Пример 5.** Вывести нечетные числа в интервале от 1 до 10.

*Постановка и решение задачи.* Для решения задачи удобно воспользоваться циклом `for` с функцией `range(0,10,2)`.

*Текст программы на Python:*

```
for i in range(1,10,2):
    print(i)
```

Следующий способ решения задачи предполагает проверку числа на четность:

```
for i in range(10):
    if i % 2 != 0:
        print(i)
```

Также можно воспользоваться циклом `while`.

```

a = 1
while (a < 10):
    if (a%2!=0):
        print(a)
    a=a+1

```

### **Пример 6.** Найти факториал числа.

*Постановка и решение задачи.* Факториалом числа называют произведение всех натуральных чисел до него включительно. Например, факториал числа 5 равен произведению  $1*2*3*4*5 = 120$ . Формулу нахождения факториала можно записать следующим образом:  $n! = 1*2 * \dots * n$ , где  $n$  – это число, а  $n!$  – факториал этого числа.

*Текст программы на Python:*

Вычисление факториала с помощью цикла while.

```

n = int(input("Введите число n="))
fact = 1
i = 0
while i < n:
    i += 1
    fact = fact * i
print ("факториал равен", fact)

```

После первого прохода по телу цикла while переменная  $fact = 1 * 1$ . На втором шаге  $fact = 1 * 2$ , затем  $fact = 2 * 3$ ,  $fact = 6 * 4$ ,  $fact = 24 * 5$ . Шестой раз цикл while выполняться не будет, т.к. значение переменной  $i$  будет равно 5 и выражение  $(i < n)$  окажется False.

Вычисление факториала с помощью цикла for.

```

n = int(input("Введите число n="))
fact = 1
for i in range(1,n+1): # n+1 т.к. левая граница
    функции range() не включается
    fact *= i

```



```
print ("факториал равен", fact)
```

### **Задания для самостоятельного решения.**

1. Среди 10 чисел, вводимых с клавиатуры, найти количество положительных и отрицательных значений.
2. Найти среднее арифметическое положительных чисел, введенных с клавиатуры. Всего ввести N различных чисел.
3. Ввести с клавиатуры 10 чисел. Найти сумму тех из них, которые принадлежат интервалу (-5; 5).
4. Найти максимальный из отрицательных элементов среди произвольных 10 чисел, вводимых с клавиатуры.
5. Найти количество чисел, кратных пяти, из последовательности, вводимой с клавиатуры до тех пор, пока не встретится ноль.

### **Глава 3. Функции и методы строк**

Базовые операции со строками:

*Конкатенация (сложение):*

```
S1 = 'стро'  
S2 = 'ка'  
print(S1 + S2) #строка
```

*Дублирование строки*

```
print('hi' * 3) #hihihi
```

*Длина строки*

```
len('hello') #5
```

*Доступ по индексу.* Индексация строки начинается с 0. Индексы могут иметь отрицательные значения для отсчета с конца – отсчет начинается с -1.

```
S = 'hello'
```

```
print(S[0])    #h
print(S[2])    #l
print(S[-4])   #e
```

*Срез.* Срез – это механизм гибкого управления строкой на основе индексации. Оператор извлечения среза: имя\_строки[X:Y], где X – начало среза, а Y – окончание; символ с номером Y в срез не входит. По умолчанию первый индекс равен 0, а второй - длине строки

```
s= 'friends'
print(s[3:5])    #en
print(s[4:-2])   #n
print(s[3:])     #ends
print(s[:])      #friends
```

*Можно задать шаг, с которым нужно извлекать срез.*

```
s= 'friends'
print(s[-2:1:-2])    #de
print(s[::-1])       #sdneirf
```

В таблице 6 представлены функции и методы работы со строками в Python.

Таблица 6

S.find(str, start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.rfind(str,[start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
S.index(str,[start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str,[start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError

S.replace(шаблон,замена)	Замена шаблона
S.split(символ)	Разбиение строки по разделителю
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.startswith(str)	Начинается ли строка S с шаблона str
S.endswith(str)	Заканчивается ли строка S шаблоном str
ord(символ)	Код символа в ASCII
chr(число)	Перевод кода ASCII в символ

**Пример 7.** Проверить, есть ли в строке буква о, если есть, то определить ее номер.

*Постановка и решение задачи.* Для решения задачи удобно воспользоваться циклом for.

*Текст программы на Python:*

```
k=0
s='строка'
for i in s:
    if i != 'о':
        k=k+1
        continue
    else:
        print("буква о найдена, ее номер= ", k)
```

### **Задания для самостоятельного решения.**

Дана строка "Том появился на тротуаре с ведром извести и длинной кистью в руках. "

1. Определить длину строки.
2. Организовать поиск подстроки в строке.
3. Перевести символы нижнего регистра в верхний, а верхнего – в нижний.
4. Разбить строку по разделителю (в качестве разделителя использовать символ пробел).
5. Проверить, состоит ли строка из цифр.

6. Преобразовать строку к верхнему регистру.
7. Первые буквы в словах представить в коде ASCII.
8. Используя методы `find` и `rfind`, определить вхождение подстроки "ведром".
9. Используя метод `replace`, заменить "Том" на "Тим".
10. Используя метод `replace` с параметром `count`, заменить символ "о" на "а", но не все вхождения, а только первые 3 из них.

#### Глава 4. Списки

В отличие от обычных переменных, которые содержат единственный элемент данных, в Python существует так называемый список, в котором может храниться несколько элементов данных. Данные хранятся последовательно в «элементах» списка, которые индексируются числовыми значениями, начиная с нуля.

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

**Пример 7.** Дан список сотрудников, распределенных по 5 рабочим дням недели. Вывести фамилию сотрудника, работающего в конкретный день.

*Постановка и решение задачи.* Для решения задачи объявим список `Week`, содержащий фамилии сотрудников. В качестве обозначения конкретного дня используем переменную `n`, вводимую пользователем с клавиатуры.

*Текст программы на Python:*

```
Week= ['Иванов', 'Петров', 'Сидоров', 'Ян', 'Гусев']  
n=int(input('n='))  
print( Week[n-1])
```

Python позволяет обрабатывать двумерные списки, называемые матрицами или двумерными массивами.

```
list=[[11,12,13,14,15],[16,17,18,19,20],[21,22,23,24,25]]  
print(list[-2][-1]) #20  
print(list[-1][-5]) #21
```

```
print(list[-2][-3]) #18
```

индексы	0	1	2	3	4	
0	11	12	13	14	15	-3
1	16	17	18	19	20	-2
2	21	22	23	24	25	-1
	-5	-4	-3	-2	-1	

Для обработки и вывода списка, как правило, используют два вложенных цикла. Первый цикл перебирает номер строки, второй цикл бежит по элементам внутри строки. Например, вывести двумерный числовой список на экран построчно, разделяя числа пробелами внутри одной строки, можно так:

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=' ')
    print()
```

Списки, содержащие множественные элементы данных, широко используются в программировании на Python. Для работы с ними существует множество так называемых методов, к которым можно обращаться через точечную запись.

Метод	Описание
<b>list.append(x)</b>	Добавление элемента в конец списка
<b>list.extend(L)</b>	Расширение списка list путем добавления в конец элементов списка L
<b>list.insert(i, x)</b>	Вставка элемента по указанному индексу i
<b>list.remove(x)</b>	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
<b>list.pop([i])</b>	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
<b>list.index(x, [start [, end]])</b>	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
<b>list.count(x)</b>	Возвращает количество элементов со значением x
<b>list.sort()</b>	Сортировка списка
<b>list.reverse()</b>	Разворачивает список
<b>list.copy()</b>	Поверхностная копия списка
<b>list.clear()</b>	Очищает список

*Кортеж*

Но список может быть создан также с фиксированными (неизменяемыми) значениями, которые остаются постоянными на протяжении всей программы. Неизменяемый список в языке Python называется кортежем и создается присваиванием значений, разделенных запятой и стоящих в скобках. Такой процесс присваивания называется упаковкой кортежа:

```
colors = ( 'Red' , 'Green' , 'Red' , 'Blue' , 'Red' )
```

К отдельному элементу кортежа можно обращаться, используя запись имени кортежа и индекса в квадратных скобках. Все значения, хранящиеся внутри кортежа, могут быть присвоены отдельным переменным. Такой процесс называется распаковкой последовательности:

```
a , b , c , d , e = colors
```

### *Множество*

В обычном списке элементы могут повторяться, как и в кортеже, но существует возможность создавать такой список, где нет повторяющихся элементов. Неизменяемый список уникальных значений называется множеством и создается он присвоением значений через запятую и заключением их в фигурные скобки:

```
phonetic-set = { 'Alpha' , 'Bravo' , 'Charlie' }
```

Нельзя обратиться к отдельному элементу множества, используя имя множества и квадратные скобки, содержащие индекс. Вместо этого у множеств существуют методы для работы с ними.

Для того чтобы определить принадлежность той или иной структуры

данных к классу списков, применяется функция **type()** , а чтобы найти значение в этом списке, используется встроенный оператор вхождения **in**.

```
bag = { 'Red' , 'Green' , 'Blue' }  
print( type( bag ) )  
print( 'Orange?:' , 'Orange' in bag )  
print( 'Red ?:' , 'Orange' in bag )
```

### **Задания для самостоятельного решения.**

1. Определить индексы элементов массива (списка), значения которых принадлежат заданному диапазону

2. Найти произведение первого, пятого и восьмого положительных элементов массива.

3. Выведите все элементы списка с четными индексами.
4. В списке все элементы различны. Поменяйте местами минимальный и максимальный элемент этого списка.
5. Найти среднее арифметическое отрицательных элементов. Заменить на него минимальный элемент

## Глава 6. Функции

Функция это блок организованного, многократно используемого кода, который используется для выполнения конкретного задания. Функции обеспечивают лучшую модульность приложения и значительно повышают уровень повторного использования кода.

Правила для создания функций в Python.

1. Блок функции начинается с ключевого слова `def`, после которого следуют название функции и круглые скобки `()`.
2. Любые аргументы, которые принимает функция должны находиться внутри этих скобок.
3. После скобок идет двоеточие `:` и с новой строки с отступом начинается тело функции.

Пример функции:

```
def sum(x, y):  
    return x + y
```

Инструкция `return` говорит, что нужно вернуть значение. В данном случае функция возвращает сумму `x` и `y`. Вызов функции выглядит следующим образом:

```
sum (5, 10) #аргументы функции- числа  
sum ('abc', 'def')# аргументы функции- строки
```

Количество передаваемых аргументов функции и их порядок должен учитываться при вызове функции, например:

```
def maximum(a,b):  
    if a > b:  
        print (a)  
    else:  
        print (b)
```

```
# В описании функции указано, что функция имеет
2 аргумента
# Корректный вызов функции
maximum(5, 6)
# Некорректный вызов функции
maximum()
maximum(4)
maximum(11, 7, 3)
```

Аргументы - ключевые слова используются при вызове функции. Благодаря ключевым аргументам, вы можете задавать произвольный порядок аргументов.

```
def man(name, age):
    print (name, 'is', age, 'years old')
man(age=25, name='Mike')
```

В Python две базовых области видимости переменных:

- глобальные переменные;
- локальные переменные.

Переменные объявленные внутри тела функции имеют локальную область видимости, те что объявлены вне какой-либо функции имеют глобальную область видимости. Это означает, что доступ к локальным переменным имеют только те функции, в которых они были объявлены, в то время как доступ к глобальным переменным можно получить по всей программе в любой функции.

Например:

```
# глобальная переменная age
age = 40
def man():
    print (age) # Вывод глобальной переменной
age=40
def local_man():
    age = 25 # Создаем локальную переменную age
    print (age)
man() # напечатает 40
local_man() # напечатает 25
```

Для того чтобы изменить глобальную переменную внутри функции необходимо использовать ключевое слово `global`.

Например:

```
# глобальная переменная age
```



```
age = 25
# функция изменяющая глобальную переменную
def age_man():
    global age
    age += 5
print (age) # напечатает 25
age_man() # увеличиваем age на 5
print (age) # напечатает 30
```

Рекурсивная функция вычисления факториала на языке Python будет выглядеть так:

```
def fact(num):
    if num == 0:
        return 1 # факториал нуля равен единице
    else:
        return num * fact(num - 1)
print(fact(5))
```

### **Задания для самостоятельного решения.**

1. Определить наименьшее общее кратное для чисел а и в.
2. Посчитать количества одинаковых элементов в списке, используя функцию.
3. Разделить элементы списка на минимальный элемент.
4. Среди 10 чисел найти максимум и заменить его на среднее арифметическое положительных элементов.
5. В списке определить количество четных и нечетных элементов.