Python

ОБЩАЯ ИНФОРМАЦИЯ.

Общая информация

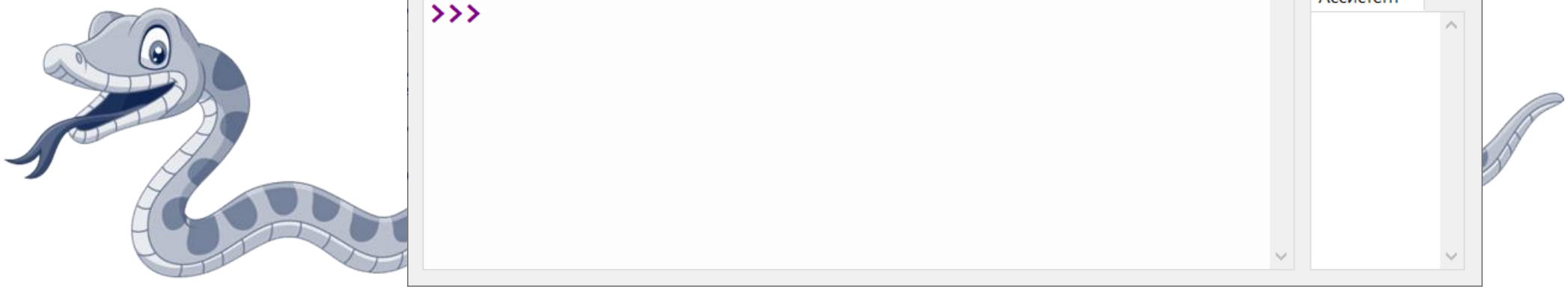
- Python – один из самых популярных языков программирования.
- Первое появление – 1991 год.
- Создатель – нидерландский программист Гвидо ван Россум.
- На языке Python можно программировать игры, web-сайты, обрабатывать большие объёмы данных, решать задачи искусственного интеллекта.
- Применение ему находится и в YouTube, и в Яндексе…
- **Входит в список языков, которые можно использовать на олимпиадах по программированию, при сдаче ОГЭ и ЕГЭ.**

Что нужно для работы

python-3

и/или

thonny-3.1.2. и выше

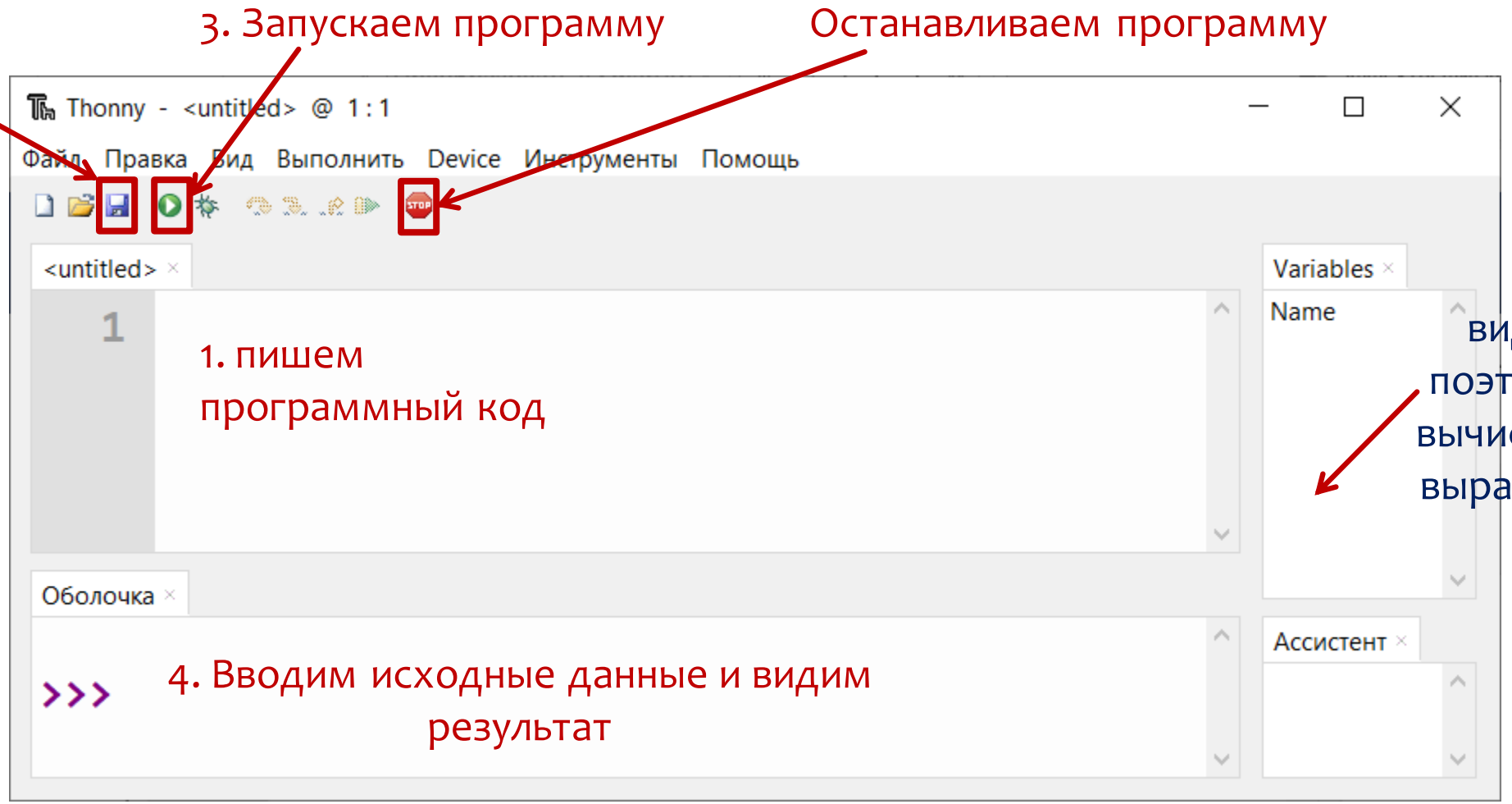


Thonny – бесплатная интегрированная среда разработки Python

2. Сохраняем программу

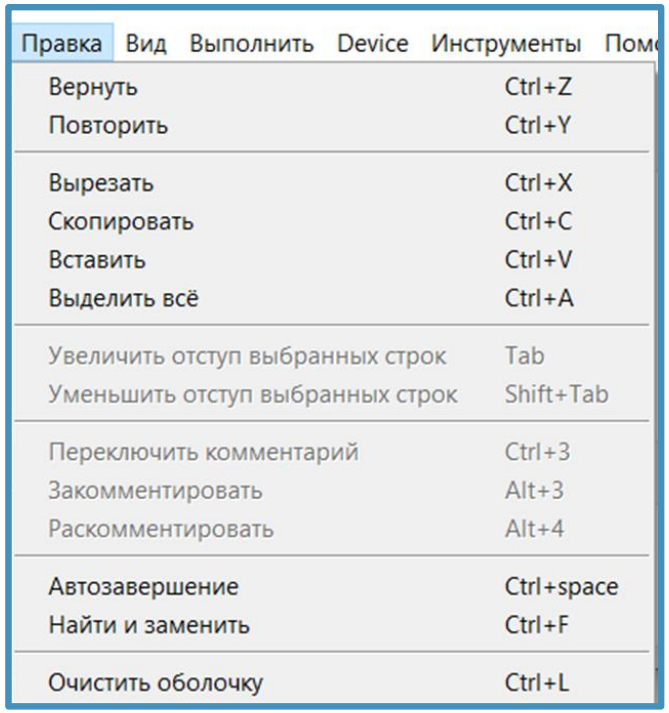
(перед первым запуском).

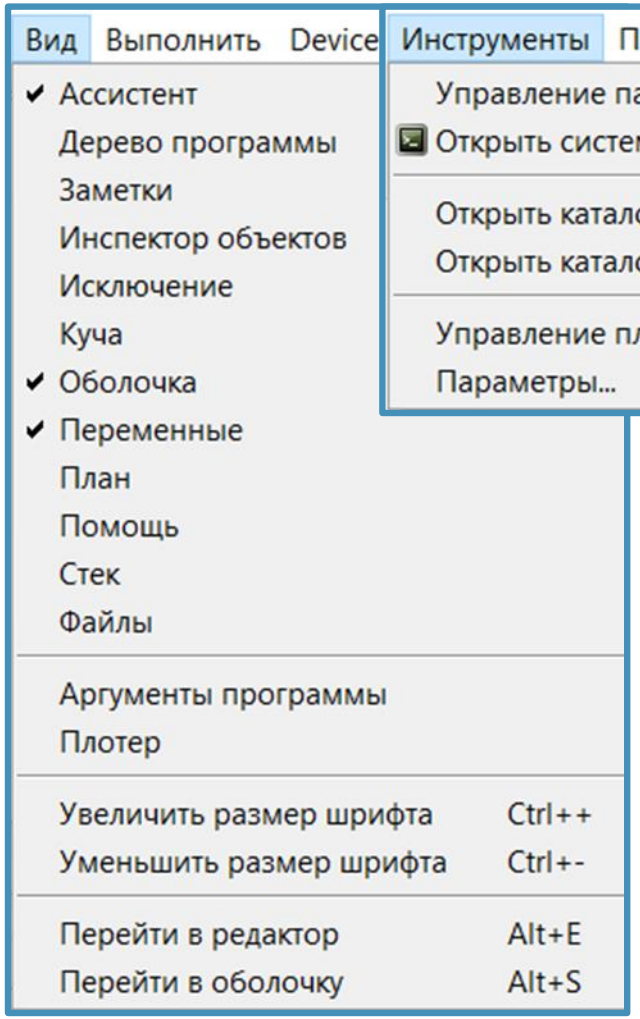
В дальнейшем, при каждом запуске программы, код сохраняется автоматически.

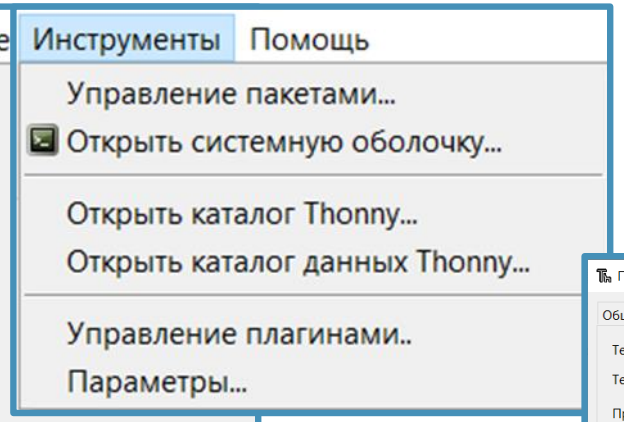


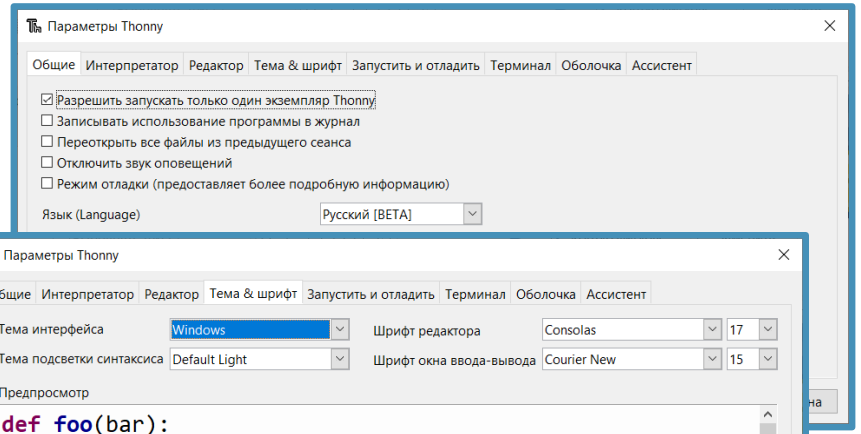
видим поэтапное вычисление выражений

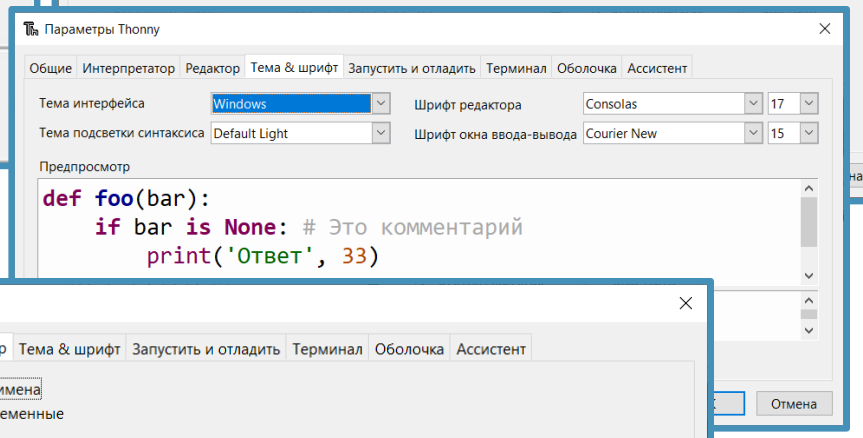
Thonny – бесплатная интегрированная среда разработки Python

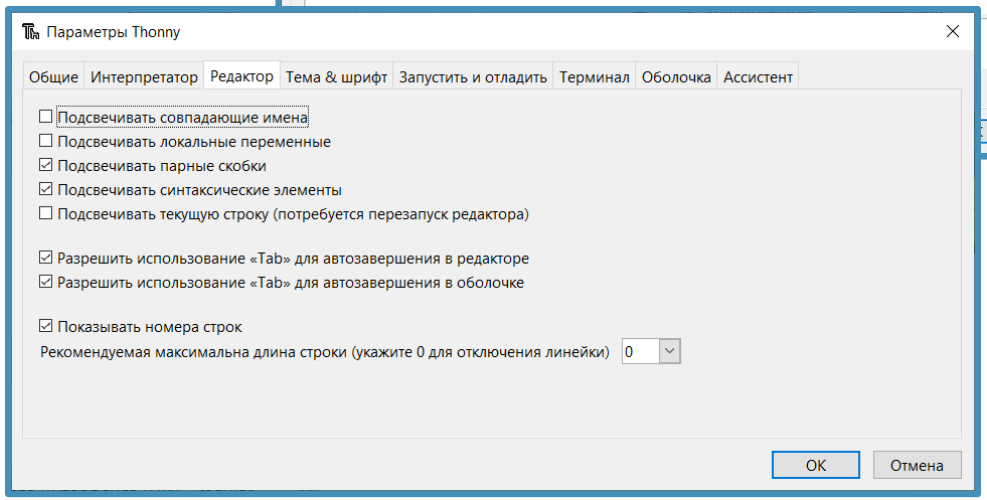












Python

ПЕРЕМЕННЫЕ И ТИПЫ ДАННЫХ.

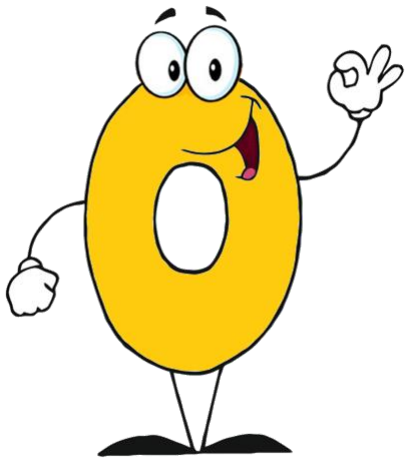
ВВОД И ВЫВОД ДАННЫХ.

СЛУЧАЙНЫЕ ЧИСЛА

Переменная

Переменная – это место в оперативной памяти для временного хранения данных. Переменная имеет имя, тип и значение.

Значение переменной может изменяться в процессе выполнения программы.



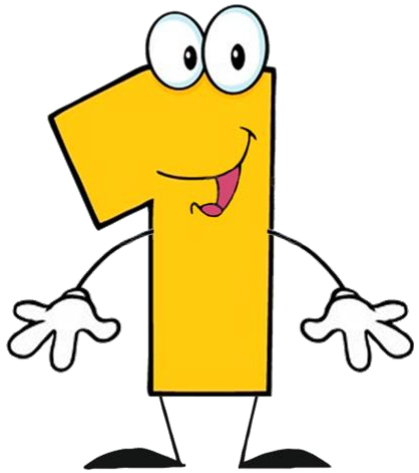
В **имени переменной** можно использовать :

- ✓ латинские буквы строчные и прописные (регистр имеет значение);
- ✓ цифры и знак подчеркивания (но начинается имя всегда с буквы).

Переменная

Для переменных:

- ✓ должны **использоваться «говорящие» имена**, чтобы можно было понять, зачем нужна та или иная переменная;
- ✓ объявлять переменные в python не нужно. **Память для переменных выделяется автоматически**.
- ✓ в языке python **при изменении значения переменной выделяется новая область памяти** и связывается с тем же именем, «старая связь» стирается безвозвратно.



Вещественные числа

- ✓ Число, в котором выделяются целая и дробная части, в информатике называется вещественным.
- ✓ Дробная часть от целой отделяются точкой (“.”).
- ✓ Очень большие или очень маленькие числа можно записывать в научном формате:

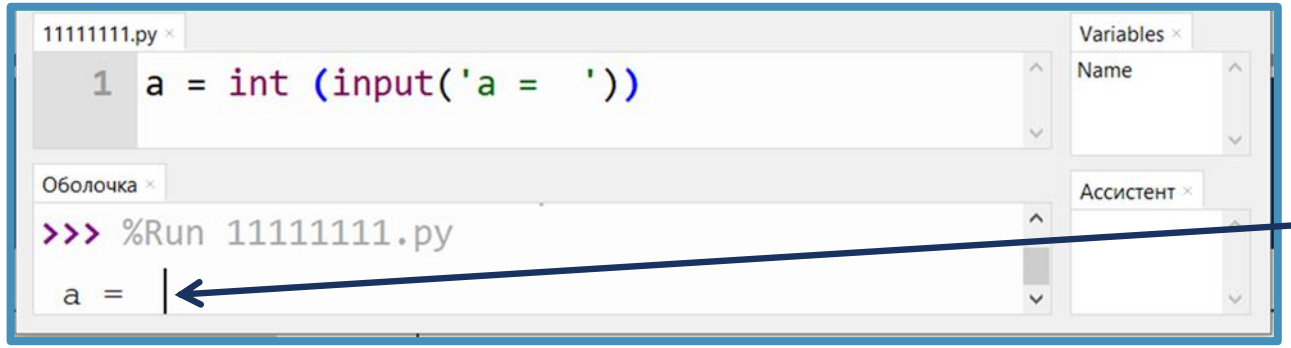
$1.60217662 \cdot 10^{-19}$	**1.60217662e-19**
$1.2345 \cdot 10^{36}$	**1.2345e+36**

Первая часть числа называется значащей частью числа или **мантиссой**.

После буквы **e** указывается **порядок числа**.

Ввод данных с клавиатуры

a = int (input('a = '))	*ввод целого числа **a** с пояснениями*
z=float (input('z = '))	*ввод вещественной переменной **z** с пояснениями*
x=str (input('x = '))	*ввод строковой переменной **x** с пояснениями*
f = bool (input('f = '))	*ввод логической переменной **f** с пояснениями*



После запуска программы выводится пояснение и в месте ввода мигает курсор.

Нужно ввести число и нажать клавишу Enter.

Ввод данных с клавиатуры

применить

разделить строку на части по пробелам

```
a, b, c = map ( int, input().split() )
```

эту операцию

к каждой части

Можно вводить через пробел несколько целых чисел и программа сама присвоит каждой переменной требуемое значение.

Прямое присвоение и генерация случайных чисел

x=15; z=20	*прямое присвоение (несколько операторов в одной строке можно разделять «;»).*
a = b = 0 a, b = 1, 2	*множественное присвоение:* *b = 0; a = b* *a = 1; b = 2*

from random import * x1 = randint(a,b)	*получение целого числа* **x1** *с помощью генератора случайных чисел*	*в интервале [a,b]* *a и b в интервал входят, но их нужно предварительно ввести одним из вышеперечисленных способов или просто написать в скобках числа*
from random import * x2 = uniform(a, b)	*получение вещественного числа* **x2** *с помощью генератора случайных чисел*	

Вывод данных на экран

print(y)	*выводится значение переменной*
print((a+b)*c)	*выводится значение выражения*
print(‘Привет!’)	*текст для вывода может записываться и в двойных, и в одинарных кавычках*
print(‘S = ‘, s)	*при выводе значение переменной можно дополнять пояснениями*

Вывод данных на экран

Вывод без разделителя и с разделителем

```
1 a = 1
2 b = 2
3 c = 3
4 print(a,b,c)
```

Оболочка

```
1 2 3
```

```
1 a = 1
2 b = 2
3 c = 3
4 print(a,b,c,sep="")
```

Оболочка

```
123
```

```
1 a = 1
2 b = 2
3 c = 3
4 print(a,b,c,sep="*")
```

Оболочка

```
1*2*3
```

```
1 a = 1
2 b = 2
3 c = 3
4 print(a,b,c,sep=",")
```

Оболочка

```
1,2,3
```

Вывод в несколько строк и в одну строку

```
1 print('2')
2 print('9')
3 print('2')
```

Оболочка

```
2
9
2
```

```
1 print('2' , end='')
2 print('9' , end='')
3 print('2')
```

Оболочка

```
292
```

По умолчанию каждая команда print осуществляет вывод с новой строки.
Для отмены дописываем **end=“”**.

Форматный вывод данных

`print("{:5} {:5} {:5}".format(x1, x2, x3))`	*Число после двоеточия – это количество позиций, которые отводятся на число*
`print("{ } { } { }".format(x1, x2, x3))`	*Данные выводятся в одной строке вплотную друг к другу*
`print("{ } + { } = { }".format(a, b, a + b))`	*Выводится пример*
При выводе вещественных значений по умолчанию выводится 16 значащих цифр	
`print("a= { : f } ".format(a))`	*:f – оставляет по умолчанию 6 цифр в дробной части числа*
`print("a= { : 7.4f } ".format(a))`	*На всё число отводится 7 позиций, из них 4 позиции на дробную часть.*

'\n' – символ перевода строки. Записывается внутри функции print. Например: **print(a, '\n\n \n ', b)**

Python

Арифметические действия с целыми и вещественными числами. Стандартные и математические функции

Арифметика

a + b	сумма
a – b	разность
a * b	произведение
a / b	частное от деления
a ** b	возведение в степень

Приоритет операций:

1. Сначала выполняются операции возведения в степень справа налево, то есть 2**3**2 – это $2^{(3^2)}$=512.
2. Далее выполняются умножения и деления слева направо.
3. Последними выполняются сложения и вычитания слева направо.

Для изменения порядка действий необходимо использовать круглые скобки.

Операции целочисленного деления и взятия остатка

- Операция деления (/) превращает любые числа в вещественные (типа float).
- Операция целочисленного деления (//) делит ту часть числа, которая делится нацело, остальное просто отбрасывает.
- Операция взятия остатка от деления (%).

```
123 % 10 = 3
123//100 = 1
123//10 % 10 = 2
```

Разбиение трёхзначного числа на отдельные цифры

```
code.py
1 print(19/3)
2 print(19//3)
3 print(19%3)
4 print((-19)//3)
5 print((-19)%3)

Оболочка
>>> %Run code.py
6.333333333333333
6
1
-7
2
>>>
```

Обратите внимание на отрицательные числа.

Операции целочисленного деления и взятия остатка

code.py

```
1 print(19/3)
2 print(19//3)
3 print(19%3)
4 print((-19)//3)
5 print((-19)%3)
```

Оболочка

```
>>> %Run code.py
6.333333333333333
6
1
-7
2
>>>
```

Обратите внимание на отрицательные числа.

19//3 = 6

18 + 1

18:3=6

19 % 3 = 1

18 + 1

-19//3 = -7

-21 (округление «вниз»!)

-21:3=-7

-19 % 3 = 2

-21 + 2

остаток ≥ 0

Рассмотрим пример

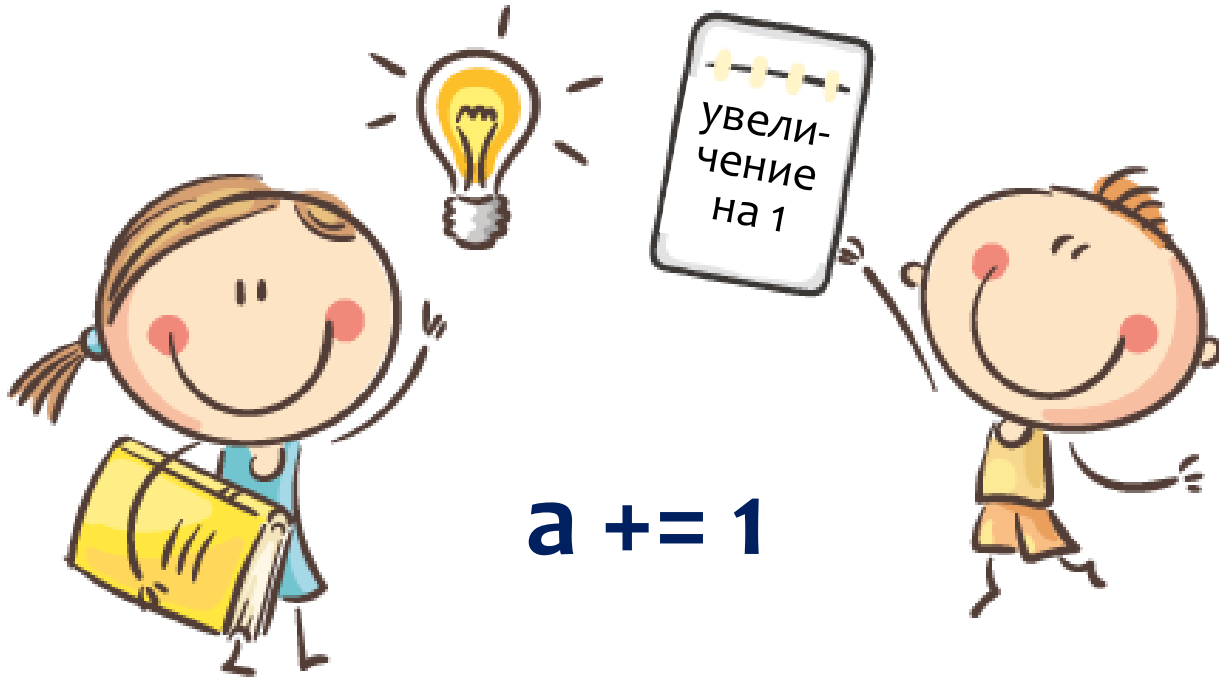
Дано трёхзначное число. Найдите сумму его цифр.

Решение

```
n = int(input('n= '))
d3 = n % 10
d2 = n // 10 % 10
d1 = n // 100
print(d1 + d2 + d3)
```

Сокращённая запись арифметических операций

a = a + b	a += b
a = a – b	a – = b
a = a * b	a *= b
a = a / b	a / = b
a = a // b	a // = b
a = a % b	a % = b



a += 1

Вещественные числа

- ✓ Число, в котором выделяются целая и дробная части, в информатике называется вещественным.
- ✓ Дробная часть от целой отделяются точкой (“.”).
- ✓ Очень большие или очень маленькие числа можно записывать в научном формате:

$1.60217662 \cdot 10^{-19}$	**1.60217662e-19**
$1.2345 \cdot 10^{36}$	**1.2345e+36**

Первая часть числа называется значащей частью числа или **мантиссой**.

После буквы **e** указывается **порядок числа**.

Стандартные функции

abs(x)	модуль числа
int (x)	отбрасывание дробной части вещественного числа
round (x)	округление вещественного числа x до ближайшего целого
bin(x)	в двоичную систему
oct(x)	в восьмеричную систему
hex(x)	в шестнадцатеричную систему

```
1 print(int(5.1))
2 print(int(5.9))
3 print(round(5.4))
4 print(round(5.5))
5 print(bin(29))
6 print(oct(29))
7 print(hex(29))
```

Оболочка

```
5
5
5
6
0b11101
0o35
0x1d
```

Математические функции

fabs(x) – модуль **x**

sqrt(x) – квадратный корень из **x**

cos(x) – косинус **x** (**x** указывается в радианах)

sin(x) – синус **x** (**x** указывается в радианах)

tan(x) – тангенс **x** (**x** указывается в радианах)

degrees(x) – конвертирует радианы в градусы

radians(x) – конвертирует градусы в радианы

ln(x) – натуральный логарифм

floor(x) – округление «вниз»

ceil(x) – округление «вверх»

Pi (pi = 3,1415926...)

e (e = 2,718281...)

code-1.py

```
1  from math import*
2  print(fabs(-55)) # модуль
3  print(sqrt(49)) # квадратный корень
4  print(cos(3)) # косинус
5  print(sin(1)) # синус
6  print(tan(2)) # тангенс
7  print(degrees(1.57)) # радианы в градусы.
8  print(radians(60)) # градусы в радианы.
9  print(pi) # pi = 3,1415926...
10 print(e) # e = 2,718281...
```

Оболочка

```
>>> %Run code-1.py
55.0
7.0
-0.9899924966004454
0.8414709848078965
-2.185039863261519
89.95437383553924
1.0471975511965976
3.141592653589793
2.718281828459045
```

Модуль **math** – предоставляет обширный функционал для работы с числами.

Основные операции над строками

Строки тоже можно складывать. Однако операция сложения для целых чисел и для строк работает по-разному: для чисел это сложение, а для строк – **конкатенация**.

A + B	конкатенация, то есть соединение строк путём размещения строки B сразу после строки A
A * n	повторение строки n раз, значение n должно быть целого типа

code.py

```
1  # Пример № 3
2  print('4'+'5')
3  print('Люблю '+'программирование')
4  print('#'* 10)
```

Оболочка

```
Python 3.8.0 (C:/Program Files/Python38/
thon.exe)
>>> %Run code.py

45
Люблю программирование
##########
```

Python

ВЕТВЛЕНИЕ

Ветвление

Разветвляющийся алгоритм – это алгоритм, содержащий хотя бы одно условие, в результате которого обеспечивается переход на один из двух возможных шагов.

При записи условий используются следующие операторы сравнения:

Обозначение	Операция	Пример	Запись на Python
==	равно	x=1	x==1
!=	не равно	x≠y	x!=y
>	больше	x+a>y	x>y
<	меньше	x<y-b	x<y
>=	больше или равно	x≥0	x>=0
<=	меньше или равно	x ≤1	x<=1

Условный переход

Условная инструкция в Python имеет следующий синтаксис:

```
if условие:
    блок инструкций 1
else:
    блок инструкций 2
```

Блок инструкций 1 будет выполнен, если условие истинно.

Если условие ложно, будет выполнен блок инструкций 2.

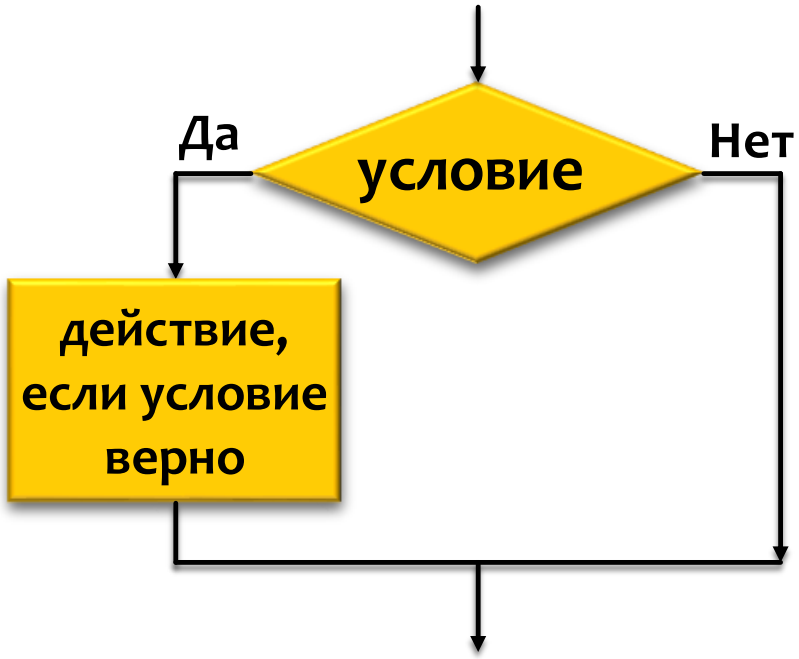
Для выделения блока инструкций, относящихся к инструкции **if** или **else**, в языке Python используются отступы.

Все инструкции, которые относятся к одному блоку, должны иметь равную величину отступа, то есть одинаковое число пробелов в начале строки.

Можно использовать в качестве отступа символ табуляции.

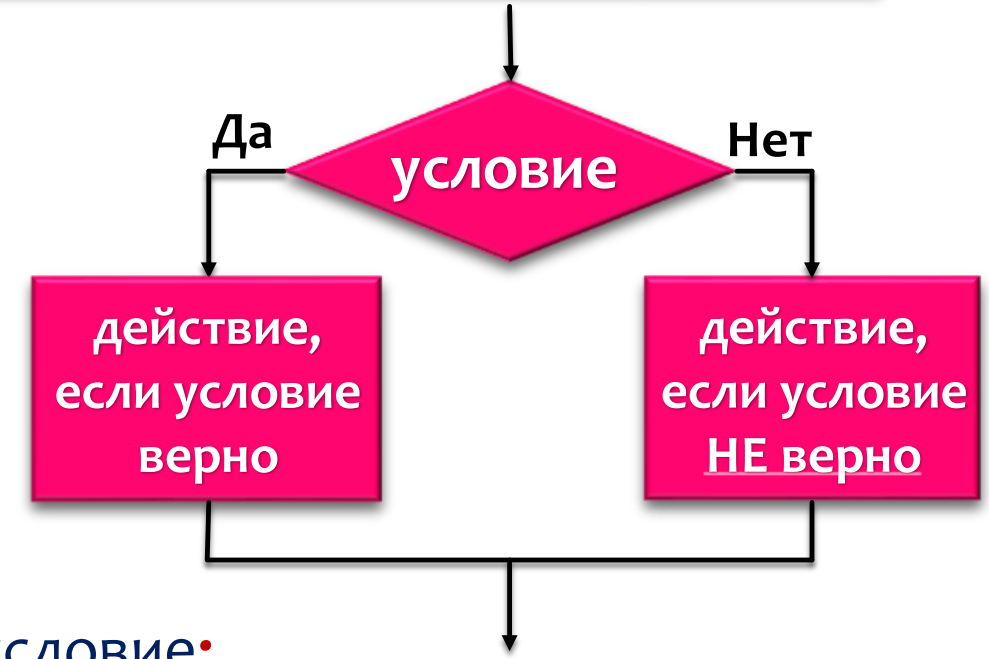
Условный переход

Неполный




```
if условие:
    действие, если условие верно
```

Полный




```
if условие:
    действие, если условие верно
else:
    действие, если условие НЕ верно
```

Сложные условия

Иногда нужно проверить одновременно не одно, а несколько условий. В этом помогают логические операции. В Python существуют стандартные логические операции: логическое И, логическое ИЛИ, логическое отрицание.

x > 0 and y < 0	**x > 0 or y < 0**	**not (x==y)**
все условия должны выполняться одновременно	должно выполняться хотя бы одно из условий	выполняется обратное условие

Приоритет выполнения операций :

1) отношения (<, >, <=, >=, ==, !=)
2) **not** («НЕ»)
3) **and** («И»)
4) **or** («ИЛИ»)

Кроме того, операции сравнения в Python можно объединять в цепочки (в отличие от большинства других языков программирования, где для этого нужно использовать логические связки), например,

x == 0 == y, 10 <= n <= 100.

Полный условный переход

Даны два числа, необходимо найти наибольшее из них и вывести его на экран (считаем, **что числа различны**).

```
a = int(input('a= '))
b = int(input('b= '))
if a > b:
    m = a
else:
    m = b
print(m)
```

Вложенные условные операторы

Найдите и выведите на экран максимальное из двух значений переменных, введённых с клавиатуры. **Если числа равны, сообщите об этом.**

```
a = int(input('a= '))
b = int(input('b= '))
if a > b:
    print('a больше')
else:
    if a == b:
        print('они равны')
    else:
        print("b больше")
```

Каскадное ветвление

Составить программу, которая по знаку арифметической операции выведет на экран вычисления в виде равенства.

elif = else if

```
1  # каскадное ветвление
2  x1=int(input("введите первое число  "))
3  x2=int(input("введите второе число  "))
4  z=str(input("введите первое число  "))
5  if z == '+':
6      print("{} + {}  = {}".format(x1, x2, x1 + x2))
7  elif z == '-':
8      print("{} - {}  = {}".format(x1, x2, x1 - x2))
9  elif z == '*':
10     print("{} * {}  = {}".format(x1, x2, x1 * x2))
11 elif z == '/':
12     if x2==0:
13         print("Делить на ноль нельзя!!!")
14     else:
15         print("{} / {}  = {}".format(x1, x2, x1 / x2))
16 else:
17   print("только +, -, * или /")
18
```

Python

ЦИКЛЫ С УСЛОВИЕМ

Понятие цикла

Цикл – это повторяющаяся последовательность действий.

Циклический алгоритм – это алгоритм, который содержит повторяющуюся последовательность действий.

Типы циклов в Python

- **цикл с условием (итерационный)**
- **цикл по переменной**

Итерация – от лат. iteratio – **повторение**

Цикл с условием (while)

Синтаксис цикла while в простейшем случае выглядит так:

```
while условие:
    блок инструкций
```

Цикл с условием while позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно.

При выполнении цикла while сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается, и управление передаётся на следующую инструкцию после тела цикла while .

Если условие истинно, то выполняются все инструкции из блока, после чего условие проверяется снова, и снова выполняется блок инструкций. Так продолжается до тех пор, пока условие будет истинно. Как только условие становится ложным, работа цикла завершается, и управление передаётся следующей инструкции после цикла.

Как правило, цикл while используется, когда невозможно определить точное количество проходов исполнения цикла.

Рассмотрим пример

Сколько всего знаков * будет выведено после исполнения фрагмента программы:

```
i = 0
while i < 5:
      print('*')
      if i % 2 == 0:
            print('**')
      if i > 2:
            print('***')
      i = i + 1
```

```
1  i = 0
2  while i < 5:
3      print('*')
4      if i % 2 == 0:
5          print('**')
6      if i > 2:
7          print('***')
8      i = i + 1
```

Оболочка

```
*
**
*
*
**
*
***
*
**
***
```

Изменения в работе цикла

break – оператор завершения цикла

```
i=0
while i<5:
    a=int(input())
    b=int(input())
    if (a==0) and (b==0):
        break  #досрочно завершаем цикл
    print (a*b)
    i+=1
```

Если вместо **break** вписать **quit()**,
то будет завершена вся программа.

continue – оператор перехода к следующей итерации цикла (если она есть)

```
i=0
while i<5:
    a=int(input())
    b=int(input())
    if (a==0) and (b==0):
        break   #досрочно завершаем
цикл
    if (a==0) or (b==0):
        continue   #переходим к
следующей итерации
    print (a*b)
    i+=1
```

Обработка потока данных

Напишите программу, которая определяет сумму последовательности целых чисел. Программа получает на вход целые числа, количество введённых чисел не известно, последовательность чисел заканчивается числом 0 (0 – признак окончания ввода, не входит в последовательность). Количество чисел не превышает 1000. Введённые числа по модулю не превышают 30 000.

В данной задаче не нужно сохранять все данные в памяти, а можно добавлять их к сумме по одному.

Пусть x – последнее введённое число, s – сумма.

```
s=0
x=int(input('x= '))
while x!=0:
    s=s+x
    x=int(input('x= '))
print('s= ', s)
```

Бесконечные циклы

```
while True:
    блок инструкций
```

В некоторых задачах используются циклы, условие в которых всегда истинно.

Поскольку условие **True** всегда истинно, такой цикл никогда не завершится. Но завершить его всё-таки можно, если применить специальный оператор **break**.

Рассмотрим на примере нахождения суммы чисел потока данных.

```
s=0
x=int(input('x= '))
while x!=0:
    s=s+x
    x=int(input('x= '))
print("s= ", s)
```

```
s=0
while True:
    x=int(input('x= '))
    if x==0: break
    s+=x
print('s= ', s)
```

условие выхода

- ✓ *при входе в цикл условие **не проверяется***
- ✓ *цикл всегда выполняется **хотя бы один раз***

Наименьшее число в последовательности

```
x=int(input('x= '))
if x!=0:
    min=x
while x!=0:
    x=int(input('x= '))
    if x<min and x!=0:
        min=x
print('min= ', min)
```

Напишите программу, которая в последовательности натуральных чисел определяет минимальное число. Программа получает на вход натуральные числа, количество введённых чисел неизвестно, последовательность чисел заканчивается числом 0 (0 – признак окончания ввода, не входит в последовательность). Количество чисел не превышает 1000. Введённые числа по модулю не превышают 30000.

Python

ЦИКЛЫ ПО ПЕРЕМЕННОЙ

Цикл по переменной (for)

Цикл **for**, также называемый циклом по переменной или циклом с параметром, удобно использовать когда количество повторений цикла известно или может быть вычислено заранее.

В цикле **for** указывается **переменная** и **множество значений**, которые может принимать переменная.

Множество значений может быть задано:

- ✓ списком;
- ✓ диапазоном;
- ✓ строкой.

Множество значений задано списком

1 Вывод на экран номеров и соответствующих им названий цветов радуги.

```
1  i = 1
2  for color in 'красный', 'оранжевый', 'желтый', 'зеленый', 'голубой', 'синий', 'фиолетовый':
3      print(i,'-й',  ' цвет радуги - ', color, sep = '')
4      i += 1
```

Оболочка

```
1-й цвет радуги - красный
2-й цвет радуги - оранжевый
3-й цвет радуги - желтый
4-й цвет радуги - зеленый
5-й цвет радуги - голубой
6-й цвет радуги - синий
7-й цвет радуги - фиолетовый
```

2

```
1  for i in 2, 3, 5, 7, 11:
2      print(i ** 2)
```

Оболочка

```
4
9
25
49
121
```

Вывод на экран квадратов первых пяти простых чисел.

Множество значений задано списком и строкой

3 Вывод на экран типов элементов списка (как их распознаёт Python).

```
1 for i in 1, 3, True, 2.345, 'one':
2     print(i,type(i))
3
```

Оболочка

```
1 <class 'int'>
3 <class 'int'>
True <class 'bool'>
2.345 <class 'float'>
one <class 'str'>
```

4 Вывод на экран последовательности букв в строке.

```
1 for c in 'python':
2     print(c)
```

Оболочка

```
p
y
t
h
o
n
```

Множество значений задано диапазоном (функция range() с одним параметром)

5

```
1  n=6
2  for i in range(n):
3      print(i)
```

Оболочка ×

```
0
1
2
3
4
5
```

Вывод на экран чисел от 0 до 5.

Чтобы вывести на экран числа от **0** до **n–1**, можно использовать цикл **for** вместе с функцией **range()**.

В качестве **n** может использоваться числовая константа или переменная.

Если значение **n** равно **нулю** или **отрицательное**, то тело цикла **не выполнится** ни разу.

Множество значений задано диапазоном (функция range() с двумя параметрами)

5

```
1  a=5
2  b=10
3  for i in range(a,b):
4          print(2**i)
```

Оболочка ×

```
32
64
128
256
512
```

Вывод на экран степеней двойки в интервале от a до (b-1).

Если требуется задать цикл от некоторого числа a до некоторого числа b-1, то можно использовать функцию range() с двумя параметрами.

Если же b ≤ a, то цикл не будет выполнен ни разу.

6

```
1  a=1
2  b=100
3  s = 0
4  for i in range(a, b):
5      s += i
6  print(s)
```

Оболочка ×

```
4950
```

Вывод на экран суммы чисел от a до (b-1).

Множество значений задано диапазоном (функция range() с тремя параметрами)

Чтобы организовать цикл, в котором индексная переменная будет уменьшаться или увеличиваться на величину отличную от 1, необходимо использовать функцию range() с тремя параметрами.

7

```
1  a=5
2  b=0
3  d=-1
4  for i in range(a, b, d):
5      print(i)
6  print('Поехали!')
```

Оболочка

```
5
4
3
2
1
Поехали!
```

Вывод на экран обратного отсчёта.

8

```
1  a=0
2  b=100
3  d=7
4  s = 0
5  for i in range(a, b, d):
6      s += i
7      print(i)
8  print('сумма чисел, кратных ',d, ' равна ',s)
```

Оболочка

```
0
7
14
21
28
35
42
49
56
63
70
77
84
91
98
сумма чисел, кратных  7  равна  735
```

Вывод на экран чисел кратных 7 в интервале от 0 до 99 и их суммы.

Волшебное слово else

SpisokFor-2.py

```
1  my_list = [1, 2, 3, 4, 5]
2  k=int(input('Что ищем? '))
3  for i in my_list:
4      if i == k:
5          print("Элемент найден!")
6          break
7  else:
8      print("Такого элемента нет!")
```

Оболочка

```
Что ищем? 5
Элемент найден!
>>> %Run SpisokFor-2.py
Что ищем? 6
Такого элемента нет!
```

Слово else, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же "естественным" образом.

Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

Python

КОМПЬЮТЕРНАЯ ГРАФИКА

Общие сведения

Программы, с которыми мы работали до сих пор, выводили текст в консольное окно, где невозможно рисовать. Если требуется написать программу с графическими возможностями, обычно используют библиотеки.

Библиотека – это набор готовых функций (команд), расширяющих возможности языка программирования. Одна из библиотек, позволяющих работать с графикой – **tkinter** – устанавливается вместе с интерпретатором языка Python.

Библиотека **graph**, с которой будем работать мы, с одной стороны, использует возможности tkinter, а с другой – упрощает работу для начинающих пользователей.

Простейшая графическая программа состоит всего из двух строк:

from graph import* ← *подключает к программе все возможности модуля graph*

run() ← *запускает рисование и открывает графическое окно*

Система координат

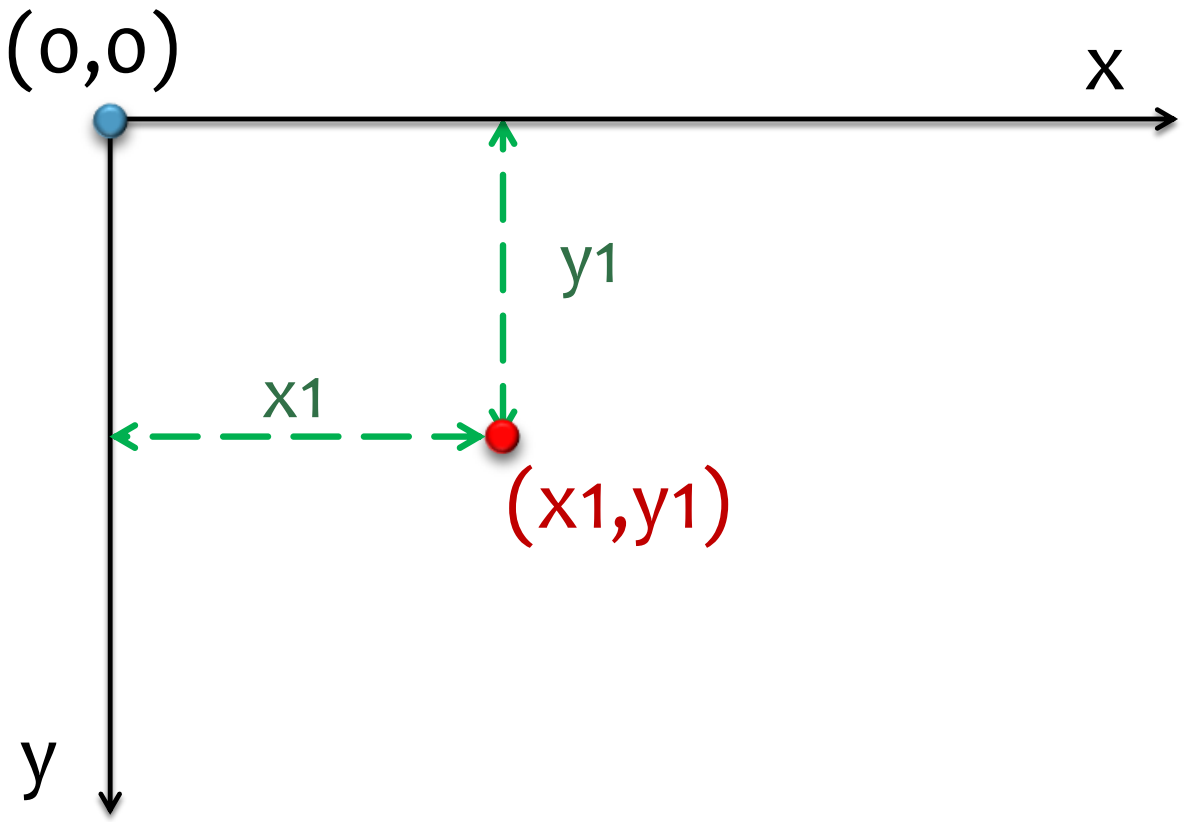
Поле для рисования в графических программах называется холстом (canvas).

Размер холста совпадает с размером графического окна.

Если рисовать что-то за пределами холста, эта часть рисунка будет потеряна.

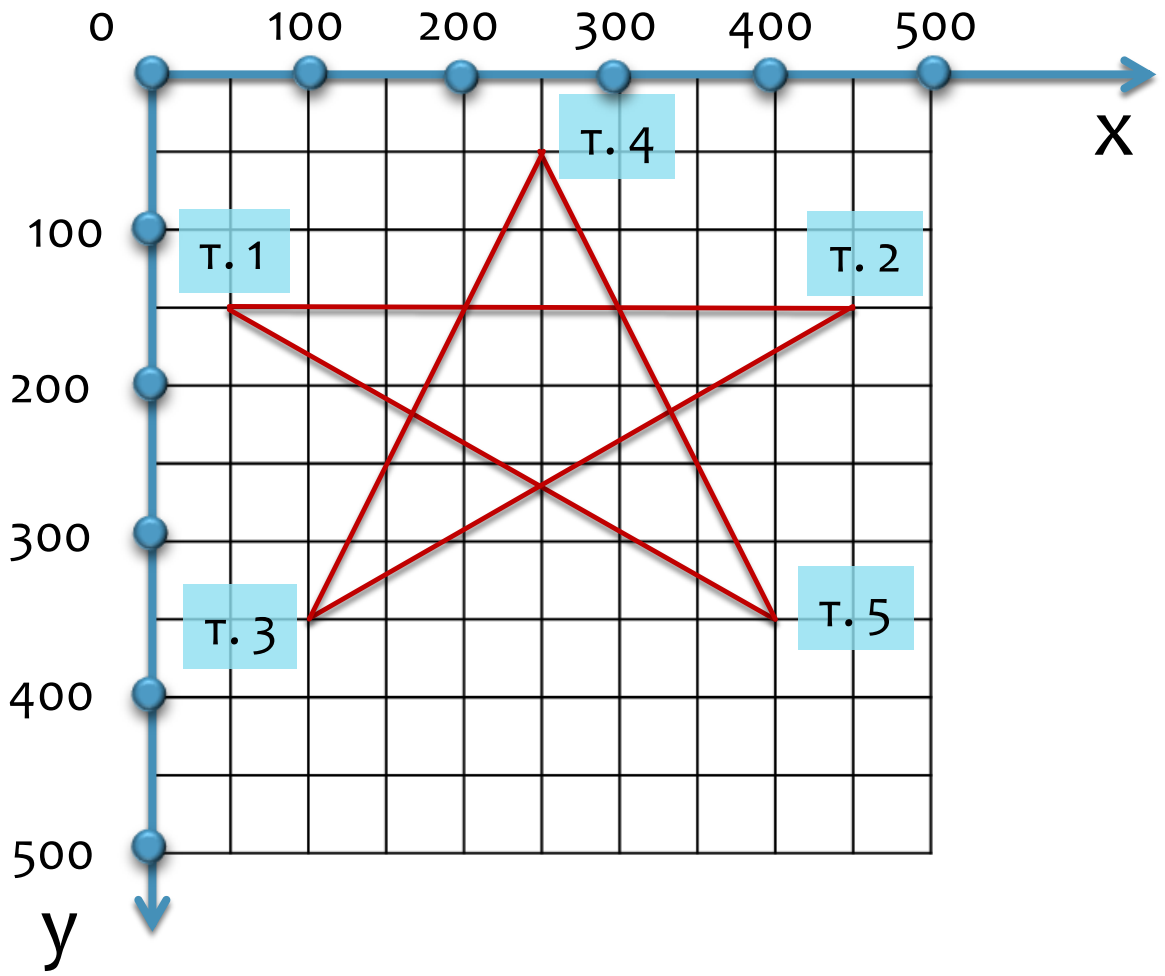
Холст – это прямоугольник, состоящий из отдельных пикселей.

Каждый пиксель имеет две координаты (x, y).



Рассмотрим пример

№ точки	Координата x	Координата y
т.1	50	150
т.2	450	150
т.3	100	350
т.4	250	50
т.5	400	350



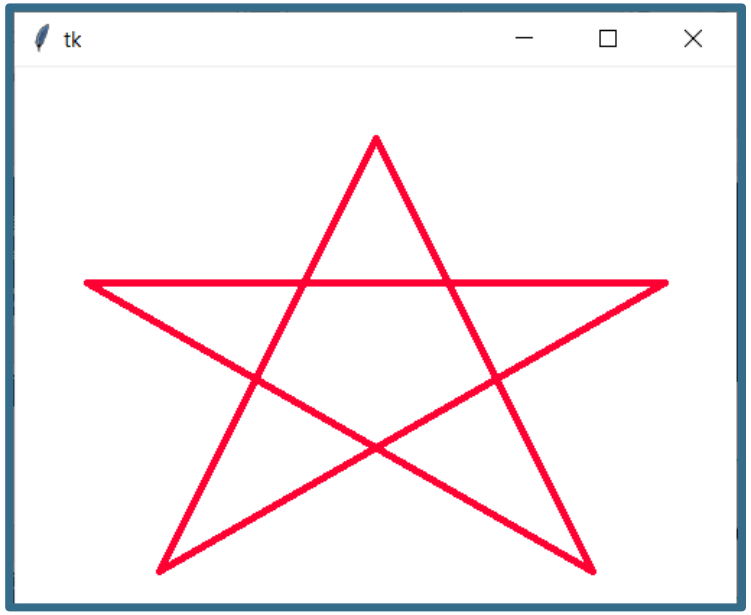
Рассмотрим пример

01_s.py

```
1  from graph import *
2  penColor(255,0,51)
3  penSize(5)
4  polygon([(50,150), (450,150), (100,350),
5      (250,50), (400,350)])
6  run()
```

подключение всех функций модуля graph

цвет пера и толщина пера

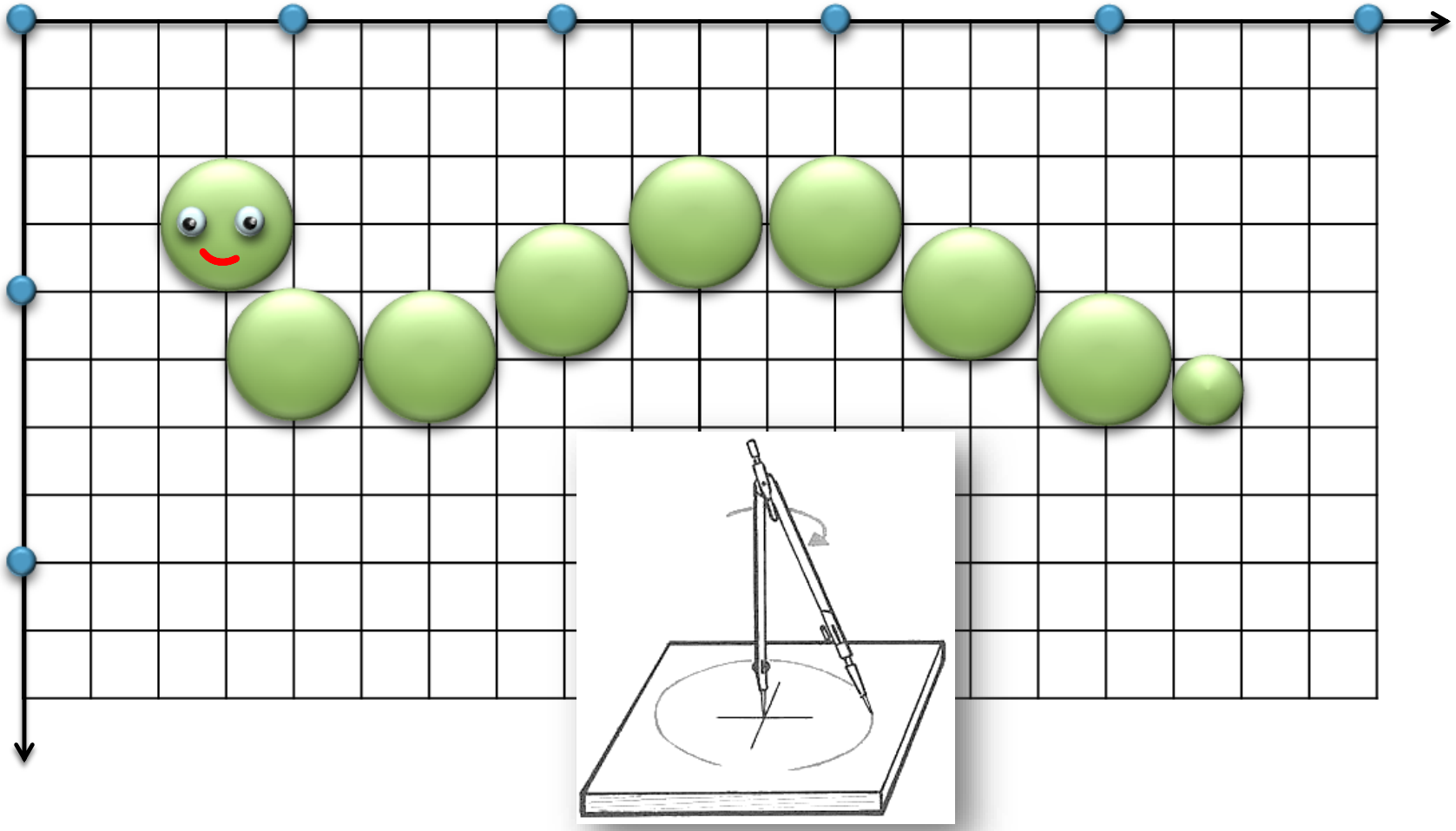


№ точки	Координата x	Координата y
т.1	50	150
т.2	450	150
т.3	100	350
т.4	250	50
т.5	400	350

команда **polygon** строит замкнутую ломанную линию по точкам

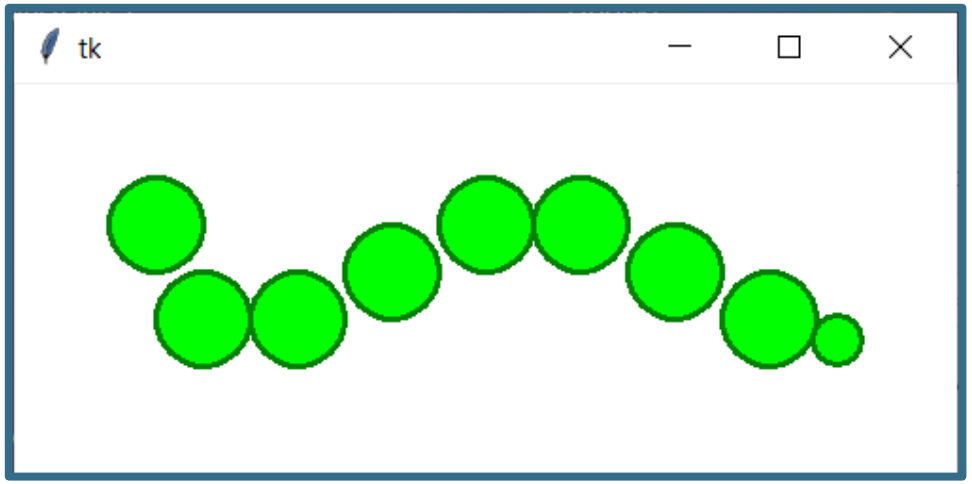
Рассмотрим пример

№	x	y	r
1	75	75	25
2	100	125	25
3	150	125	25
4	200	100	25
5	250	75	25
6	300	75	25
7	350	100	25
8	400	125	25
9	436	136	13

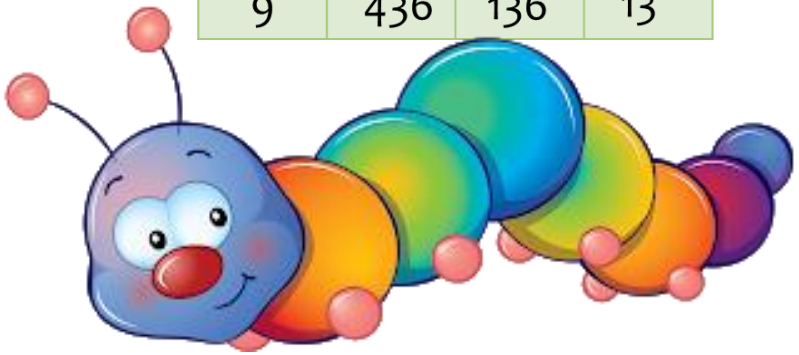


Рассмотрим пример

```
1  from graph import*
2  penColor("green")
3  penSize(3)
4  brushColor("lime")
5  circle(75,75,25)
6  circle(100,125,25)
7  circle(150,125,25)
8  circle(200,100,25)
9  circle(250,75,25)
10 circle(300,75,25)
11 circle(350,100,25)
12 circle(400,125,25)
13 circle(436,136,13)
```



№	x	y	r
1	75	75	25
2	100	125	25
3	150	125	25
4	200	100	25
5	250	75	25
6	300	75	25
7	350	100	25
8	400	125	25
9	436	136	13



Управление цветом

Цвет в формате RGB

```
penColor( 255, 0, 255 )
```

Red
0...255

Green
0...255

Blue
0... 255

Цвет по названию

```
penColor("red")
```

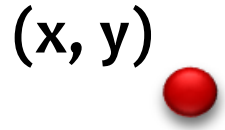
white, black, gray, navy, blue, cyan, green, yellow, red, orange, brown, maroon, violet, purple, ...

Линии и заливка

Цвет линий:	**penColor("red")**
Толщина линий:	**penSize(2)**
Цвет заливки:	**brushColor("green")**

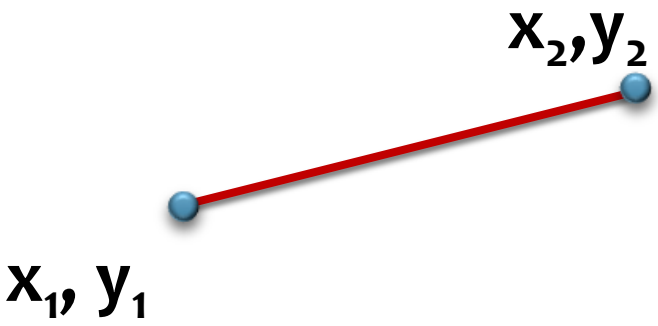
Графические примитивы
(простейшие фигуры)

Точка



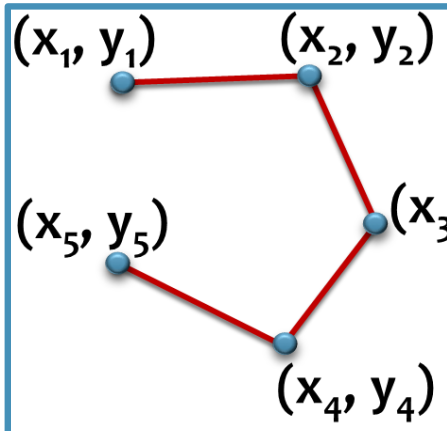
point(x, y)

Линия



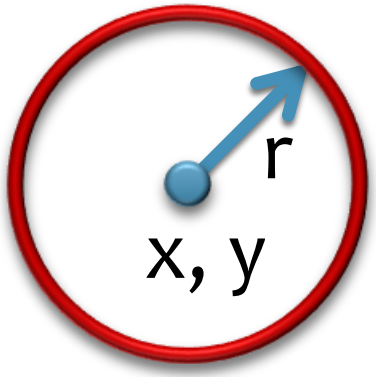
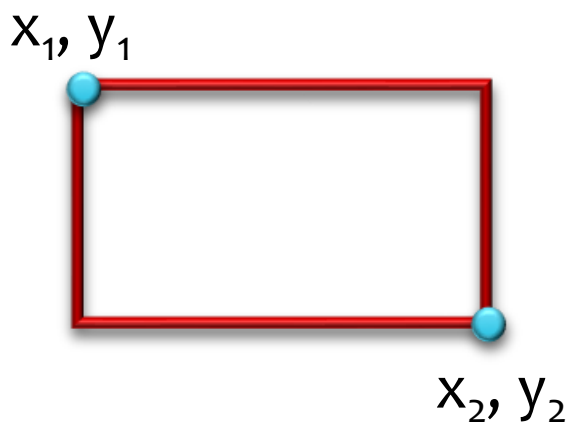
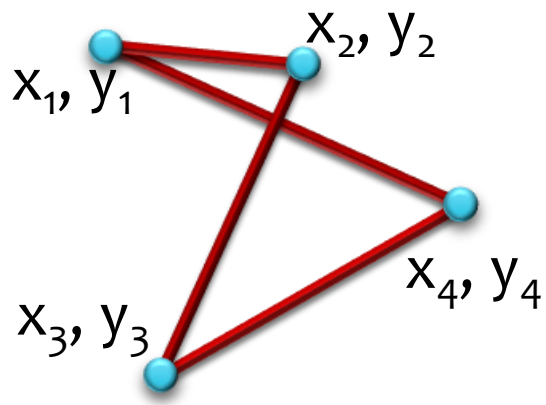
line(x1, y1, x2, y2)

Непрерывный ввод



moveTo(x1, y1)
lineTo(x2, y2)
lineTo(x3, y3)
lineTo(x4, y4)
lineTo(x5, y5)

Графические примитивы
(простейшие фигуры)

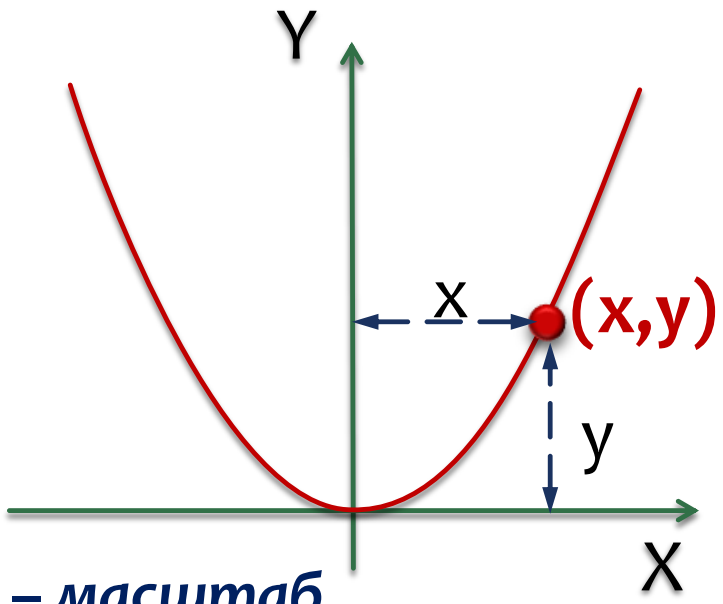
Окружность	Прямоугольник	Полигон
		
circle(x, y, r)	rectangle(x1, y1, x2, y2)	polygon([(x1, y1), (x2, y2), (x3, y3), (x4, y4)])

Python

ГРАФИКА В ЦИКЛАХ

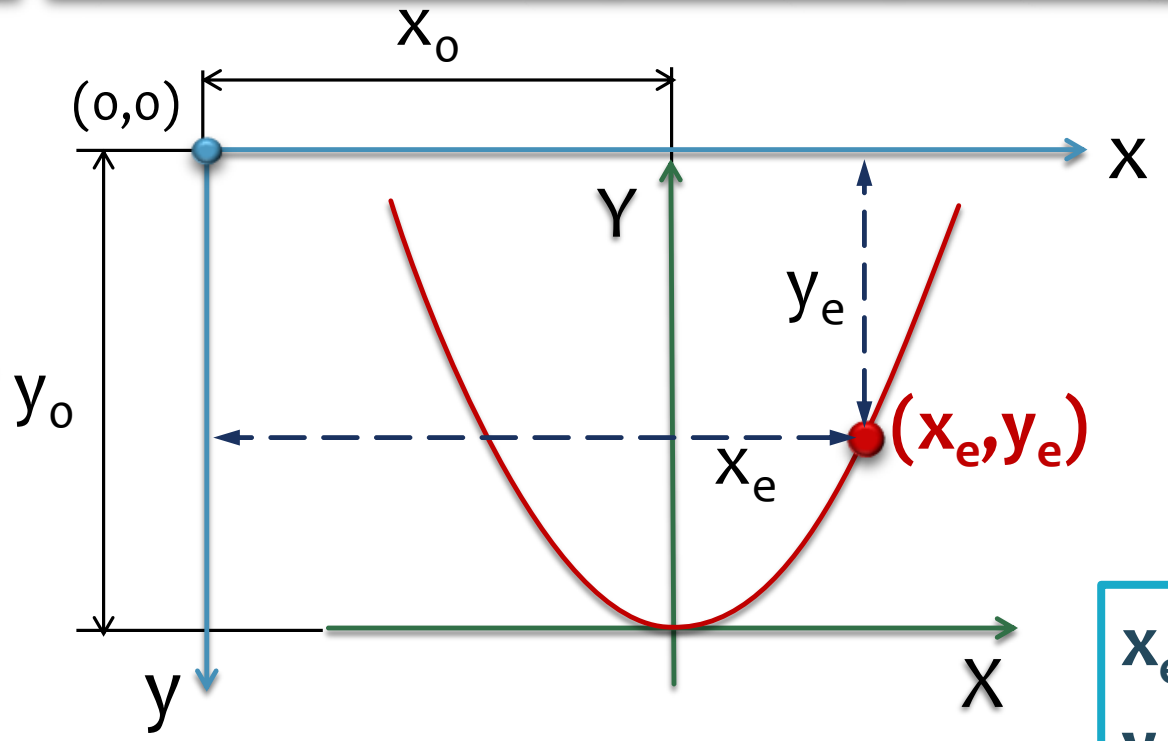
Преобразование математической системы координат в экранную

Математическая система координат



k – масштаб
(длина изображения единичного отрезка на экране)

Экранная система координат (пиксели)



$$x_e = x_o + kx$$
$$y_e = y_o - ky$$

Рассмотрим пример

Задача: построить график функции $y = x^2$ на отрезке от -2 до 2.

Анализ:

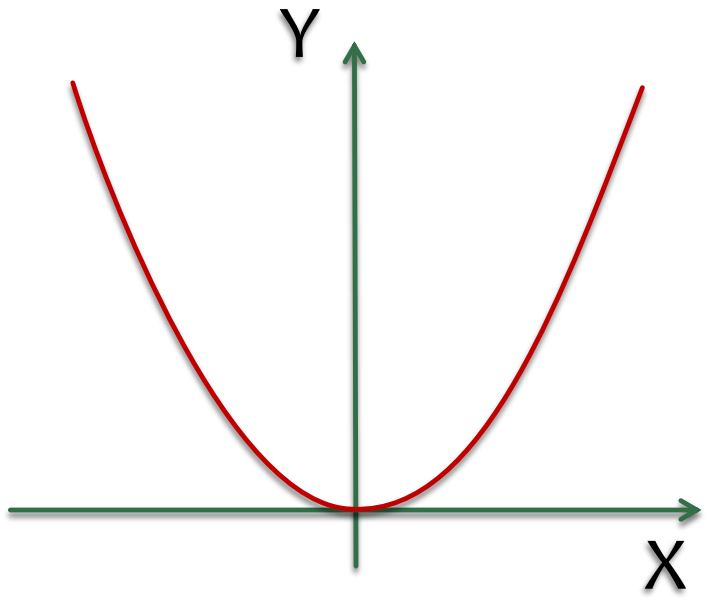
максимальное значение

$y_{max} = 4$ при $x = \pm 2$

минимальное значение

$y_{min} = 0$ при $x = 0$

Проблема: функция задана в математической системе координат, строить надо на экране, указывая координаты в пикселях.



Рассмотрим пример

Начало координат (высота окна – 600, ширина окна – 500)

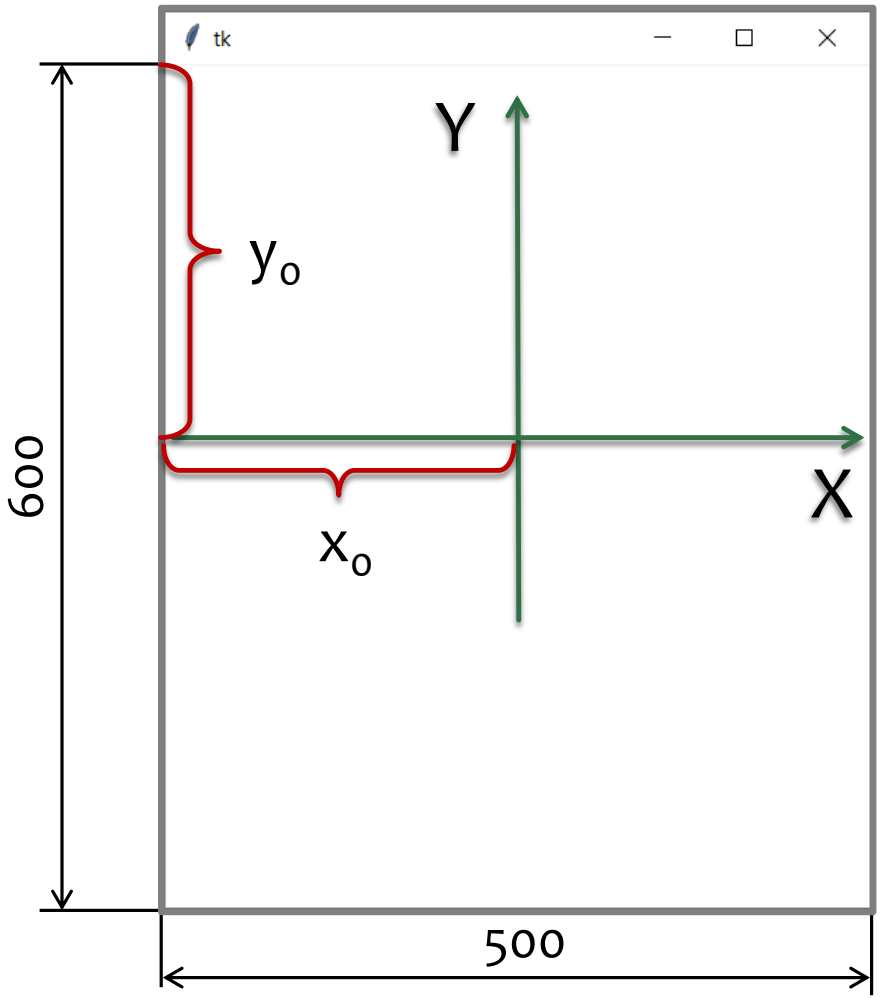
x0 = 250

y0 = 300

Координатные оси

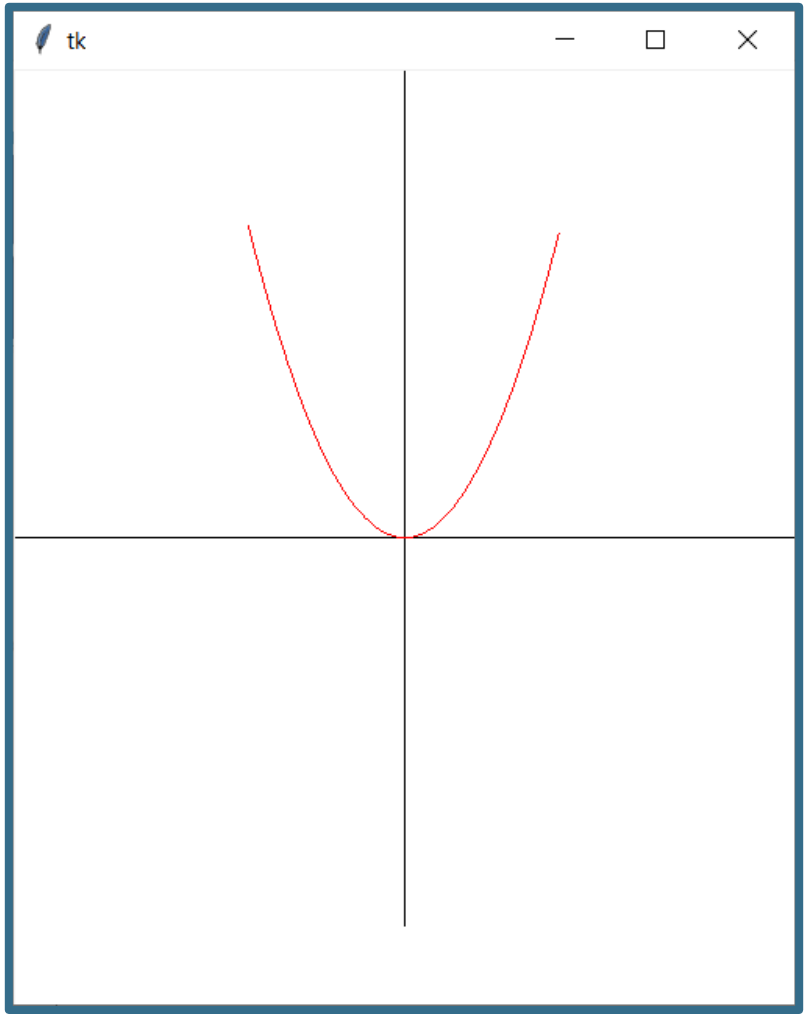
line(0, y0, x0+250, y0)

line(x0, 0, x0, y0+250)

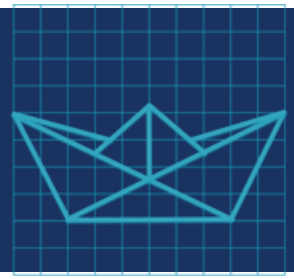


Рассмотрим пример

```
from graph import *
x0 = 250 # начало координат
y0 = 300 # начало координат
k = 50   # масштаб
xmin = -2; xmax = 2 # пределы по x
line(0, y0, x0+250, y0) # горизонтальная ось
line(x0, 0, x0, y0+250) # вертикальная ось
points = [] # пустой массив для записи координат точек
x = xmin  # начальное значение x
h = 0.02  # шаг изменения x
penColor("red")
while x <= xmax:
  y = x*x
  xe = x0 + k*x # экранные координаты (в пикселях)
  ye = y0 - k*y # экранные координаты (в пикселях)
  points.append((xe, ye)) # добавляем точку в массив
  x += h # к следующей точке
polyline(points)# рисуем линию
run()
```



Разберитесь как работает программа



```
from graph import *
def update():
  moveObjectBy(obj, 5, 0)
  moveObjectBy(obj1, 5, 0)
  moveObjectBy(obj2, 5, 0)
  moveObjectBy(obj3, 5, 0)
  if xCoord(obj) >= 600-20:
close()
def keyPressed(event):
  if event.keycode == VK_ESCAPE:
    close()
```

```
brushColor("blue")
rectangle(0, 0, 600, 800)
penSize(5)
x=80
penColor(50,110,255)
obj = polygon([(x,160), (x+120,160),
(x+160,80), (x,160), (x-40,80),
(x+120,160)])
obj1 = polygon([(x+20,110), (x+60,75),
(x+100,110), (x+60,130), (x+60,75)])
obj2 = line(x+160,80, x+90, 100)
obj3 = line(x-40,80, x+30, 100)
onKey(keyPressed)
onTimer(update, 50)
run()
```

Python

ПРОЦЕДУРЫ И ФУНКЦИИ

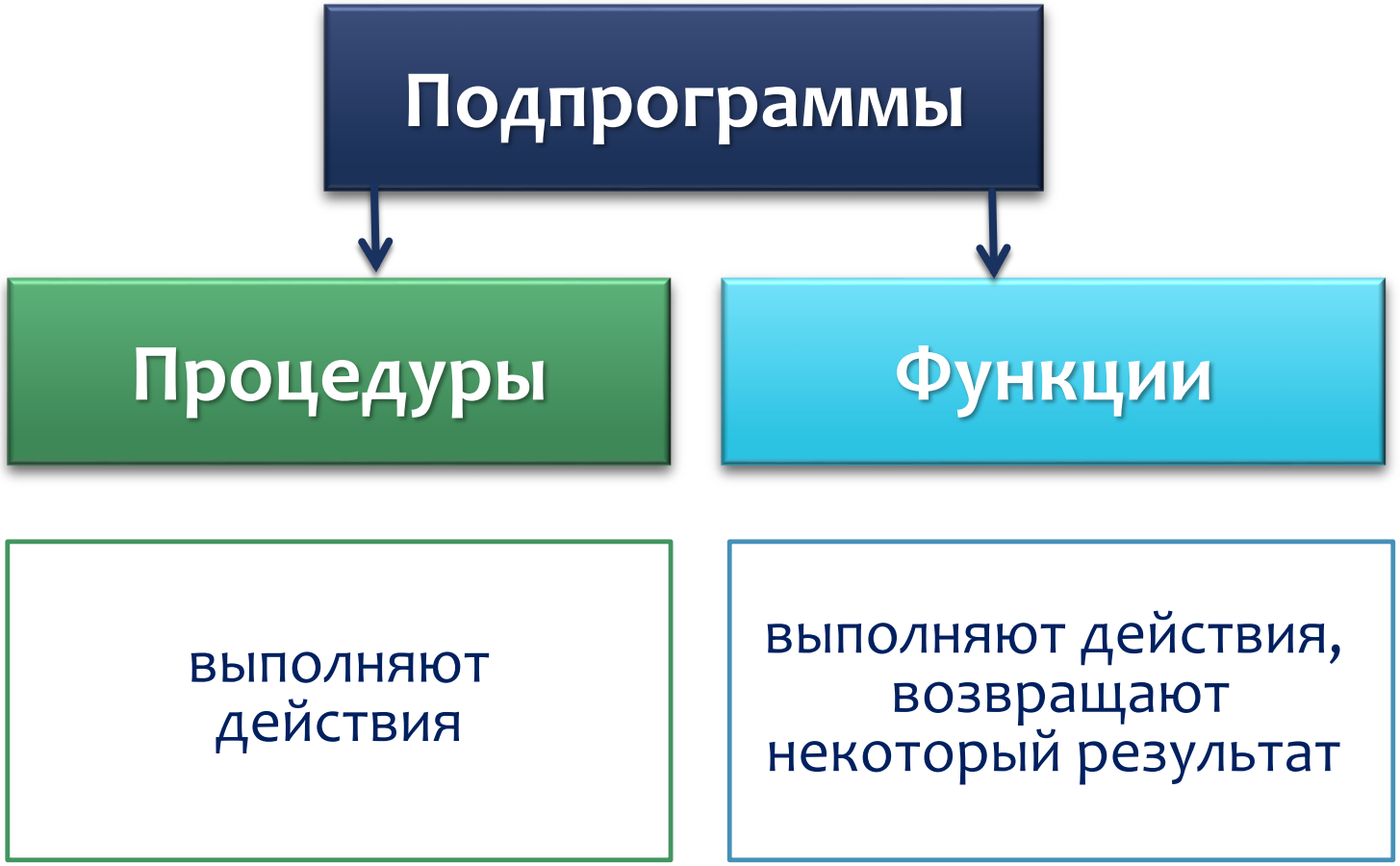
Подпрограммы

Подпрограмма – это отдельная функционально независимая часть программы, имеющая имя и решающая отдельную задачу.

Подпрограммы:

- ✓ избавляют от необходимости многократно повторять в тексте программы аналогичные фрагменты;
- ✓ улучшают структуру программы, облегчая ее понимание;
- ✓ позволяют распределять большие задачи между несколькими разработчиками или стадиями проекта;
- ✓ повышают устойчивость к ошибкам программирования и непредвидимым последствиям при модификациях программы.

Два типа подпрограмм



В Python нет формального разделения подпрограмм на функции и процедуры (как, например, в Паскале или Си), и процедурой можно считать функцию, возвращающую пустое значение – в основном используется единственный термин – функция.

Общий вид процедуры

define – определить

имя процедуры, даётся тем, кто пишет программу

```
def name_procedury(параметры):
    тело процедуры
```

Параметр – это переменная, от значения которой зависит работа подпрограммы.

если параметры не нужны ставят пустые скобки ()

пишут в скобках имена одной или нескольких переменных через запятую: (n) или (a,b,c,d)

Вызов процедуры

При вызове процедуры нужно в скобках передать ей фактические значения параметров.

Аргумент – это значение параметра, которое передаётся подпрограмме при её вызове.

Аргументом может быть не только постоянное значение (число, символ), но также переменная, и даже арифметическое выражение.

```
name_procedury()
```

или

```
name_procedury(n)
```

или

```
name_procedury(a,b,c,d)
```

Простая процедура

```
1  def printLine():
2      print("----------------------------------")
3  printLine()
4  print('эта процедура отделяет текст линиями')
5  printLine()
6  printLine()
```

процедура

основная часть программы

вызов процедуры

Оболочка

```
----------------------------------
эта процедура отделяет текст линиями
----------------------------------
----------------------------------
```

✓можно вызывать сколько угодно раз

✓нет дублирования кода

✓изменять в одном месте

Процедура с параметрами

```
1  def printLine(n):
2        print("=) "*n)
3  n=int(input('n= '))
4  printLine(n)
5  print('эта процедура рисует смайлики')
6  printLine(n)
```

Оболочка

```
n= 7
=) =) =) =) =) =) =)
эта процедура рисует смайлики
=) =) =) =) =) =) =)
```

символьная строка

```
1  def printLine(a,n):
2        print(a*n)
3  a=str(input('a= '))
4  n=int(input('n= '))
5  printLine(a,n)
6  print('эта процедура выводит символы')
7  printLine(a,n)
```

Оболочка

```
a= (*)
n= 10
(*)(*)(*)(*)(*)(*)(*)(*)(*)(*)
эта процедура выводит символы
(*)(*)(*)(*)(*)(*)(*)(*)(*)(*)
```

Общий вид функции

define – определить

имя функции, даётся тем, кто пишет программу

```
def name_function(параметры):
    тело функции
    return (результат)
```

Выражение, стоящее после ключевого слова return будет возвращаться в качестве результата вызова функции.

Результат вызова функции можно:

- ✓ присвоить переменной,
- ✓ использовать его в качестве операндов математических выражений, т.е. составлять более сложные выражения.

Функция – это вспомогательный алгоритм, который возвращает результат (число, строку символов и др.).

Функция

Пример. Написать функцию, которая вычисляет среднее арифметическое двух целых чисел.

исходные данные

```
def Avg(a, b):
    return (a+b)/2
a = 2; b = 5
print(Avg(a,b))
```

результат функции

Ещё пример.

```
1  def Avg(a, b):
2      return (a+b)/2
3  a = int(input('a= '))
4  b = int(input('b= '))
5  print(Avg(a,b))
6  if Avg(a,b) > 5:
7      print("Да!")
8  else:
9      print("Нет!")
```

Оболочка

```
a= 5
b= 4
4.5
Нет!
```

Что делает программа?

Функция

```
1  def nod1 (a,b):
2      while a != 0 and b != 0:
3          if a > b:
4              a = a % b
5          else:
6              b = b % a
7      return a + b
8  print(nod1(int(input()),int(input())))
```

Оболочка ×

```
45
125
5
```

Пример. Написать функцию, которая вычисляет наименьший общий делитель двух целых чисел.

Глобальные и локальные переменные

Переменные, которые введены в основной программе, называются **глобальными** (общими). Их **могут использовать все подпрограммы** (процедуры и функции).

Переменные, которые используются только внутри процедуры или функции, называются **локальными** (местными). К ним можно обращаться **только внутри этой подпрограммы, остальные** подпрограммы и основная программа **их не видят.** Такой прием называется **инкапсуляцией** (от латинского «помещение в капсулу»).

Локальная переменная создается только при вызове процедуры или функции. Как только работа подпрограммы будет закончена, все локальные переменные будут удаляться из памяти.

Имена локальных переменных в каждой подпрограмме можно выбирать независимо от имён локальных переменных других подпрограмм.

Глобальные и локальные переменные

def show():
  print(s)
	def showLocal():
	  s = 7
	  print(s)
		def showGlobal():
		  global s
		  s = 7
		  print(s)

Процедура выводит значение **s**. После запуска транслятор сначала ищет локальную переменную с таким именем – её нет. Потом он начинает искать глобальную переменную: если такая переменная есть, на экран выводится её значение, если нет – будет выдано сообщение об ошибке.	Даже если существует глобальная переменная **s**, в первой строчке этой процедуры будет создана новая локальная переменная **s**, и её значение (7) появится на экране.	Эта процедура работает с глобальной переменной **s**. Она присвоит ей новое значение 7 (это «увидят» все остальные подпрограммы) и выведет его на экран.

Нужно стараться **писать подпрограммы, которые не обращаются к глобальным переменным.**

Глобальные и локальные переменные

В данном коде переменная b в функции локальная или глобальная?

```
def f(b):
    b += 0
    print(b)
b = 5
f(b)
```

Не запуская код, ответьте на вопрос: что выведет на экран данная программа?

```
def f(a):
    global b
    b += 3
    print(a + b)
b = 2
f(b)
```

```
def f():
    global a
    b = 2
    a, b = b, a
    print(a, b, end = " ")
a = 1
b = 2
f()
print(a, b, end = " ")
```

Глобальные и локальные переменные

Не запуская код, выберите, какие из программ во время запуска получат ошибку выполнения.

```
def f():
    a = a
    print(a)
a = 5
f()
```

```
def f():
    print(a)
    a = a
a = 5
f()
```

```
def f(a):
    a = a
    print(a)
a = 5
f(a)
```

```
def f():
    print(a)
    global a
a = 5
f()
```

Глобальные и локальные переменные

Не запуская код, ответьте на вопрос: что выведет на экран данная программа?

```
def f():
    global a
    global b
    b, c = a, b
def g():
        global a
        global d
        c = '0'
        a = d + c
```

```
a = '2'
b = '3'
c = '5'
d = '7'
f()
g()
f()
print(a + b + c + d)
```

Функция

```
def binary(n):
    s = ''
    while n > 0:
        s = str(n%2) + s
        n //= 2
    return s

while 1:
    n = int(input())
    if n != 0:
        print(binary(n))
    else:
        break
```

Оболочка

```
7
111
5
101
6
110
```

Пример. Функция перевода десятичного числа в двоичное.

В основной ветке программы будем выполнять бесконечный цикл, в котором

1. запрашивается десятичное число,

2. если оно не ноль, то вызывается функция перевода его в двоичное представление и выводится результат работы функции на экран,

3. иначе (когда введен 0) будем прерывать цикл оператором break.

Функция

```
def triangle(a, b, c):
    return a + b > c and a + c > b and b + c > a

a = int(input())
b = int(input())
c = int(input())
d = int(input())
if triangle(a,b,c) or triangle(a,b,d) \
  or triangle(a,c,d) or triangle(b,c,d):
    print("YES")
else:
    print("NO")
```

Оболочка

```
5
4
3
6
YES
```

Пример. Вам даны 4 отрезка. Выведите YES, если среди них найдутся 3, из которых можно составить треугольник, и NO в противном случае.

Для решения напишите функцию triangle(a, b, c), которая будет возвращать True, если из трёх заданных отрезков можно составить треугольник, и False иначе.

Задачи

1) Напишите процедуру, которая принимает параметр – натуральное число N – и выводит на экран две линии из N символов "–".

2) Напишите процедуру, которая принимает один параметр – натуральное число *N*, – и выводит на экран прямоугольник длиной *N* и высотой 3 символа.

3) Напишите процедуру, которая выводит на экран треугольник со стороной N символов. При запуске программы N нужно ввести с клавиатуры.

Пример:

```
Длина цепочки: 7
-------
-------
```

Пример:

```
Длина прямоугольника: 7
ooooooo
o     o
ooooooo
```

Пример:

```
Сторона: 5
o
oo
ooo
oooo
ooooo
```

Задачи

4) Напишите функцию, которая вычисляет среднее арифметическое пяти целых чисел.
5) Напишите функцию, которая находит количество цифр в десятичной записи числа.
6) Напишите функцию, которая находит количество нулей в двоичной записи числа (с клавиатуры вводится десятичное число).
7) Найти периметр треугольника, заданного координатами своих вершин. (Определить функцию для расчета длины отрезка по координатам его вершин.)
8) Найти все трехзначные простые числа. (Определить функцию, позволяющую распознавать простые числа.)

Найти все трехзначные простые числа

```
def prost(x):
    for i in range(2,x):
        if x % i == 0:
            return 0
    return 1

a=int(input('a= '))
b=int(input('b= '))
for i in range(a,b):
    if prost(i)==1:
            print(i, end='   ')
```

Задачи

9) Получить все шестизначные счастливые номера. Счастливым называют такое шестизначное число, в котором сумма его первых трёх цифр равна сумме его последних трёх цифр. (Определить функцию для расчета суммы цифр трёхзначного числа.)

10) Даны шесть различных чисел. Определить максимальное из них. (Определить функцию, находящую максимум из двух различных чисел.)

11) Составить программу, в результате которой величина a меняется значением с величиной b, а величина c – с величиной d. (Определить процедуру, осуществляющую обмен значениями двух переменных величин.)

Задачи

12) Даны стороны двух треугольников. Найти сумму их периметров и сумму их площадей. (Определить процедуру для расчета периметра и площади **треугольника по его сторонам.)**

13) Даны основания и высоты двух равнобедренных трапеций. Найти сумму их периметров и сумму их площадей. (Определить процедуру для расчета периметра и площади равнобедренной трапеции по ее основаниям и высоте.)

14) В зависимости от выбора пользователя вычислить площадь круга, прямоугольника или треугольника. Для вычисления площади каждой фигуры должна быть написана отдельная функция.

15) Напишите функцию, которая возвращает количество цифр в восьмеричной записи числа.

Задачи

16) Напишите процедуру с параметрами **n** и **s**, которая выводит квадрат размером **n**x**n** из символа, который вводится с клавиатуры. Используя эту процедуру, напишите программу, которая запрашивает два значения – строну квадрата и символ, и вызывает процедуру рисования требуемого квадрата

17) Напишите процедуру, которая выводит на экран все делители числа **N** в одну строку через пробел. Используя данную процедуру, составьте программу, которая для всех введенных натуральных чисел (числа вводятся до 0, 0 - признак окончания ввода) выводит делители текущего числа.

18) Напишите логическую функцию isByte, которая возвращает значение True, если переданное ей число помещается в 8-битную ячейку памяти (вспомните, какое минимальное и какое максимальное числа можно записать с помощью 8 бит).

Задачи

19) Напишите логическую функцию pointInRect, которая возвращает значение True, если точка с заданными координатами находится внутри прямоугольника, для которого заданы координаты верхнего левого и правого нижнего углов. Стороны прямоугольника параллельны осям координат.

20) Напишите логическую функцию pointInTriangle, которая возвращает значение True, если точка с заданными координатами находится внутри треугольника, заданного координатами трёх своих вершин.

21) На соревнованиях выступление спортсменов оценивают пять экспертов, каждый из них выставляет оценку в баллах (целое число от 0 до 100). Для получения итоговой оценки лучшая и худшая из оценок экспертов отбрасываются, а для остальных трёх находится среднее арифметическое. Напишите функцию, которая принимает пять оценок экспертов и возвращает итоговую оценку спортсмена.

Рекурсия

Рекурсия – это определение объекта через такой же объект (или объекты), но с другими параметрами.

Рекурсивная подпрограмма вызывает саму себя напрямую или через другие подпрограммы.

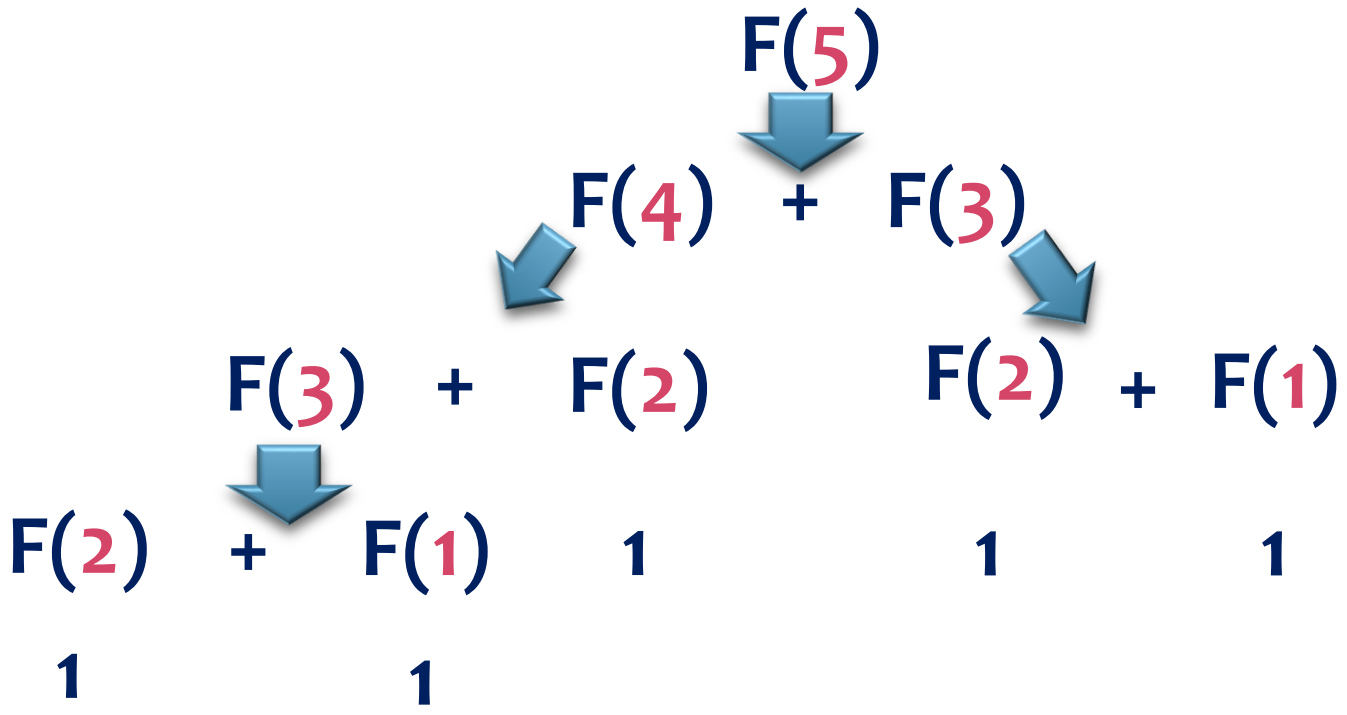
Количество вложенных вызовов функции или процедуры называется **глубиной рекурсии**. По умолчанию глубина рекурсии в языке Питон ограничена 1000 вызовов.

Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причем без явных повторений частей программы и использования циклов.

Рекурсия

Рассмотрим рекурсию на примере:

```
def F(n):
        if n > 2:
                return F(n-1)+ F(n-2)
        else: return 1
n=int(input())
print(F(n))
```



Примеры

Рассмотрим пример функции вычисления факториала.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
n=int(input('n= '))
print(factorial(n))
```

Программирование рекурсии выглядит так:

- ✓ Функция должна сначала проверить, не является ли переданный набор параметров простым (крайним) случаем. В этом случае функция должна вернуть значение (или выполнить действия), соответствующие простому случаю.
- ✓ Иначе функция должна вызвать себя рекурсивно для другого набора параметров, и на основе полученных значений вычислить значение, которое она должна вернуть.

Примеры

Вычисление суммы натуральных чисел от 1 до n.

```
def sum(n):
    if n == 1:
        return 1
    else:
        return n + sum(n-1)
```

Быстрое возведение в степень

```
def power(a, n):
    if n == 0:
        return 1
    elif n % 2 == 1:
        return power(a, n - 1) * a
    else:
        return power(a, n // 2) ** 2
```

Задачи

Чему равна сумма всех чисел, напечатанных на экране при выполнении вызова F(1)?

```
def F(n):
    print(n)
    if n < 5:
        F(n + 1)
        F(n + 3)
```

Чему будет равно значение, вычисленное при выполнении вызова F(6)?

```
def F(n):
    print(n)
    if n > 1:
        F(n - 1)
        F(n - 3)
```

```
def F(n):
  if n > 2:
    return F(n-1) + G(n-2)
  else: return n
def G(n):
  if n > 2:
    return G(n-1) + F(n-2)
  else: return n+1
```

Python

СИМВОЛЬНЫЕ СТРОКИ

Символьные строки

Символьная строка – это последовательность символов, которая рассматривается как единый объект.

Строки в языке Python являются объектами типа **str**.

```
1  a="Люблю "
2  b='программировать'
3  c=' на языке '
4  lang = input('укажите язык программирования  ')
5  s=a+b+c+lang # операция сложения (конкатенация)
6  print ('*' * 30 ) # умножение строки на число
7  print(s)
8  print ('*' * 30 ) # умножение строки на число
```

Оболочка

```
укажите язык программирования  python
******************************
Люблю программировать на языке python
******************************
```

Переменной можно присвоить значение, являющееся строкой, с помощью двойных или одинарных кавычек.

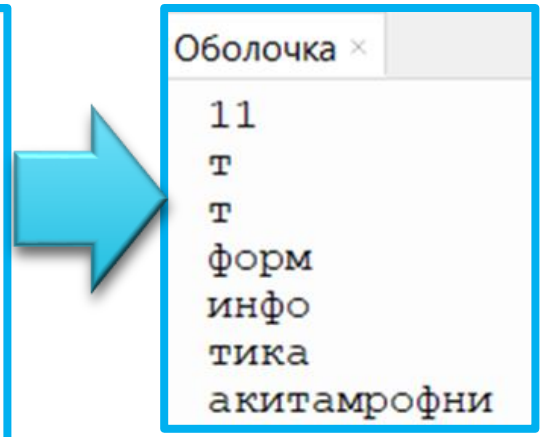
Символьные переменные можно складывать, можно умножать на число…

Символьные строки

В Python каждый символ строки имеет свой номер (индекс), причём нумерация всегда начинается в нуля.

0	1	2	3	4	5	6	7	8	9	10
и	н	ф	о	р	м	а	т	и	к	а
s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]	s[10]

```
1  s="информатика"
2
3  print(len(s)) # - длина строки
4  print(s[7])   # печать символа «т».
5  print(s[-4])  # печать символа «т» (отрицательный индекс – отсчет с конца).
6
7  #Срезы строк
8  print(s[2:6])       # "форма"
9  print(s[:4])        # "инфо" (от начала строки)
10 print(s[7:])        # "тика"  (от конца строки)
11
12 # реверс строки
13 print(s[::-1])
```



Срез S[: : 2] вернёт символы с чётными индексами, а срез S[1 : : 2] – с нечётными.

Символьные строки. Методы

Метод – это функция, применяемая, в данном случае, к строке.
Метод вызывается в виде: Имя_объекта.Имя_метода(параметры).

```
1  S="информатика"
2  S3='Люблю тебя, Петра творенье,'
3  S2="C:/Catalog/file"
4  print(S.find("a")) # возвращает индекс первого вхождения искомой подстроки.
5  print(S.rfind("a")) # возвращает индекс последнего вхождения искомой подстроки или -1.
6  print(S.replace("И", "и")) #  заменить в строке S все вхождения подстроки "и" на подстроку "И".
7  print(S.count("a")) # подсчитывает сколько раз строка "a" встречается внутри строки S
8  print(S.replace('a', '*')) # заменяет одну подстроку на другую
9  print(S2.split("/"))# разбиение строки по разделителю '/'
10 print(S3.split(" "))# разбиение строки по пробелу
11
12 S1=list(S)# превращает строку в список символов
13 S1.sort() # сортирует список по алфавиту
14 print(S1)
15
16 S1.reverse() # сортирует список по алфавиту
17 print(S1)
18
19 print("".join(S1)) # формирует из списка строку
```

Оболочка

```
6
10
информатика
2
информ*тик*
['C:', 'Catalog', 'file']
['Люблю', 'тебя,', 'Петра', 'творенье,']
['а', 'а', 'и', 'и', 'к', 'м', 'н', 'о', 'р', 'т', 'ф']
['ф', 'т', 'р', 'о', 'н', 'м', 'к', 'и', 'и', 'а', 'а']
фтронмкииаа
```

Рассмотрим пример

Дана строка. Получите новую строку, вставив между всеми парами соседних символов исходной строки символ *.

```
1  s=input()
2  s1=''
3  for i in range(len(s)):
4      s1=s1 + s[i]+'*'
5  print(s1[:-1])
```

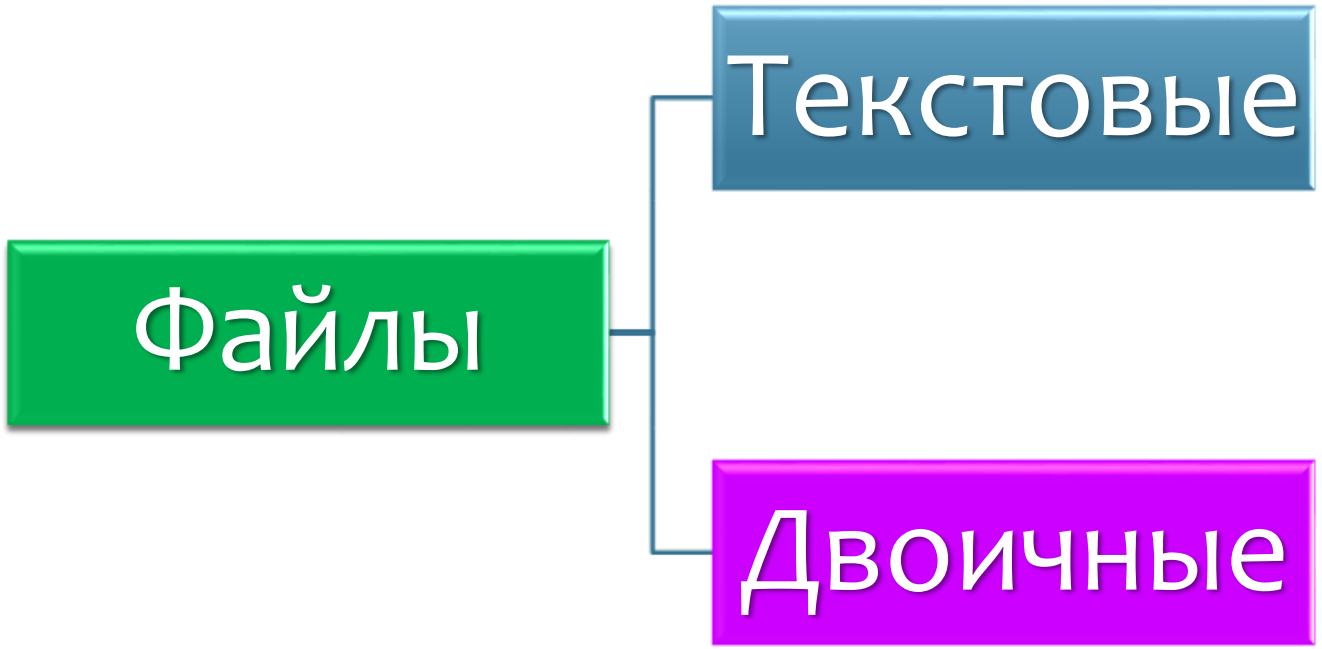
```
Оболочка
python
p*y*t*h*o*n
```

Python

РАБОТА С ТЕКСТОВЫМИ ФАЙЛАМИ

Файлы

Файл – это набор данных во внешней памяти, имеющий имя.



- текст, разбитый на строки;
- из специальных символов только символы перехода на новую строку (“\n”).

- любые двоичные данные: рисунки, звуки, видео, …

Работа с текстовыми файлами

Работа с файлом из программы включает три этапа:

1. открытие файла (открытый файл блокируется так, что другие программы не могут одновременно использовать его);
2. работа с файлом;
3. закрытие файла (именно при закрытии, все сделанные программой в этом файле изменения записываются на диск).

по умолчанию – на чтение (режим **"r"**)

```
Fin = open ( "input.txt" )
Fout = open ( "output.txt",  "w" )
  # здесь работаем с файлами
Fin.close()
Fout.close()
```

"r" – чтение
"w" – запись
"a" – добавление

Ввод данных

```
Fin = open( "input.txt" )
```

Чтение строк:

```
s = Fin.readline()

s = Fin.readline().split()
```

чтение строки и разбивка по пробелам

Чтение целых чисел:

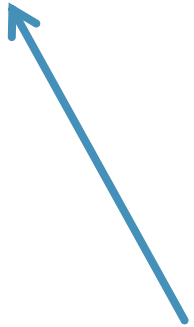
```
s = Fin.readline().split()
a, b = int(s[0]), int(s[1])

a, b = [int(x) for x in s]

a, b = map( int, s )
```

Вывод данных

```
a = 1
b = 2
Fout = open( "output.txt", "w" )
Fout.write ( "{:d} + {:d} = {:d}\n".format(a, b, a+b) )
Fout.close()
```



Все данные для записи в файл необходимо преобразовать в строку!

Рассмотрим пример

В файле записано в столбик неизвестное количество чисел. Найти их сумму.

```
sum = 0
for s in open ( "input.txt" ):
    sum += int(s)
```

Рассмотрим пример

В файле записаны в столбик целые числа. Вывести в другой текстовый файл те же числа, отсортированные в порядке возрастания.

Ввод:

```
s = Fin.read().split()
A = list ( map(int, s) )
```

Сортировка:

```
A.sort()
```

Вывод:

```
Fout = open ( "output.txt", "w" )
Fout.write ( str(A) )
Fout.close()
```

→ [1, 2, 3]

или:

```
for x in A:
    Fout.write ( str(x)+"\n" )
```

→

1
2
3

или:

```
for x in A:
    Fout.write ( "{:4d}".format(x) )
```

→ 1 2 3