# Программирование на языке Python

§ 54. Введение в язык § 54. Введение в язык Python
§ 55. Вычисления
§ 56. Ветвления
§ 57. Циклические алгоритмы
§ 58. Циклы по переменной
§ 59. Процедуры
§ 60. Функции
§ 61. Рекурсия

© К.Ю. Поляков, Е.А. Ерёмин, 2018 http://kpolyakov.spb.ru

# Программирование на языке Python

## § 54. Введение в язык Python

# Простейшая программа

```
# Это пустая программа
```

? Что делает эта программа?

комментарии после # не обрабатываются

кодировка utf-8 по умолчанию)

```
# -*- coding: utf-8 -*-
# Это пустая программа
```

*Windows:* `cp1251`

```
"""
Это тоже комментарий
"""
```

# Вывод на экран

```
print ( "2+2=?" )
print ( "Ответ: 4" )
```

автоматический переход на новую строку

**Протокол:**

```
2+2=?
Ответ: 4
```

```
print ( '2+2=?' )
print ( 'Ответ: 4' )
```

# Сложение чисел

**Задача.** Ввести с клавиатуры два числа и найти их сумму.

**Протокол:**

```
Введите два целых числа
25 30
25+30=55
```

компьютер

пользователь

компьютер считает сам!

?

1. Как ввести числа в память?
2. Где хранить введенные числа?
3. Как вычислить?
4. Как вывести результат?

# Сумма: псевдокод

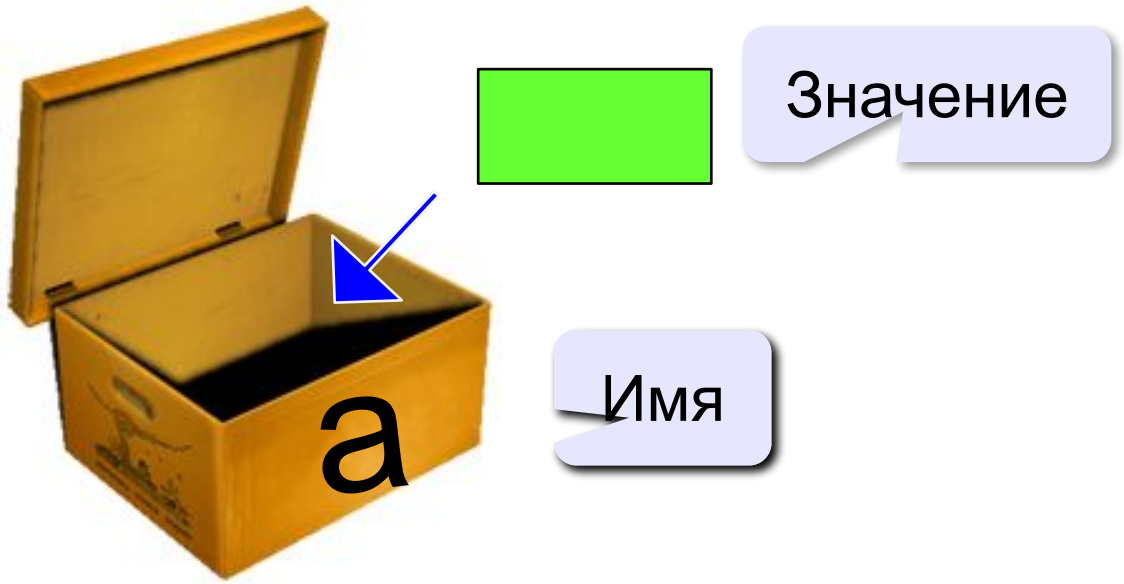
```
ввести два числа
вычислить их сумму
вывести сумму на экран
```

**Псевдокод** – алгоритм на русском языке с элементами языка программирования.

Компьютер не может исполнить псевдокод!

# Переменные

**Переменная** – это величина, имеющая имя, тип и значение. Значение переменной можно изменять во время работы программы.



# Имена переменных

**МОЖНО** использовать

- латинские буквы (A-Z, a-z)

заглавные и строчные буквы **различаются**

- русские буквы (не рекомендуется!)
- цифры

имя не может начинаться с цифры

- знак подчеркивания _

**НЕЛЬЗЯ** использовать

- ~~скобки~~
- ~~знаки +, =, !, ? и др.~~

Какие имена правильные?

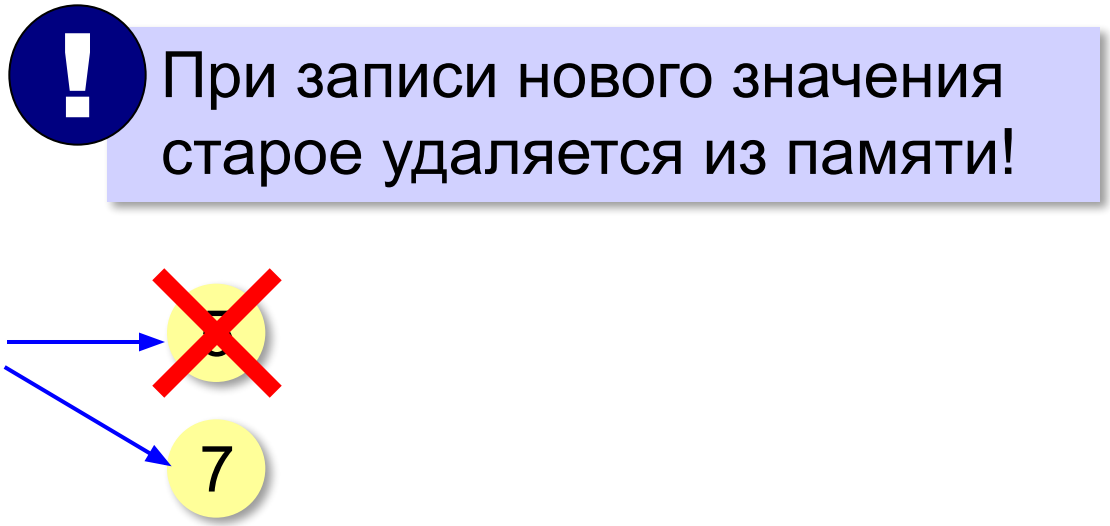
**AXby R&B 4Wheel Вася “PesBarbos”**
**TU154 [QuQu] _ABBA A+B**

# Как записать значение в переменную?

оператор присваивания

```
a = 5
```

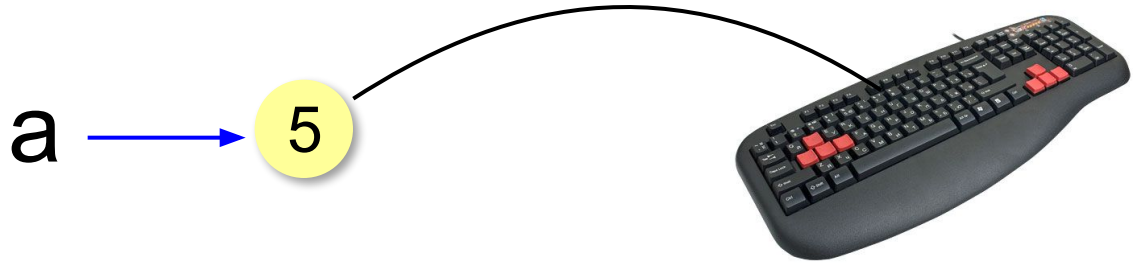
```
a = 7
```




**Оператор** – это команда языка программирования (инструкция).

**Оператор присваивания** – это команда для записи нового значения переменной.

# Ввод значения с клавиатуры




1. Программа ждет, пока пользователь введет значение и нажмет *Enter*.
2. Введенное значение записывается в переменную **a** (связывается с именем **a**)

# Ввод значения с клавиатуры

```
a = input()
b = input()
c = a + b
print ( c )
```

ввести строку с клавиатуры и связать с переменной **a**

**Протокол:**

```
21
33
2133
```

? Почему?

! Результат функции **input** – строка символов!

преобразовать в целое число

```
a = int( input() )
b = int( input() )
```

# Ввод двух значений в одной строке

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

21 33 — `input()` — ввести строку с клавиатуры

21 33 — `input().split()` — разделить строку на части по пробелам

целые — применить

21 33 — `map ( int, input().split() )`

эту операцию — к каждой части

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

# Ввод с подсказкой

```
a = input ( "Введите число: " )
```

```
Введите число: 26
```

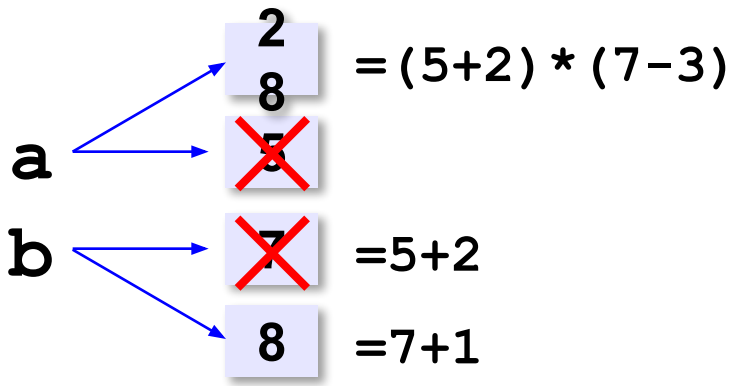
подсказка

Что не так?

```
a = int( input("Введите число: ") )
```

# Изменение значений переменной

```
a = 5
b = a + 2
a = (a + 2)*(b - 3)
b = b + 1
```



# Вывод данных

```
print ( a )
```

значение переменной

```
print ( "Ответ: ", a )
```

значение и текст

перечисление через запятую

```
print ( "Ответ: ", a+b )
```

вычисление выражения

```
print ( a, "+", b, "=", c )
```

```
2 + 3 = 5
```

через пробелы

```
print ( a, "+", b, "=", c, sep = "" )
```

```
2+3=5
```

убрать разделители

# Сложение чисел: простое решение

```
a = int( input() )
b = int( input() )
c = a + b
print( c )
```

Что плохо?

# Сложение чисел: полное решение

```
print( "Введите два числа: " )
a=int( input() )
b=int( input() )
c=a+b
print( a, "+", b, "=", c )
```

подсказка

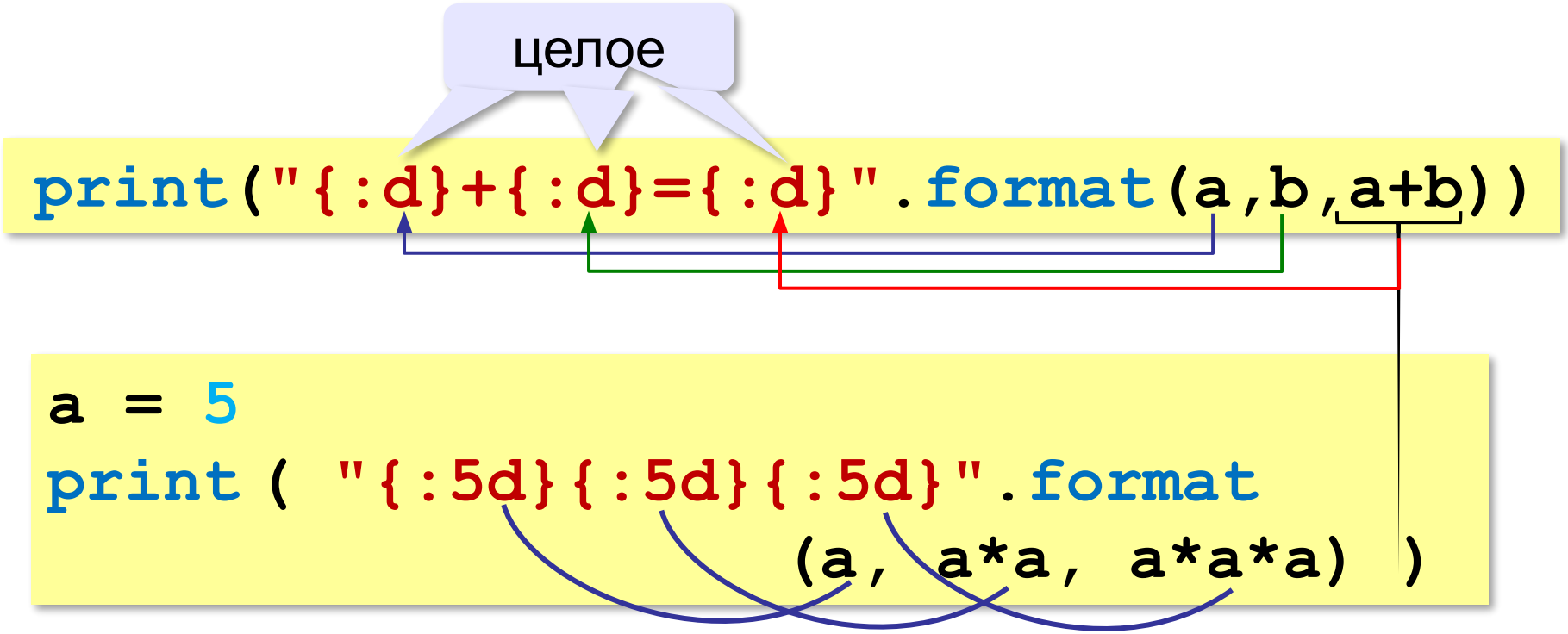
компьютер

**Протокол:**

```
Введите два целых числа
25
30
25 + 30 = 55
```

пользователь

# Форматный вывод



# Типы переменных

- `int` # целое
- `float` # вещественное
- `bool` # логические значения
- `str` # символьная строка

```
a = 5
print( type(a) )
a = 4.5
print( type(a) )
a = True
print( type(a) )
a = "Вася"
print( type(a) )
```

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

# Зачем нужен тип переменной?

Тип определяет:

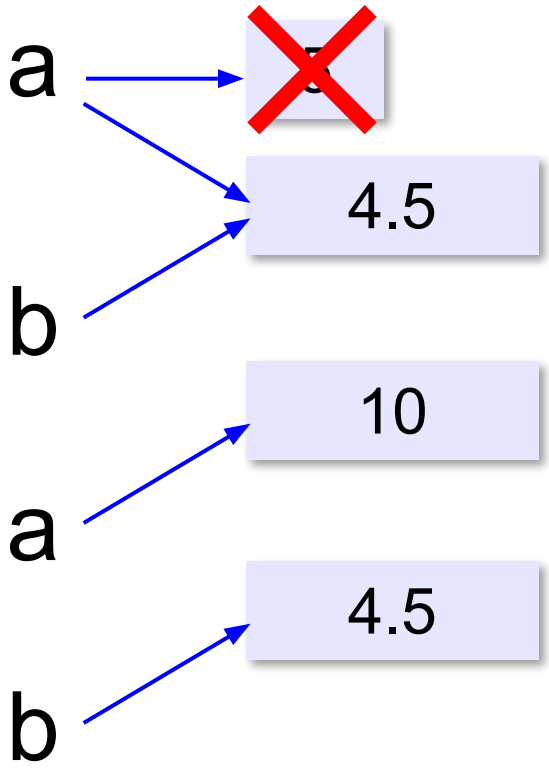
- область допустимых значений
- допустимые операции
- объём памяти
- формат хранения данных

# Размещение переменных в памяти

оператор присваивания

При записи нового значения старое удаляется из памяти!

```
a = 5
a = 4.5
b = a
a = 10
```




«сборщик мусора»

# Арифметическое выражения

```
          3    1  2   4       5   6
a = (c + b**5*3 - 1) / 2 * d
```

**Приоритет** (*старшинство*):

1) скобки
2) возведение в степень **
3) умножение и деление
4) сложение и вычитание

$$a = \frac{c + b^5 \cdot 3 - 1}{2} \cdot d$$

```
a = (c + b*5*3 - 1) \
              / 2 * d
```

перенос на следующую строку

```
a = (c + b*5*3
              - 1) / 2 * d
```

перенос внутри скобок разрешён

# Деление

Классическое деление:

```
a = 9; b = 6
x = 3 / 4     # = 0.75
x = a / b     # = 1.5
x = -3 / 4    # = -0.75
x = -a / b    # = -1.5
```

Целочисленное деление (округление «вниз»!):

```
a = 9; b = 6
x = 3 // 4     # = 0
x = a // b     # = 1
x = -3 // 4    # = -1
x = -a // b    # = -2
```

# Сокращенная запись операций

```
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 = a % b
```

```
a += 1
```

увеличение на 1

**Множественное присваивание**:

```
a = b = 0      # b = 0, a = b
a, b = 1, 2    # a = 1; b = 2
```

# Программирование на языке Python

## § 55. Вычисления

# Остаток от деления

`%` – остаток от деления

```
d = 85
b = d // 10
a = d % 10
d = a % b
d = b % a
```

## Для отрицательных чисел:

```
a = -7
b = a // 2   # -4
d = a % 2    # 1
```

! Как в математике!

остаток ≥ 0

-7 = (-4)*2 + 1

# Вещественные числа

Целая и дробная части числа разделяются **точкой**!

## Форматы вывода:

```
x = 123.456
print( x )
print("{:10.2f}".format(x))
```

```
123.456
␣␣␣␣ 123.46
```

всего знаков

в дробной части

```
print("{:10.2g}".format(x))
```

```
␣␣␣1.2e+02
```

значащих цифр

$1{,}2 \cdot 10^2$

# Вещественные числа

## Экспоненциальный формат:

```
x = 1./30000
print("{:e}".format(x))
x = 12345678.
print("{:e}".format(x))
```

3,333333 · $10^{-5}$

```
3.333333e-05
```

1,234568 · $10^{7}$

```
1.234568e+07
```

```
x = 123.456
print("{:e}".format(x))
print("{:10.2e}".format(x))
```

```
1.234560e+02
␣␣1.23e+02
```

всего знаков

в дробной части

# Стандартные функции

**abs(x)** – модуль числа

**int(x)** – преобразование к целому числу

**round(x)** – округление

```
x = abs( -1.6 )      # 1.6
x = int( -1.6 )      # -1
x = round( -1.6 )    # -2
```

**bin(x)** – в двоичную систему

**oct(x)** – в восьмеричную систему

**hex(x)** – в шестнадцатеричную систему

```
x = bin( 29 )   # '0b11101'
x = oct( 29 )   # '0o35'
x = hex( 29 )   # '0x1d'
```

# Математические функции

```
import math
```

подключить математический модуль

```
math.pi        – число «пи»
math.sqrt(x)   – квадратный корень
math.sin(x)    – синус угла, заданного в радианах
math.cos(x)    – косинус угла, заданного в радианах
math.exp(x)    – экспонента e^x
math.ln(x)     – натуральный логарифм
math.floor(x)  – округление «вниз»
math.ceil(x)   – округление «вверх»
```

```
x = math.floor(1.6)  # 1
x = math.ceil(1.6)   # 2
```

```
x = math.floor(-1.6)  #-2
x = math.ceil(-1.6)   #-1
```

# Документирование программы

```
from math import sqrt
print("Введите a, b, c:")
a, b, c = map(float, input().split())
D = b*b - 4*a*c
if D < 0:
  print("Нет")
else:
  x1 = (-b + sqrt(D))/(2*a)
  x2 = (-b - sqrt(D))/(2*a)
  print("x1={:5.3f} x2={:5.3f}".format(
                                x1, x2))
```

? Что делает?

# Документирование программы

**Руководство пользователя**:

- назначение программы
- формат входных данных
- формат выходных данных
- примеры использования программы

**Назначение**:

программа для решения уравнения

$$ax^2 + bx + c = 0$$

**Формат входных данных**:

значения коэффициентов $a$, $b$ и $c$ вводятся с клавиатуры через пробел в одной строке

# Документирование программы

**Формат выходных данных**:

значения вещественных корней уравнения; если вещественных корней нет, выводится слово «нет»

**Примеры использования программы**:

1. Решение уравнения $x^2 - 5x + 1 = 0$

```
Введите a, b, c: 1 -5 1
x1=4.791 x2=0.209
```

2. Решение уравнения $x^2 + x + 6 = 0$

```
Введите a, b, c: 1 1 6
Нет.
```

# Случайные числа

## Случайно…

- встретить друга на улице
- разбить тарелку
- найти 10 рублей
- выиграть в лотерею

## Случайный выбор:

- жеребьевка на соревнованиях
- выигравшие номера в лотерее

## Как получить случайность?

# Случайные числа на компьютере

## Электронный генератор

- нужно специальное устройство
- нельзя воспроизвести результаты

**Псевдослучайные числа** – обладают свойствами случайных чисел, но каждое следующее число вычисляется по заданной формуле.

**Метод середины квадрата** (Дж. фон Нейман)

зерно **564321**

в квадрате

**318458191041**

**209938992481**

- малый период (последовательность повторяется через $10^6$ чисел)

# Линейный конгруэнтный генератор

```
X = (a*X+b) % c | интервал от 0 до c-1
```

```
X = (X+3) % 10 | интервал от 0 до 9
```

X = 0 → 3 → 6 → 9 → 2 → 5 → 8

8 → 1 → 4 → 7 → 0

зерно

зацикливание

**!** Важен правильный выбор параметров $a$, $b$ и $c$!

**Компилятор GCC**:

a = 1103515245

b = 12345

c = $2^{31}$

# Генератор случайных чисел

```
import random
```

англ. *random* – *случайный*

## Целые числа на отрезке [a,b]:

```
X = random.randint(1,6) # псевдосл. число
Y = random.randint(1,6) # уже другое!
```

## Генератор на [0,1):

```
X = random.random()  # псевдослучайное число
Y = random.random()  # это уже другое число!
```

## Генератор на [a, b] (вещественные числа):

```
X = random.uniform(1.2, 3.5)
Y = random.uniform(1.2, 3.5)
```

# Генератор случайных чисел

```
from random import *
```

подключить все!

## Целые числа на отрезке [a,b]:

```
X = randint(10,60) # псевдослучайное число
Y = randint(10,60) # это уже другое число!
```

## Генератор на [0,1):

```
X = random(); # псевдослучайное число
Y = random()  # это уже другое число!
```

## Генератор на [a, b] (вещественные числа):

```
X = uniform(1.2, 3.5) # псевдосл. число
Y = uniform(1.2, 3.5) # уже другое число!
```

# Задачи

**«A»:** Ввести с клавиатуры три целых числа, найти их сумму, произведение и среднее арифметическое.

**Пример**:

```
Введите три целых числа:
5 7 8
5+7+8=20
5*7*8=280
(5+7+8)/3=6.667
```

**«B»:** Ввести с клавиатуры координаты двух точек (A и B) на плоскости (вещественные числа). Вычислить длину отрезка AB.

**Пример**:

```
Введите координаты точки A:
5.5 3.5
Введите координаты точки B:
1.5 2
Длина отрезка AB = 4.272
```

# Задачи

**«С»:** Получить случайное трехзначное число и вывести через запятую его отдельные цифры.

**Пример**:

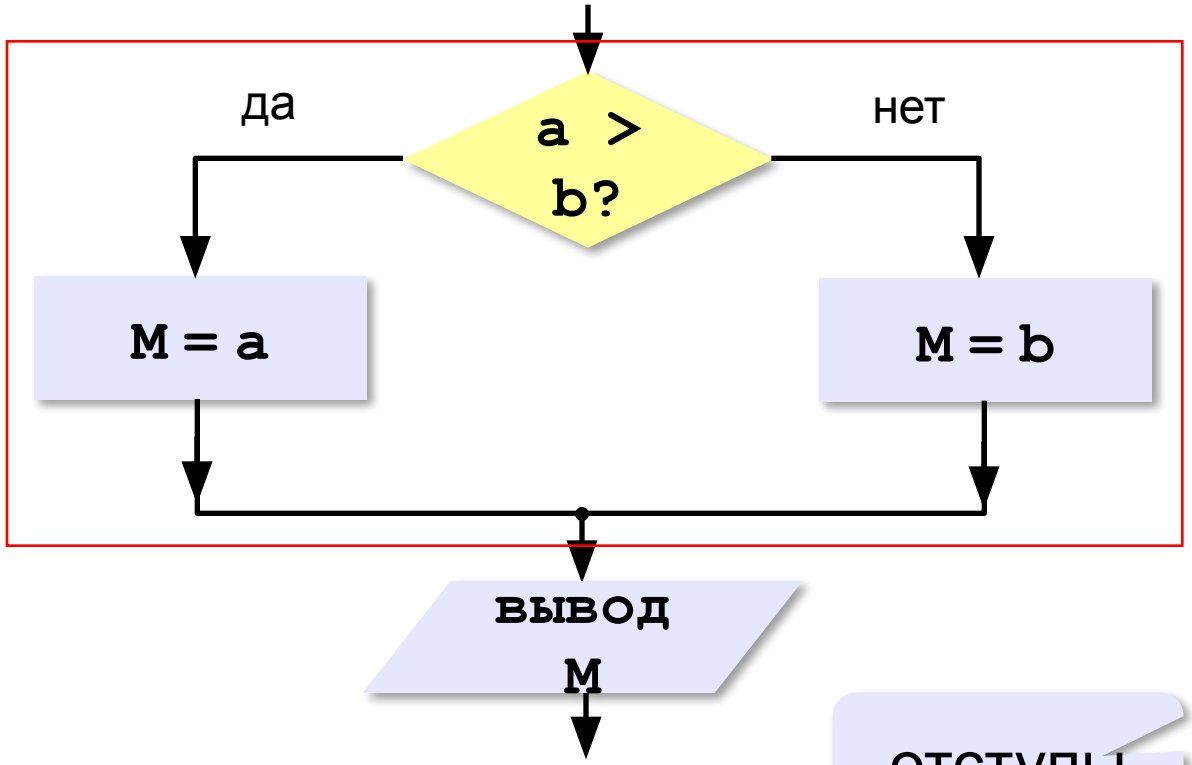
```
Получено число 123.
Его цифры 1, 2, 3.
```

# Программирование на языке Python

## § 56. Ветвления

# Условный оператор

Задача: **изменить порядок действий** в зависимости от выполнения некоторого условия.



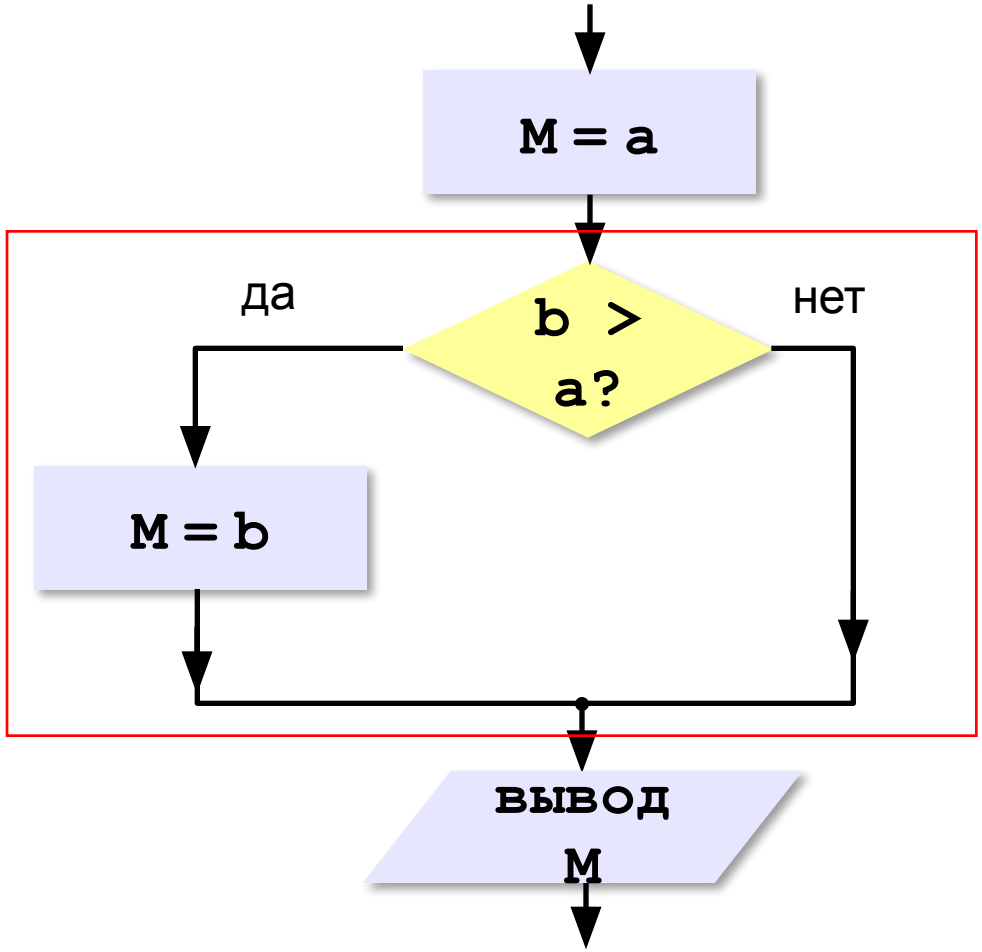

полная форма ветвления

? Если a = b?

```
if a > b:
    M = a
else:
    M = b
```

отступы

# Условный оператор: неполная форма




```
M = a
if b > a:
   M = b
```

неполная форма ветвления

**Решение в стиле Python:**

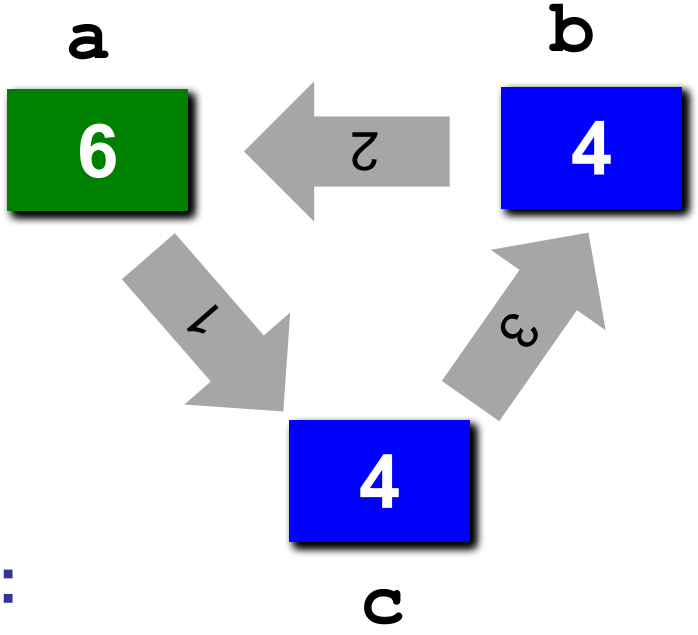
```
M = max(a, b)
```

```
M = a if a > b else b
```

# Условный оператор

```
if a < b:
   c = a
   a = b
   b = c
```

? Что делает?




? Можно ли обойтись без переменной **c**?

**Решение в стиле Python:**

```
a, b = b, a
```

# Знаки отношений

| | |
|---|---|
| `>` `<` | больше, меньше |
| `>=` | больше или равно |
| `<=` | меньше или равно |
| `==` | равно |
| `!=` | не равно |

# Вложенные условные операторы

*Задача*: в переменных **a** и **b** записаны возрасты Андрея и Бориса. Кто из них старше?

Сколько вариантов?

```
if a > b:
   print("Андрей старше")
else:
   if a == b:
      print("Одного возраста")
   else:
      print("Борис старше")
```

Зачем нужен?

вложенный условный оператор

# Каскадное ветвление

```
if a > b:
  print("Андрей старше")
elif a == b:
  print("Одного возраста")
else:
  print("Борис старше")
```

`elif = else if`

# Каскадное ветвление

```
cost = 1500
if cost<1000:
  print( "Скидок нет." )
elif cost<2000:
  print( "Скидка 2%." )
elif cost<5000:
  print( "Скидка 5%." )
else:
  print( "Скидка 10%." )
```

первое сработавшее условие

? Что выведет?

```
Скидка 2%.
```

# Задачи

**«A»:** Ввести три целых числа, найти максимальное из них.

**Пример:**

```
Введите три целых числа:
1 5 4
Максимальное число 5
```

**«B»:** Ввести пять целых чисел, найти максимальное из них.

**Пример:**

```
Введите пять целых чисел:
1 5 4 3 2
Максимальное число 5
```

# Задачи

**«С»:** Ввести последовательно возраст Антона, Бориса и Виктора. Определить, кто из них старше.

**Пример**:

```
Возраст Антона: 15
Возраст Бориса: 17
Возраст Виктора: 16
Ответ: Борис старше всех.
```

**Пример**:

```
Возраст Антона: 17
Возраст Бориса: 17
Возраст Виктора: 16
Ответ: Антон и Борис старше Виктора.
```

# Сложные условия

*Задача*: набор сотрудников в возрасте **25-40 лет** (включительно).

сложное условие

```
if v >= 25 and v <= 40 :
   print("подходит")
else:
   print("не подходит")
```

**and** «И»

**or** «ИЛИ»

**not** «НЕ»

**Приоритет :**

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

# Задачи

**«A»:** Напишите программу, которая получает три числа и выводит количество одинаковых чисел в этой цепочке.

**Пример**:

```
Введите три числа:
5 5 5
Все числа одинаковые.
```

**Пример**:

```
Введите три числа:
5 7 5
Два числа одинаковые.
```

**Пример**:

```
Введите три числа:
5 7 8
Нет одинаковых чисел.
```

# Задачи

**«B»:** Напишите программу, которая получает номер месяца и выводит соответствующее ему время года или сообщение об ошибке.

**Пример**:

```
Введите номер месяца:
5
Весна.
```

**Пример**:

```
Введите номер месяца:
15
Неверный номер месяца.
```

# Задачи

**«С»:** Напишите программу, которая получает возраст человека (целое число, не превышающее 120) и выводит этот возраст со словом «год», «года» или «лет». Например, «21 год», «22 года», «25 лет».

**Пример**:

```
Введите возраст: 18
Вам 18 лет.
```

**Пример**:

```
Введите возраст: 21
Вам 21 год.
```

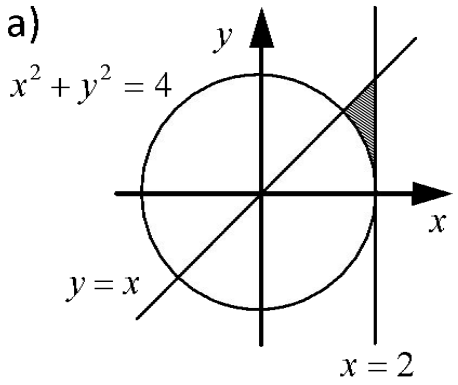
**Пример**:

```
Введите возраст: 22
Вам 22 года.
```

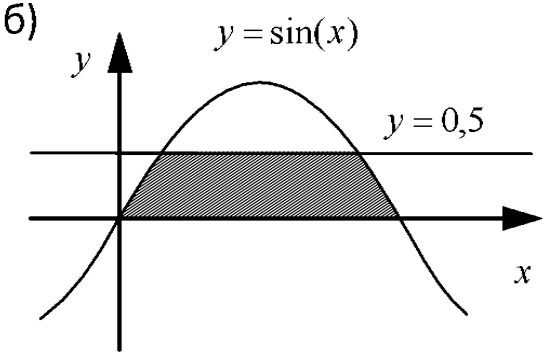
# Задачи

**«A»:** Напишите условие, которое определяет заштрихованную область.

а)

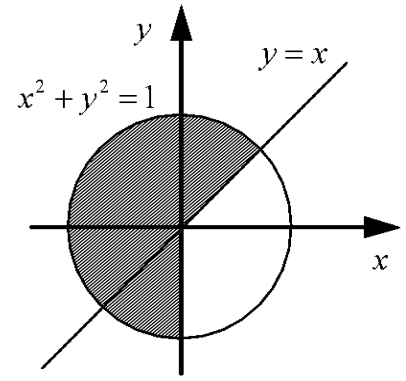



б)

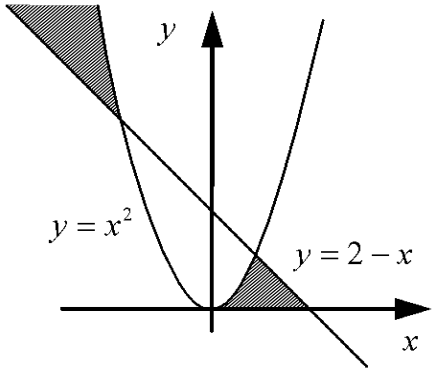



**«B»:** Напишите условие, которое определяет заштрихованную область.

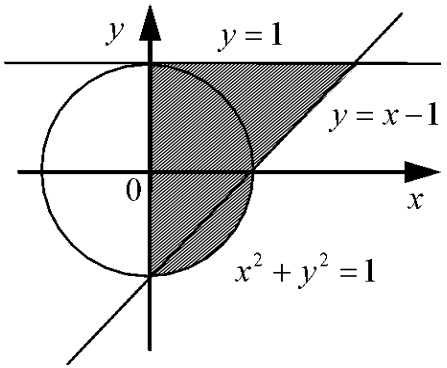
а)




б)

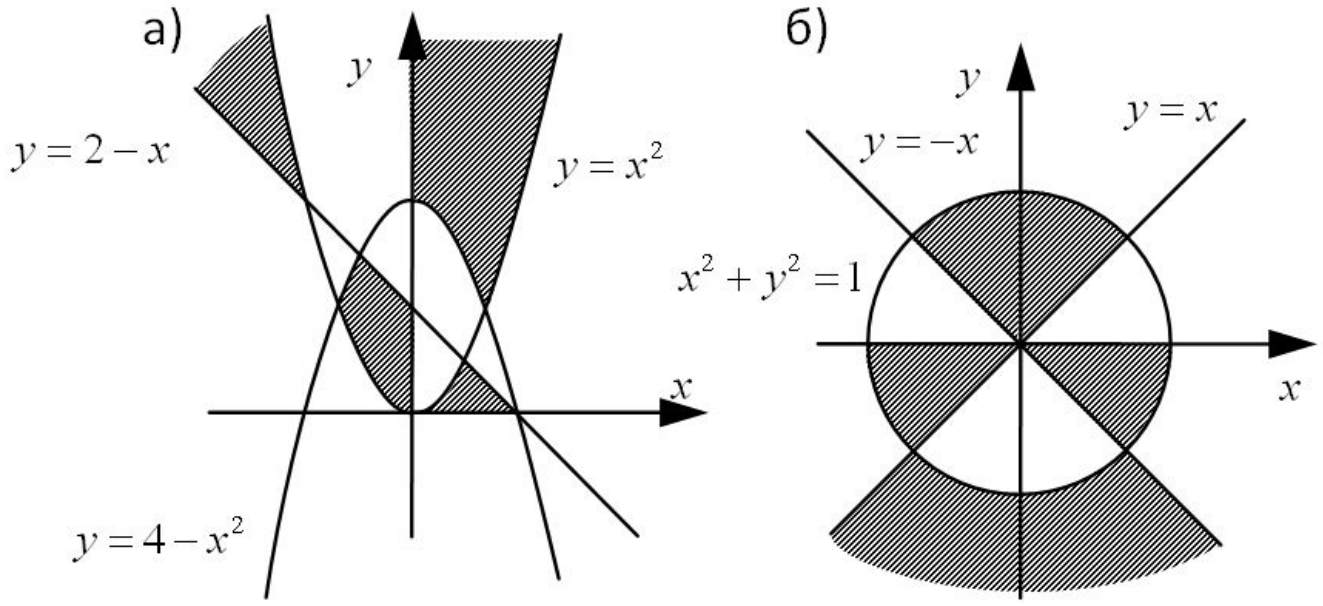



в)



# Задачи

**«C»:** Напишите условие, которое определяет заштрихованную область.



# Программирование на языке Python

## § 57. Циклические алгоритмы

# Что такое цикл?

**Цикл** – это многократное выполнение одинаковых действий.

**Два вида циклов**:

- цикл с **известным** числом шагов (сделать 10 раз)
- цикл с **неизвестным** числом шагов (делать, пока не надоест)

*Задача*. Вывести на экран 10 раз слово «Привет».

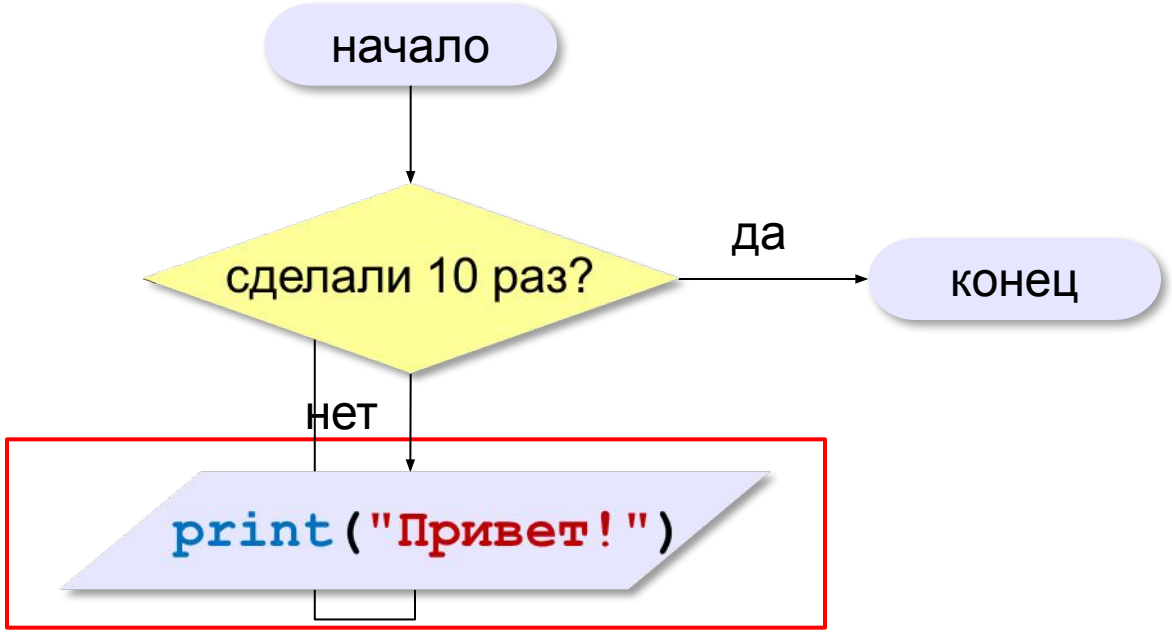
Можно ли решить известными методами?

# Повторения в программе

```
print("Привет“)
print("Привет")
...
print("Привет")
```

Что плохо?

# Блок-схема цикла




тело цикла

# Как организовать цикл?

```
счётчик = 0
пока счётчик < 10:
   print("Привет")
   увеличить счётчик на 1
```

результат операции автоматически сравнивается с нулём!

```
счётчик = 10
пока счётчик > 0:
   print("Привет")
   уменьшить счётчик на 1
```

? Какой способ удобнее для процессора?

# Цикл с условием

*Задача*. Определить **количество цифр** в десятичной записи целого положительного числа, записанного в переменную **n**.

```
счётчик = 0
пока n > 0:
   отсечь последнюю цифру n
   увеличить счётчик на 1
```

| n | счётчик |
|---|---|
| 1234 | 0 |
| | |

? Как отсечь последнюю цифру?

```
n = n // 10
```

? Как увеличить счётчик на 1?

```
счётчик = счётчик + 1
```

```
счётчик += 1
```

# Считаем цифры

начальное значение счётчика

условие продолжения

заголовок цикла

```
count = 0
while n > 0 :
    n = n // 10
    count += 1
```

тело цикла

! Цикл с предусловием – проверка на входе в цикл!

# Максимальная цифра числа

*Задача*. Определить **максимальную цифру** в десятичной записи целого положительного числа, записанного в переменную **n**.

```
n = int(input())
M = -1
while n > 0:
    d = n % 10
    if d > M:
        M = d
    n = n // 10
print( M )
```

последняя цифра

пока остались цифры

поиск максимума

? Что плохо!

отсечь последнюю цифру

# Цикл с условием

## При известном количестве шагов:

```
k = 0
while k < 10:
   print ( "привет" )
   k += 1
```

## Зацикливание:

```
k = 0
while  k < 10:
   print ( "привет" )
```

# Сколько раз выполняется цикл?

```
a = 4; b = 6
while a < b: a += 1
```

2 раза
a = 6

```
a = 4; b = 6
while a < b: a += b
```

1 раз
a = 10

```
a = 4; b = 6
while a > b: a += 1
```

0 раз
a = 4

```
a = 4; b = 6
while a < b: b = a - b
```

1 раз
b = -2

```
a = 4; b = 6
while a < b: a -= 1
```

зацикливание

# Задачи

**«A»:** Напишите программу, которая получает два целых числа A и B (0 < A < B) и выводит квадраты всех натуральных чисел в интервале от A до B.

**Пример**:

```
Введите два целых числа:
10 12
10*10=100
11*11=121
12*12=144
```

**«B»:** Напишите программу, которая получает два целых числа и находит их произведение, не используя операцию умножения. Учтите, что числа могут быть отрицательными.

**Пример**:

```
Введите два числа:
10 -15
10*(-15)=-150
```

# Задачи

**«C»:** Ввести натуральное число N и вычислить сумму всех чисел Фибоначчи, меньших N. Предусмотрите защиту от ввода отрицательного числа N.

**Пример:**

```
Введите число N:
10000
Сумма 17709
```

# Задачи-2

**«A»:** Ввести натуральное число и найти сумму его цифр.

**Пример**:

```
Введите натуральное число:
12345
Сумма цифр 15.
```

**«B»:** Ввести натуральное число и определить, верно ли, что в его записи есть две одинаковые цифры, стоящие рядом.

**Пример**:

```
Введите натуральное число:
12342
Нет.
```

**Пример**:

```
Введите натуральное число:
12245
Да.
```

# Задачи-2

**«С»:** Ввести натуральное число и определить, верно ли, что в его записи есть две одинаковые цифры (не обязательно стоящие рядом).

**Пример:**

```
Введите натуральное число:
12342
Да.
```

**Пример:**

```
Введите натуральное число:
12345
Нет.
```

# Алгоритм Евклида

**Алгоритм Евклида**. Чтобы найти НОД двух натуральных чисел, нужно вычитать из большего числа меньшее до тех пор, пока они не станут равны. Это число и есть НОД исходных чисел.

**НОД(14,21) = НОД(14,7) = НОД(7, 7) = 7**

```
пока a != b:
  если a > b:
    a -= b  # a = a - b
  иначе:
    b -= a  # b = b - a
```

```
while a != b:
  if a > b:
    a -= b
  else:
    b -= a
```

**НОД(1998,2) = НОД(1996,2) = … = НОД(2, 2) = 2**

# Алгоритм Евклида

**Модифицированный алгоритм Евклида**. Заменять большее число на остаток от деления большего на меньшее до тех пор, пока меньшее не станет равно нулю. Другое (ненулевое) число и есть НОД чисел.

**НОД(1998,2) = НОД(0,2) = 2**

```
пока a!=0 and b!=0:
  если a > b:
    a = a % b
  иначе:
    b = b % a
```

? Какое условие?

```
если a != 0:
  вывести a
иначе:
  вывести b
```

? Как вывести результат?

# Задачи

**«3»:** Ввести с клавиатуры два натуральных числа и найти их НОД с помощью алгоритма Евклида.

**Пример:**

```
Введите два числа:
21 14
НОД(21,14)=7
```

**«4»:** Ввести с клавиатуры два натуральных числа и найти их НОД с помощью **модифицированного** алгоритма Евклида. Заполните таблицу:

| a | 64168 | 358853 | 6365133 | 17905514 | 549868978 |
|---|---|---|---|---|---|
| b | 82678 | 691042 | 11494962 | 23108855 | 298294835 |
| НОД(a,b) | | | | | |

# Задачи

**«5»:** Ввести с клавиатуры два натуральных числа и сравнить количество шагов цикла для вычисления их НОД с помощью обычного и модифицированного алгоритмов Евклида.

**Пример**:

```
Введите два числа:
1998 2
НОД(1998,2)=2
Обычный алгоритм: 998
Модифицированный: 1
```

# Цикл с постусловием

*Задача*. Обеспечить ввод **положительного** числа в переменную **n**.

бесконечный цикл

```
while True:
    print ( "Введите положительное число:" )
    n = int ( input() )
    if n > 0: break
```

тело цикла

условие выхода

прервать цикл

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

# Программирование на языке Python

## § 58. Циклы по переменной

# Цикл с переменной

*Задача*. Вывести 10 раз слово «Привет!».

? Можно ли сделать с циклом «пока»?

```
i = 0
while i < 10:
    print("Привет!")
    i += 1
```

**Цикл с переменной:**

```
for i in range(10):
    print("Привет!")
```

в диапазоне [0,**10**)

! Не включая **10**!

`range(10)` → `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`

# Цикл с переменной

*Задача*. Вывести все степени двойки от $2^1$ до $2^{10}$.

? Как сделать с циклом «пока»?

```
k = 1
while k <= 10 :
    print ( 2**k )
    k += 1
```

**Цикл с переменной:**

```
for k in range(1,11) :
    print ( 2**k )
```

в диапазоне [1,**11**)

! Не включая **11**!

```
range(1,11) → 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

# Цикл с переменной: другой шаг

10,9,8,7,6,5,4,3,2,1

шаг

```
for k in range(10,0,-1):
    print ( k**2 )
```

```
100
81
64
49
36
25
16
9
4
1
```

? Что получится?

1,3,5,7,9

```
for k in range(1,11,2):
    print ( k**2 )
```

```
1
9
25
49
81
```

# Сколько раз выполняется цикл?

```
a = 1
for i in range( 3): a += 1
```

a = 4

```
a = 1
for i in range( 3,1): a += 1
```

a = 1

```
a = 1
for i in range( 1,3,-1): a += 1
```

a = 1

```
a = 1
for i in range( 3,1,-1): a += 1
```

a = 3

# Задачи

**«A»:** Найдите все пятизначные числа, которые при делении на 133 дают в остатке 125, а при делении на 134 дают в остатке 111.

**«B»:** Натуральное число называется **числом Армстронга**, если сумма цифр числа, возведенных в N-ную степень (где N – количество цифр в числе) равна самому числу. Например, $153 = 1^3 + 5^3 + 3^3$. Найдите все трёхзначные Армстронга.

# Задачи

**«C»:** Натуральное число называется автоморфным, если оно равно последним цифрам своего квадрата. Например, $25^2 = 625$. Напишите программу, которая получает натуральное число N и выводит на экран все автоморфные числа, не превосходящие N.

**Пример:**

```
Введите N:
1000
1*1=1
5*5=25
6*6=36
25*25=625
76*76=5776
```

# Вложенные циклы

*Задача*. Вывести все простые числа в диапазоне от 2 до 1000.

```
сделать для n от 2 до 1000
  если число n простое то
    вывод n
```

нет делителей [2.. n-1]: проверка в цикле!

? Что значит «простое число»?

```
for n in range(2, 1001):
  if число n простое:
    print( n )
```

# Вложенные циклы

```
for n in range(2, 1001):
    count = 0
    for k in range(2,n):
        if n % k == 0:
            count += 1
    if count == 0:
        print( n )
```

вложенный цикл

# Вложенные циклы

```
for i in range(1,4):
  for k in range(1,4):
    print( i, k )
```

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

? Как меняются переменные?

! Переменная внутреннего цикла изменяется быстрее!

# Вложенные циклы

```
for i in range(1,5):
  for k in range(1,i+1):
    print( i, k )
```

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

? Как меняются переменные?

! Переменная внутреннего цикла изменяется быстрее!

# Поиск простых чисел – как улучшить?

$$n = k \cdot m, \quad k \le m \quad \Rightarrow k^2 \le n \quad \Rightarrow k \le \sqrt{n}$$

```
while k <= math.sqrt(n):
    ...
```

? Что плохо?

```
count = 0
k = 2
while k*k <= n :
    if n % k == 0:
        count += 1
    k += 1
```

? Как ещё улучшить?

выйти из цикла

```
while k*k <= n:
    if n % k == 0: break
    k += 1
if k*k > n:
    print ( n )
```

если вышли по условию

# Задачи

**«A»:** Напишите программу, которая получает натуральные числа A и B (A<B) и выводит все простые числа в интервале от A до B.

**Пример:**

```
Введите границы диапазона:
10 20
11 13 17 19
```

**«B»:** В магазине продается мастика в ящиках по 15 кг, 17 кг, 21 кг. Как купить ровно 185 кг мастики, не вскрывая ящики? Сколькими способами можно это сделать?

# Задачи

**«С»:** Ввести натуральное число N и вывести все натуральные числа, не превосходящие N и делящиеся на каждую из своих цифр.

**Пример:**

```
Введите N:
15
1 2 3 4 5 6 7 8 9 11 12 15
```

# Программирование на языке Python

## § 59. Процедуры

# Зачем нужны процедуры?

```
print ( "Ошибка программы" )
```

много раз!

## Процедура:

*define*
определить

```
def Error():
  print( "Ошибка программы" )
```

```
n = int ( input() )
if n < 0:
  Error()
```

вызов процедуры

# Что такое процедура?

**Процедура** – вспомогательный алгоритм, который выполняет некоторые действия.

- текст (расшифровка) процедуры записывается **до** её вызова в основной программе
- в программе может быть **много процедур**
- чтобы процедура заработала, нужно **вызвать** её по имени из основной программы или из другой процедуры

# Процедура с параметрами

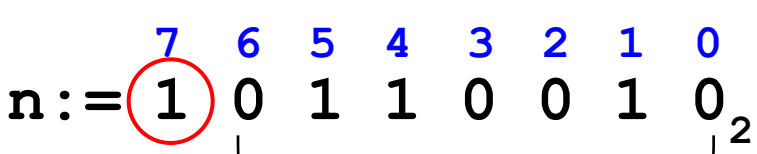
*Задача*. Вывести на экран запись целого числа (0..255) в 8-битном двоичном коде.

много раз!

**Алгоритм:**

$178 \Rightarrow 10110010_2$

? Как вывести первую цифру?




`n // 128`

`n % 128`

? Как вывести вторую цифру?

`n1 // 64`

# Процедура с параметрами

*Задача*. Вывести на экран запись целого числа (0..255) в 8-битном двоичном коде.

**Решение:**

```
k = 128
while k > 0:
    print( n // k,
           end = "" )
    n = n % k
    k = k // 2
```

| n | k | вывод |
|---|---|---|
| 178 | 128 | 1 |
| | | |
| | | |
| | | |
| | | |

178 ⇒ 10110010

Результат зависит от **n**!

# Процедура с параметрами

**Параметры** – данные, изменяющие работу процедуры.

```
def printBin( n ):
  k = 128
  while k > 0:
    print ( n // k, end = "" )
    n = n % k;
    k = k // 2
```

локальная переменная

```
printBin ( 99 )
```

значение параметра (**аргумент**)

**Несколько параметров:**

```
def printSred( a, b ):
  print ( (a + b)/2 )
```

# Локальные и глобальные переменные

глобальная переменная

локальная переменная

```
a = 5
def qq():
    a = 1
    print ( a )
qq()
print ( a )
```

1

5

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

5

```
a = 5
def qq():
    global a
    a = 1
qq()
print ( a )
```

1

работаем с глобальной переменной

# Неправильная процедура

```
x = 5; y = 10


xSum()
```

```
def xSum():
    print ( x+y )
```

Что плохо?

1) процедура связана с глобальными переменными, нельзя перенести в другую программу
2) печатает только сумму **x** и **y**, нельзя напечатать сумму других переменных или сумму **x*y** и **3x**

Как исправить?

передавать данные через параметры

# Правильная процедура

Глобальные:

| x | y |
|---|---|
| 5 | 10 |

| z | w |
|---|---|
| 17 | 3 |

```
x = 5; y = 10
Sum2( x, y )
z=17; w=3
Sum2( z, w )
Sum2( z+x, y*w )
```

```
def Sum2(a, b):
  print ( a+b )
```

Локальные:

| a | b |
|---|---|
| 5<br>17 | 10<br>3 |

```
15
20
52
```

1) процедура не зависит от глобальных переменных
2) легко перенести в другую программу
3) печатает только сумму любых выражений

# Задачи

**«A»:** Напишите процедуру, которая принимает параметр – натуральное число N – и выводит на экран линию из N символов '–'.

**Пример**:

```
Введите N:
10
----------
```

**«B»:** Напишите процедуру, которая выводит на экран в столбик все цифры переданного ей числа, начиная с первой.

**Пример**:

```
Введите натуральное число:
1234
1
2
3
4
```

# Задачи

**«С»:** Напишите процедуру, которая выводит на экран запись переданного ей числа в римской системе счисления.

**Пример:**

```
Введите натуральное число:
2013
MMXIII
```

# Программирование на языке Python

## § 60. Функции

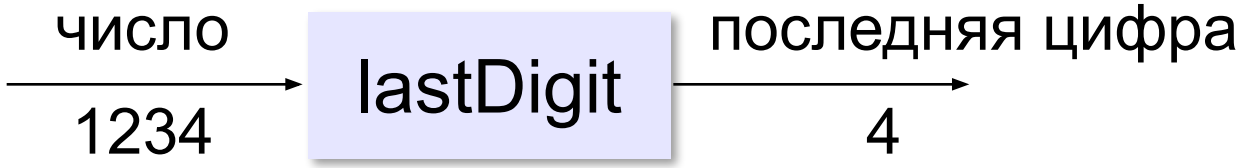
# Что такое функция?

**Функция** – это вспомогательный алгоритм, который возвращает *значение-результат* (число, символ или объект другого типа).

```
s = input()
n = int( s )
x = randint( 10, 20 )
```

# Что такое функция?

*Задача*. Написать функцию, которая вычисляет младшую цифру числа (разряд единиц).




```
def lastDigit( n ):
    d = n % 10
    return d
```

результат работы функции – значение **d**

передача результата

```
# вызов функции
k = lastDigit( 1234 )
print( k )
```

# Сумма цифр числа

*Задача*. Написать функцию, которая вычисляет сумму цифр числа.

```
def sumDigits( n ):
    sum = 0
    while n != 0:
        sum += n % 10
        n = n // 10
    return sum
```

передача результата

```
# основная программа
print ( sumDigits(12345) )
```

# Использование функций

```
x = 2*sumDigits( n+5 )
z = sumDigits( k ) + sumDigits( m )
if sumDigits( n ) % 2 == 0:
   print ( "Сумма цифр чётная" )
   print ( "Она равна", sumDigits( n ) )
```

! Функция, возвращающая целое число, может использоваться везде, где и целая величина!

**Одна функция вызывает другую:**

```
def middle ( a, b, c ):
   mi = min ( a, b, c )
   ma = max ( a, b, c )
   return a + b + c - mi - ma
```

вызываются **min** и **max**

? Что вычисляет?

# Задачи

**«A»:** Напишите функцию, которая находит наибольший общий делитель двух натуральных чисел.

**Пример**:

```
Введите два натуральных числа:
7006652 112307574
НОД(7006652,112307574) = 1234.
```

**«B»:** Напишите функцию, которая определяет сумму цифр переданного ей числа.

**Пример**:

```
Введите натуральное число:
123
Сумма цифр числа 123 равна 6.
```

# Задачи

**«С»:** Напишите функцию, которая «переворачивает» число, то есть возвращает число, в котором цифры стоят в обратном порядке.

**Пример:**

```
Введите натуральное число:
1234
После переворота: 4321.
```

# Как вернуть несколько значений?

```
def divmod ( x, y ):
    d = x // y
    m = x % y
    return d, m
```

d – частное,
m – остаток

```
a, b = divmod ( 7, 3 )
print ( a, b )      # 2 1
```

```
q = divmod ( 7, 3 )
print ( q )         # (2, 1 (2, 1)
```

кортеж – набор элементов

# Задачи

**«A»:** Напишите функцию, которая переставляет три переданные ей числа в порядке возрастания.

**Пример**:

```
Введите три натуральных числа:
10 15 5
5 10 15
```

**«B»:** Напишите функцию, которая сокращает дробь вида M/N.

**Пример**:

```
Введите числитель и знаменатель дроби:
25 15
После сокращения: 5/3
```

# Задачи

**«С»:** Напишите функцию, которая вычисляет наибольший общий делитель и наименьшее общее кратное двух натуральных чисел.

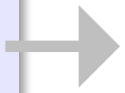
**Пример**:

```
Введите два натуральных числа:
10 15
НОД(10,15)=5
НОК(10,15)=30
```

# Логические функции

**Логическая функция** – это функция, возвращающая логическое значение (True/False).

```
def even(n):
    if n % 2 == 0:
        return True
    else:
        return False
```



```
def even(n):
    return (n % 2 == 0)
```

```
k = int( input() )
if even( k ):
    print( "Число", k, "чётное." )
else:
    print( "Число", k, "нечётное." )
```

# Логические функции

*Задача*. Найти все простые числа в диапазоне от 2 до 100.

```
for i in range(2,1001):
  if isPrime(i) :
    print( i )
```

функция, возвращающая логическое значение (True/False)

# Функция: простое число или нет?

? Какой алгоритм?

```
def isPrime ( n ):
    k = 2
    while k*k <= n and n % k != 0:
        k += 1
    return (k*k > n)
```

```
if k*k > n:
    return True
else:
    return False
```

# Логические функции: использование

Функция, возвращающая логическое значение, может использоваться везде, где и логическая величина!

```
n = int ( input() )
while isPrime(n):
  print ( n, "- простое число" )
  n = int ( input() )
```

# Задачи

**«A»:** Напишите логическую функцию, которая определяет, является ли переданное ей число совершенным, то есть, равно ли оно сумме своих делителей, меньших его самого.

**Пример**:

```
Введите натуральное число:
28
Число 28 совершенное.
```

**Пример**:

```
Введите натуральное число:
29
Число 29 не совершенное.
```

# Задачи

**«B»:** Напишите логическую функцию, которая определяет, являются ли два переданные ей числа взаимно простыми, то есть, не имеющими общих делителей, кроме 1.

**Пример**:

```
Введите два натуральных числа:
28 15
Числа 28 и 15 взаимно простые.
```

**Пример**:

```
Введите два натуральных числа:
28 16
Числа 28 и 16 не взаимно простые.
```

# Задачи

**«С»:** Простое число называется гиперпростым, если любое число, получающееся из него откидыванием нескольких цифр, тоже является простым. Например, число 733 – гиперпростое, так как и оно само, и числа 73 и 7 – простые. Напишите логическую функцию, которая определяет, верно ли, что переданное ей число – гиперпростое. Используйте уже готовую функцию **`isPrime`**, которая приведена в учебнике.

**Пример:**

```
Введите натуральное число:
733
Число 733 гиперпростое.
```

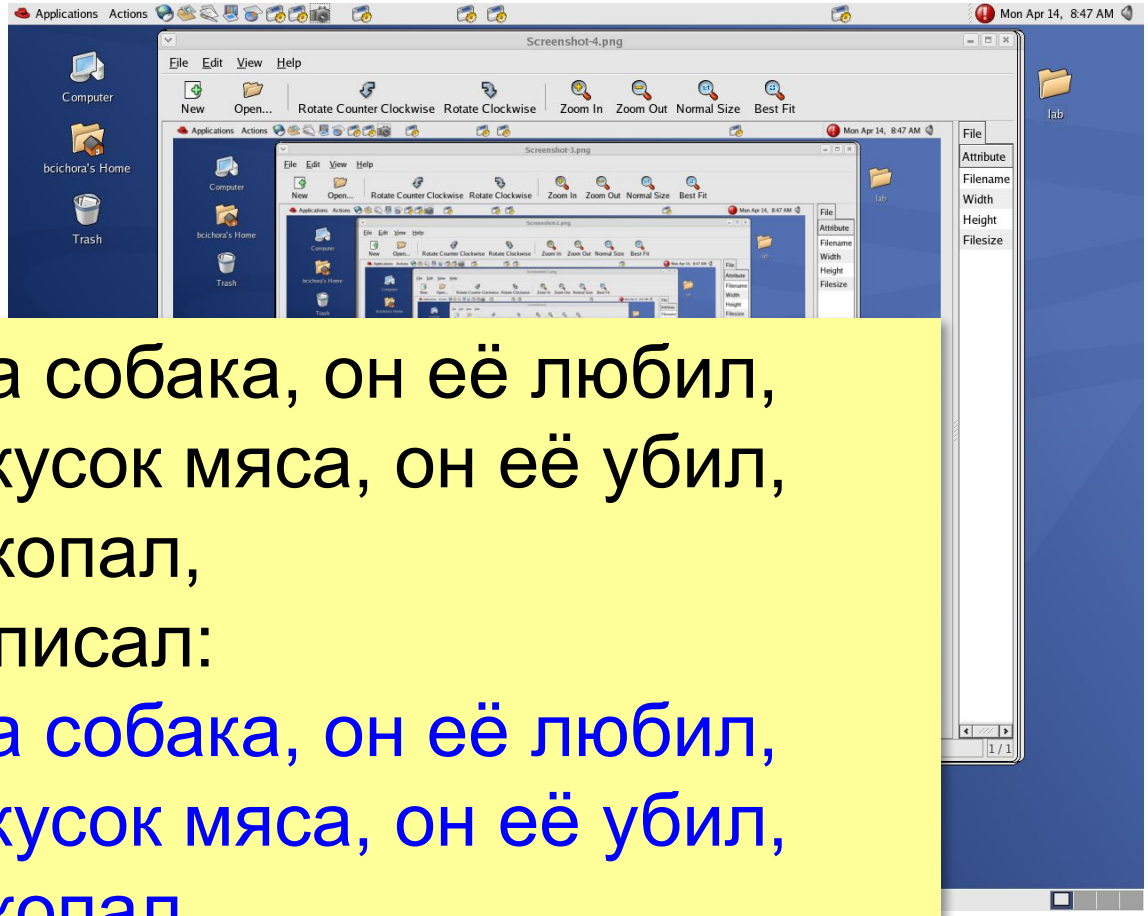
**Пример:**

```
Введите натуральное число:
19
Число 19 не гиперпростое.
```

# Программирование на языке Python

## § 61. Рекурсия

# Что такое рекурсия?




У попа была собака, он её любил,
Она съела кусок мяса, он её убил,
В землю закопал,
Надпись написал:
У попа была собака, он её любил,
Она съела кусок мяса, он её убил,
В землю закопал,
Надпись написал:
…

# Что такое рекурсия?

**Натуральные числа**:

- 1 – натуральное число
- если $n$ – натуральное число, то $n+1$ – натуральное число

индуктивное определение

**Рекурсия** — это способ определения множества объектов через само это множество на основе заданных простых базовых случаев.

**Числа Фибоначчи**:

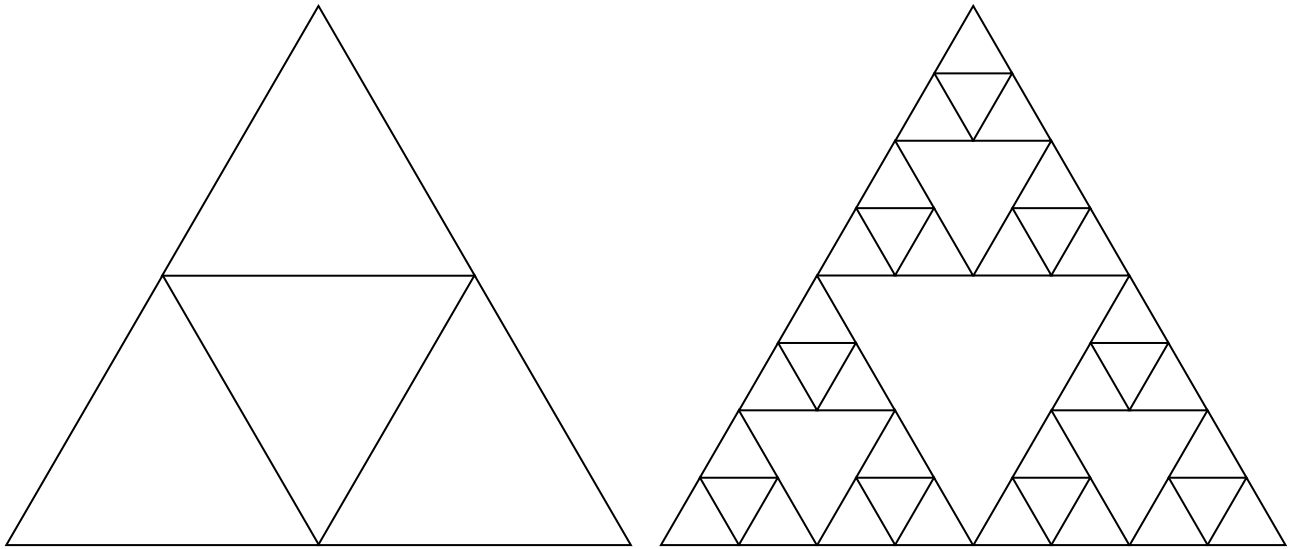
- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$ при $n > 2$

**1, 1, 2, 3, 5, 8, 13, 21, 34, …**

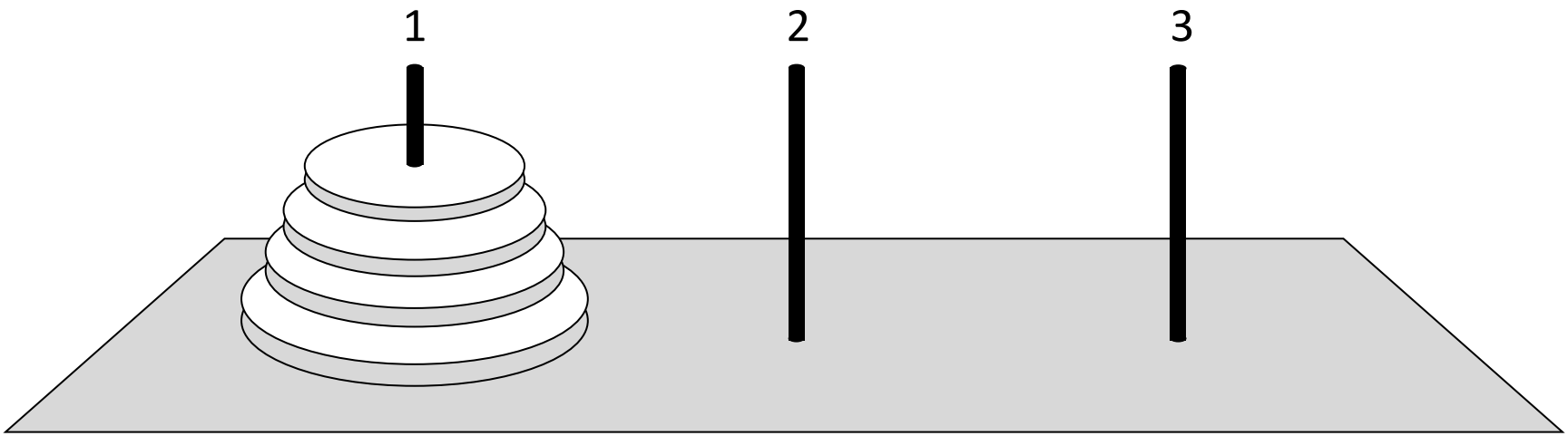
# Фракталы

**Фракталы** – геометрические фигуры, обладающие самоподобием.

## Треугольник Серпинского:



# Ханойские башни




- за один раз переносится один диск
- класть только меньший диск на больший
- третий стержень вспомогательный

```
перенести (n, 1, 3)
```

```
перенести (n-1, 1, 2)
1 -> 3
перенести (n-1, 2, 3)
```

# Ханойские башни – процедура

сколько | откуда | куда

```
def Hanoi ( n, k, m ):
    p = 6 - k - m
    Hanoi ( n-1, k, p )
    print ( k, "->", m )
    Hanoi ( n-1, p, m )
```

номер вспомогательного стержня (1+2+3=6!)

рекурсия

рекурсия

? Что плохо?

! **Рекурсия никогда не остановится!**

# Ханойские башни – процедура

**Рекурсивная процедура (функция)** — это процедура (функция), которая вызывает сама себя напрямую или через другие процедуры и функции.

```
def Hanoi ( n, k, m ):
  if n == 0: return
  p = 6 - k - m
  Hanoi ( n-1, k, p )
  print ( k, "->", m )
  Hanoi ( n-1, p, m )
```

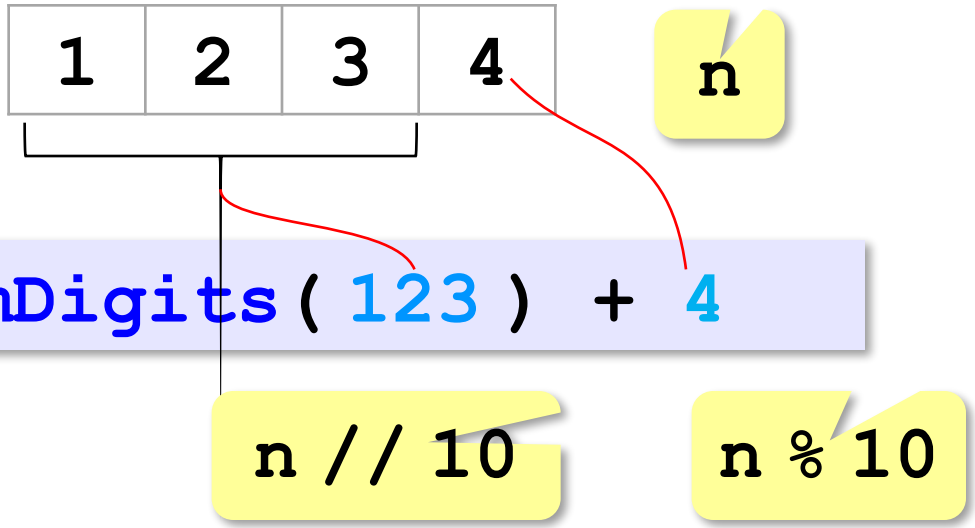
условие выхода из рекурсии

```
# основная программа
Hanoi( 4, 1, 3 )
```

# Вычисление суммы цифр числа

*Задача*. Написать рекурсивную функцию, которая вычисляет сумму цифр числа.

```
s = sumDigits( 1234 )
```




```
sumDigits( n )= sumDigits( n // 10 )+(n % 10)
sumDigits( n )= n для n < 10
```

? Это всё?

# Вычисление суммы цифр числа

последняя цифра

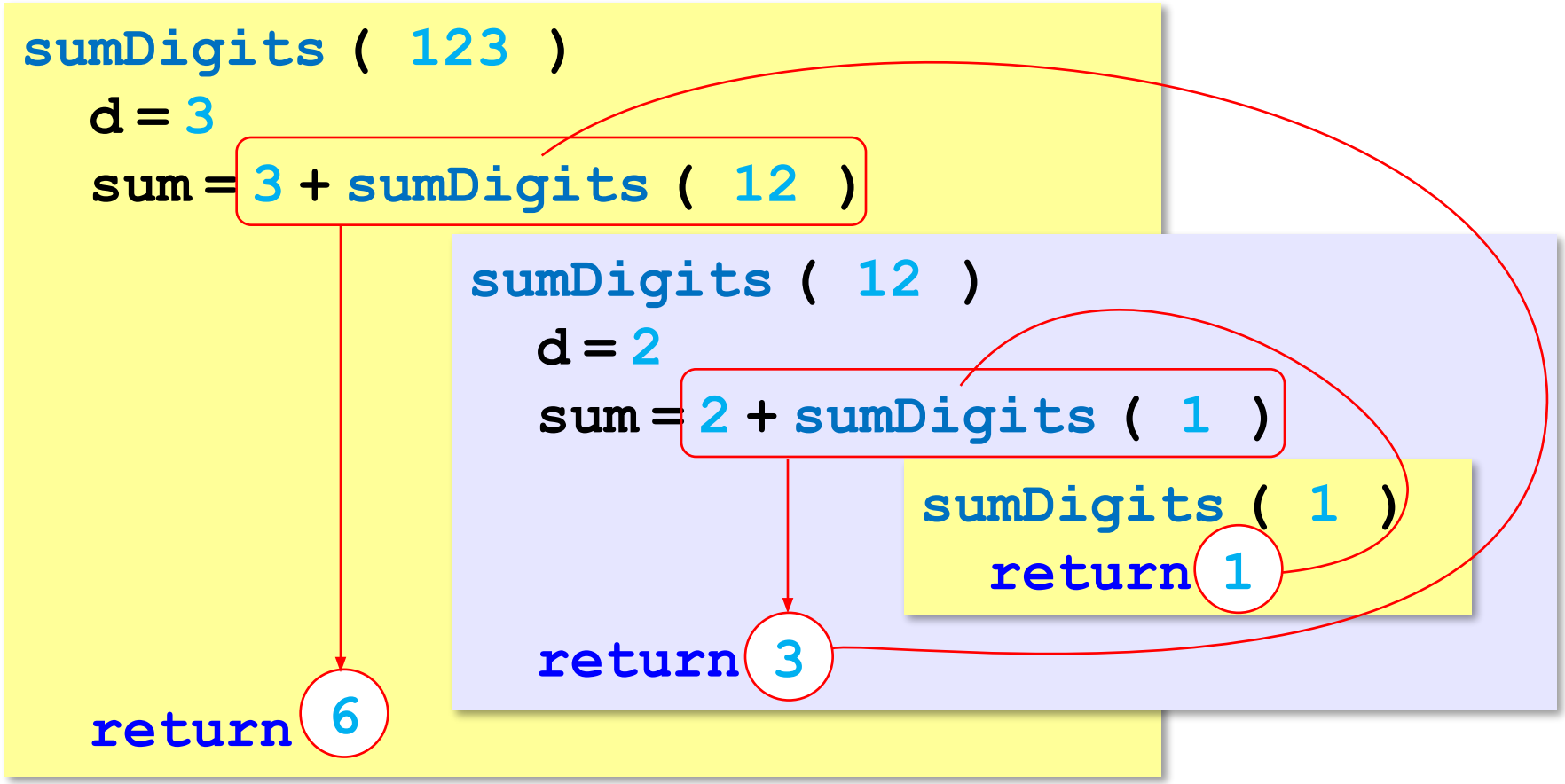
```
def sumDigits ( n ):
    if n<10: return n
    d=n%10
    sum=d+sumDigits ( n//10 )
    return sum
```

рекурсивный вызов

? Где условие окончания рекурсии?

```
sumDigits( 1234 )
    4 + sumDigits( 123 )
        4 + 3 + sumDigits( 12 )
            4 + 3 + 2 + sumDigits( 1 )
                4 + 3 + 2 + 1
```

# Вычисление суммы цифр числа



# Вывод двоичного кода числа

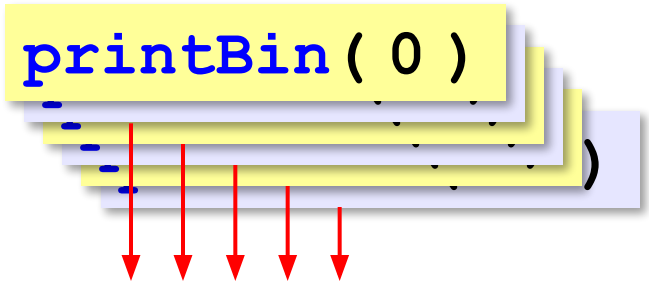
*Задача*. Написать рекурсивную процедуру, которая выводит двоичную запись числа.

```
def printBin ( n ):
    if n == 0: return
    printBin ( n // 2 )
    print ( n % 2, end = "" )
```

условие выхода из рекурсии

напечатать все цифры, кроме последней

вывести последнюю цифру




? Как без рекурсии?

# Алгоритм Евклида

**Алгоритм Евклида**. Чтобы найти НОД двух натуральных чисел, нужно вычитать из большего числа меньшее до тех пор, пока меньшее не станет равно нулю. Тогда второе число и есть НОД исходных чисел.

```
def NOD ( a, b ):
  if a == 0 or b == 0:
    return a + b;
  if a > b:
    return NOD( a - b, b )
  else:
    return NOD( a, b - a )
```

условие окончания рекурсии

рекурсивные вызовы

# Задачи

**«A»:** Напишите рекурсивную функцию, которая вычисляет НОД двух натуральных чисел, используя модифицированный алгоритм Евклида.

**Пример**:

```
Введите два натуральных числа:
7006652 112307574
НОД(7006652,112307574)=1234.
```

**«B»:** Напишите рекурсивную функцию, которая раскладывает число на простые сомножители.

**Пример**:

```
Введите натуральное число:
378
378 = 2*3*3*3*7
```

# Задачи

**«С»:** Дано натуральное число N. Требуется получить и вывести на экран количество всех возможных *различных* способов представления этого числа в виде суммы натуральных чисел (то есть, 1 + 2 и 2 + 1 – это один и тот же способ разложения числа 3). Решите задачу с помощью рекурсивной функции.

**Пример:**

```
Введите натуральное число:
4
Количество разложений: 4.
```

# Как работает рекурсия?

**Факториал:**

$$N! = \begin{cases} 1, & N = 1 \\ N \cdot (N-1)!, & N > 1 \end{cases}$$

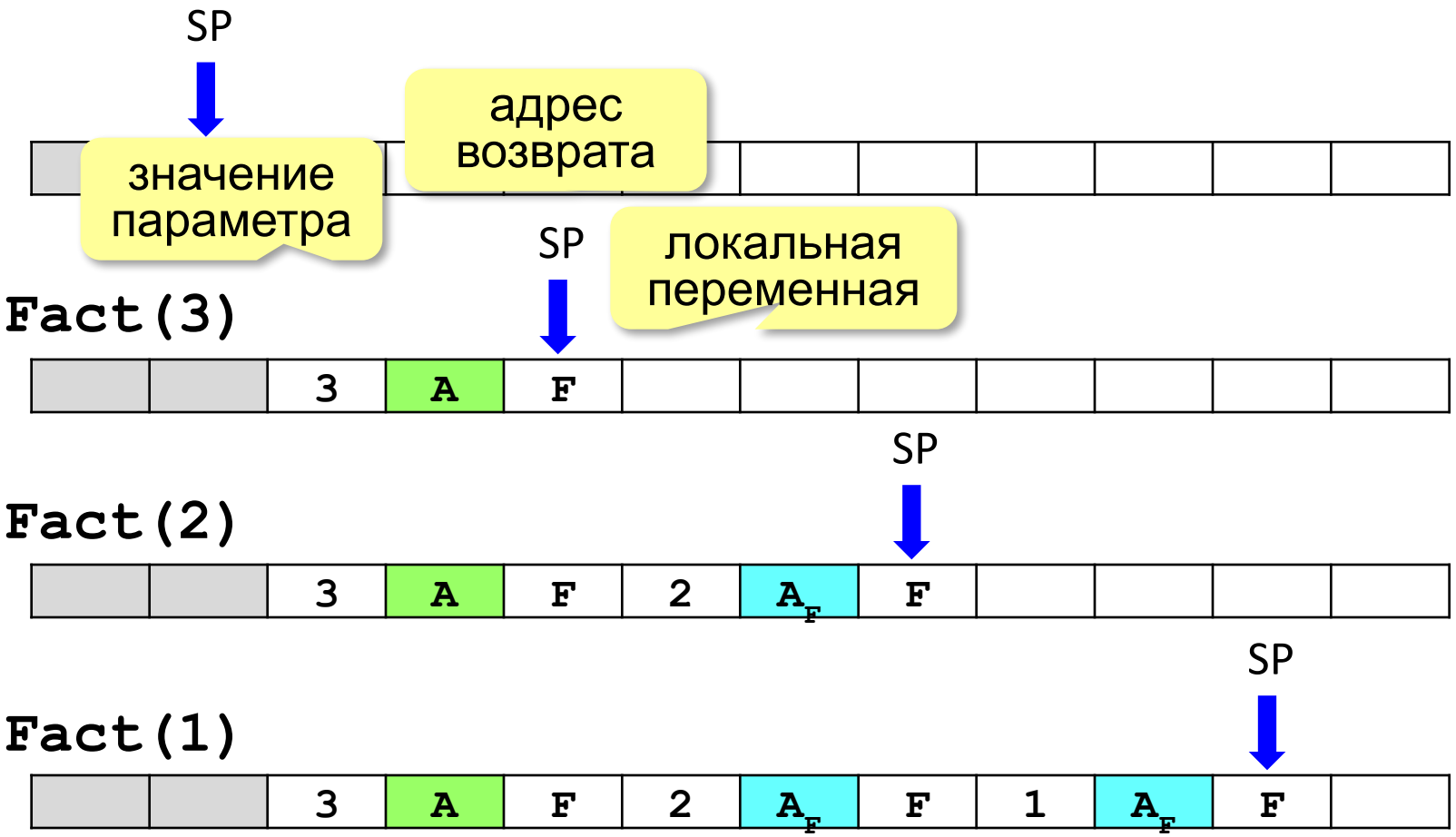
```
def Fact(N):
  print ( "->", N )
  if N <= 1: F = 1
  else:
    F = N * Fact ( N - 1 )
  print ( "<-", N )
  return F
```

```
-> N = 3
  -> N = 2
    -> N = 1
    <- N = 1
  <- N = 2
<- N = 3
```

? Как сохранить состояние функции перед рекурсивным вызовом?

# Стек

**Стек** – область памяти, в которой хранятся локальные переменные и адреса возврата.



# Рекурсия – «за» и «против»

- с каждым новым вызовом расходуется память в стеке (возможно переполнение стека)
- затраты на выполнение служебных операций при рекурсивном вызове

- программа становится более короткой и понятной

- возможно переполнение стека
- замедление работы

Любой рекурсивный алгоритм можно заменить нерекурсивным!

**итерационный** алгоритм

```
def Fact ( n ):
    f = 1
    for i in range(2,n+1):
        f *= i
    return f
```

# Анализ рекурсивных функций

*Задача*. Что выведет эта программа?

```
def f( x ):
    if x < 3:
        return 1
    else:
        return f( x-1 ) + 2
print( f( 5 ) )
```

$$f(x) = \begin{cases} 1, & x < 3 \\ f(x-1) + 2, & x \geq 3 \end{cases}$$

**Метод подстановки**:

```
f(5) = f(4) + 2 = 5 + 2 = 7
f(4) = f(3) + 2 = 3 + 2 = 5
f(3) = f(2) + 2 = 1 + 2 = 3
f(2) = 1
```

линейная структура

$f(5) \rightarrow f(4) \rightarrow f(3) \rightarrow f(2)$

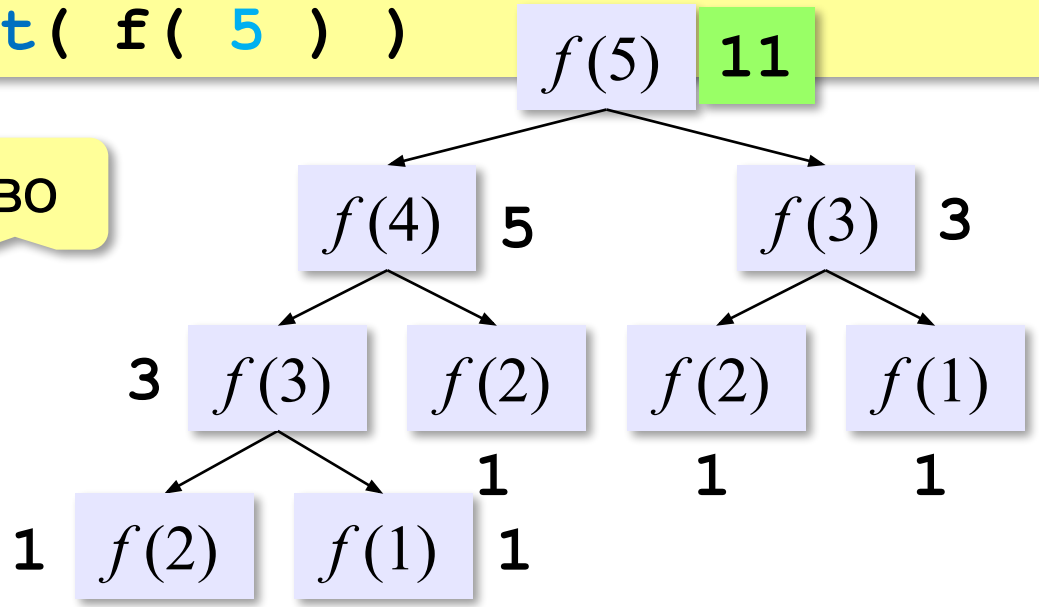
# Анализ рекурсивных функций

*Задача*. Что выведет эта программа?

```
def f( x ):
    if x < 3:
        return 1
    else:
        return f(x-1) + 2* f(x-2)
print( f( 5 ) )
```

$$f(x) = \begin{cases} 1, & x < 3 \\ f(x-1) + 2f(x-2), & x \geq 3 \end{cases}$$

дерево



# Анализ рекурсивных функций

Чему равно $f(5)$?

$$f(x)=\begin{cases}1, & x<3\\ f(x-1)+2f(x-2), & x\geq 3\end{cases}$$

**Табличный метод** :

| $x$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $f(x)$ | | | | | 11 |

*2 *2 *2

начальные значения

```
f(3) = f(2) + 2*f(1) = 3
f(4) = f(3) + 2*f(2) = 5
f(5) = f(4) + 2*f(3) = 11
```

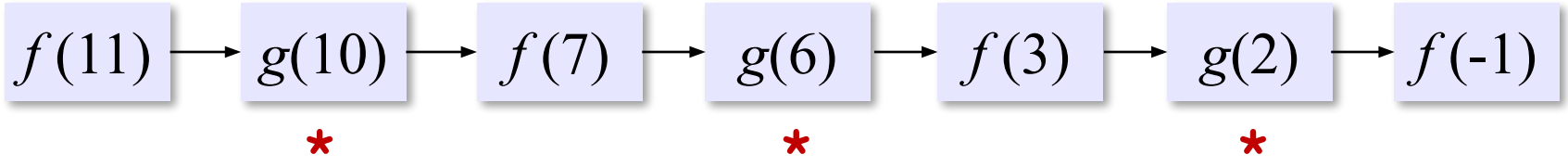
# Анализ рекурсивных функций

*Задача*. Сколько звёздочек выведет эта программа?

```
def f( x ):
  if x > 0: g(x-1)

def g( x ):
  print( "*" )
  if x > 1: f(x-3)

f( 11 )
```




Ответ: 3

# Анализ рекурсивных функций

*Задача*. Сколько звёздочек выведет эта программа?

```
def f( x ):
    if x > 0:
        g( x-1 )
        f( x-2 )
    print( "*" )

def g( x ):
    print( "*" )
    if x > 1: f(x-3)

f( 9 )
```

$$f(x)=\begin{cases}1, & x\le 0\\ g(x-1)+f(x-2)+1, & x>0\end{cases}$$

$$g(x)=\begin{cases}1, & x\le 1\\ 1+f(x-3), & x>1\end{cases}$$

# Анализ рекурсивных функций

$$f(x) = \begin{cases} 1, & x \le 0 \\ g(x-1) + f(x-2) + 1, & x > 0 \end{cases}$$

$$g(x) = \begin{cases} 1, & x \le 1 \\ 1 + f(x-3), & x > 1 \end{cases}$$

| $x$ | –1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $f(x)$ | **1** | **1** | | | | | | | | | |
| $g(x)$ | **1** | **1** | **1** | | | | | | | | |

$f(1) = g(0) + f(-1) + 1 = 3$

$f(2) = g(1) + f(0) + 1 = 3$

$g(2) = 1 + f(-1) = 2$

Ответ: 32

# Конец фильма

**ПОЛЯКОВ Константин Юрьевич**
д.т.н., учитель информатики
ГБОУ СОШ № 163, г. Санкт-Петербург
kpolyakov@mail.ru

**ЕРЕМИН Евгений Александрович**
к.ф.-м.н., доцент кафедры мультимедийной дидактики и ИТО ПГГПУ, г. Пермь
eremin@pspu.ac.ru

# Источники иллюстраций

1. old-moneta.ru
2. www.random.org
3. www.allruletka.ru
4. www.lotterypros.com
5. logos.cs.uic.edu
6. ru.wikipedia.org
7. www.cgtrader.com
8. иллюстрации художников издательства «Бином»
9. авторские материалы