

# Программирование на языке Python

- § 54. Алгоритм и его свойства
- § 55. Простейшие программы
- § 56. Вычисления
- § 57. Ветвления
- § 58. Циклические алгоритмы
- § 59. Процедуры
- § 60. Функции
- § 61. Рекурсия

# Программирование на языке Python

## § 54. Алгоритм и его свойства

# Что такое алгоритм?

**Алгоритм** — это точное описание порядка действий, которые должен выполнить исполнитель для решения задачи за конечное время.

**Исполнитель** – это устройство или одушевленное существо (человек), способное понять и выполнить команды, составляющие алгоритм.

**Формальные исполнители:** не понимают (и не могут понять) смысл команд.



Мухаммед ал-Хорезми  
(ок. 783—ок. 850 гг.)

# Свойства алгоритма

**Дискретность** — алгоритм состоит из отдельных команд, каждая из которых выполняется за конечное время.

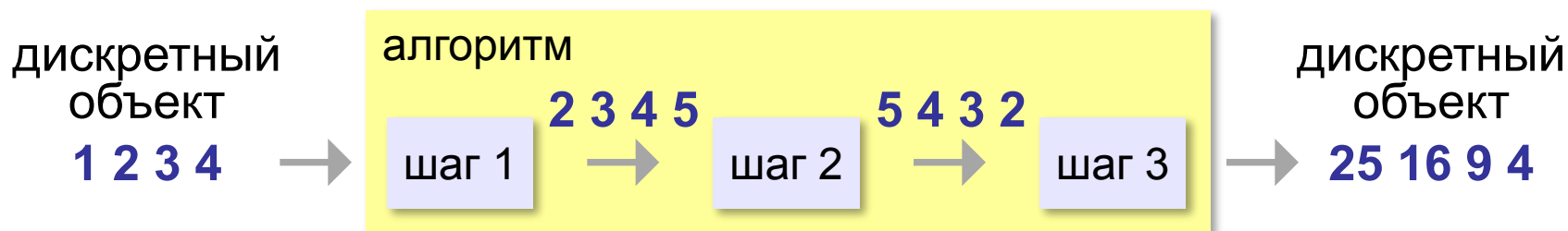
**Детерминированность** (определённость) — при каждом запуске алгоритма с одними и теми же исходными данными получается один и тот же результат.

**Понятность** — алгоритм содержит только команды, входящие в **систему команд исполнителя**.

**Конечность** (результативность) — для корректного набора данных алгоритм должен завершаться через конечное время.

**Корректность** — для допустимых исходных данных алгоритм должен приводить к правильному результату.

# Как работает алгоритм?



- получает на вход дискретный объект
- в результате строит другой дискретный объект (или выдаёт сообщение об ошибке)
- обрабатывает объект по шагам
- на каждом шаге получается новый дискретный объект

# Способы записи алгоритмов

---

- **естественный язык**

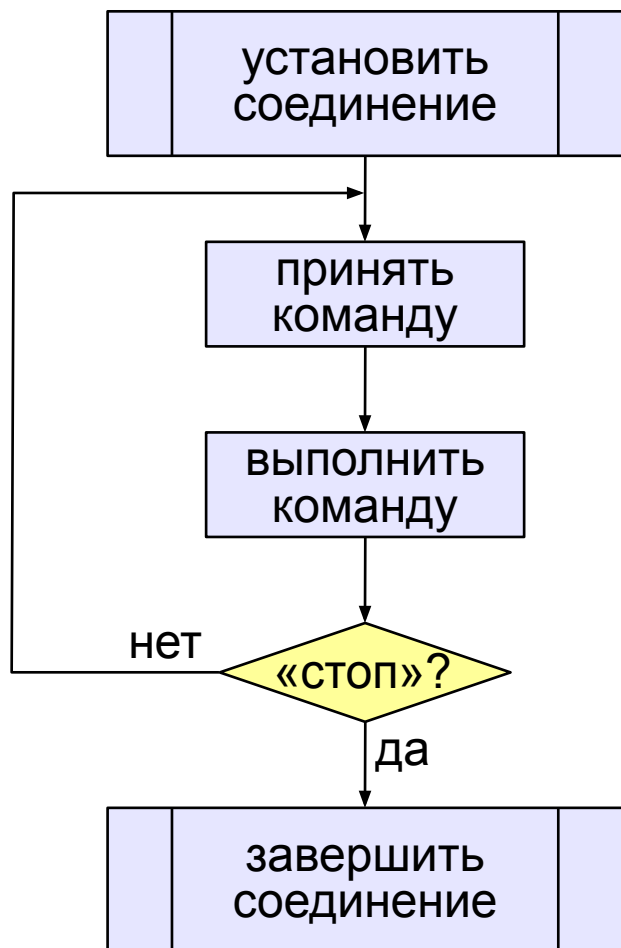
```
установить соединение  
пока не принята команда «стоп»  
    принять команду  
    выполнить команду  
завершить сеанс связи
```

- **псевдокод**

```
установить соединение  
начало цикла  
    принять команду  
    выполнить команду  
конец цикла при команда = 'stop'  
завершить сеанс связи
```

# Способы записи алгоритмов

- блок-схема



- программа

```
установитьСоединение
начало цикла
cmd := получитьКоманду
выполнитьКоманду (cmd)
конец при cmd = 'stop'
закретьСоединение
```

# Программирование на языке Python

## § 55. Простейшие программы



# 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' )
```

# Задания

---

«В»: Вывести на экран текст «лесенкой»

Вася

пошел

гулять

«С»: Вывести на экран рисунок из букв

```
  Ж
 ЖЖЖ
 ЖЖЖЖЖ
 ЖЖЖЖЖЖЖ
 НН  НН
 ZZZZZ
```

# Сложение чисел

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

**Протокол:**

Введите два целых числа

компьютер

25 30

пользователь

25+30=55

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**

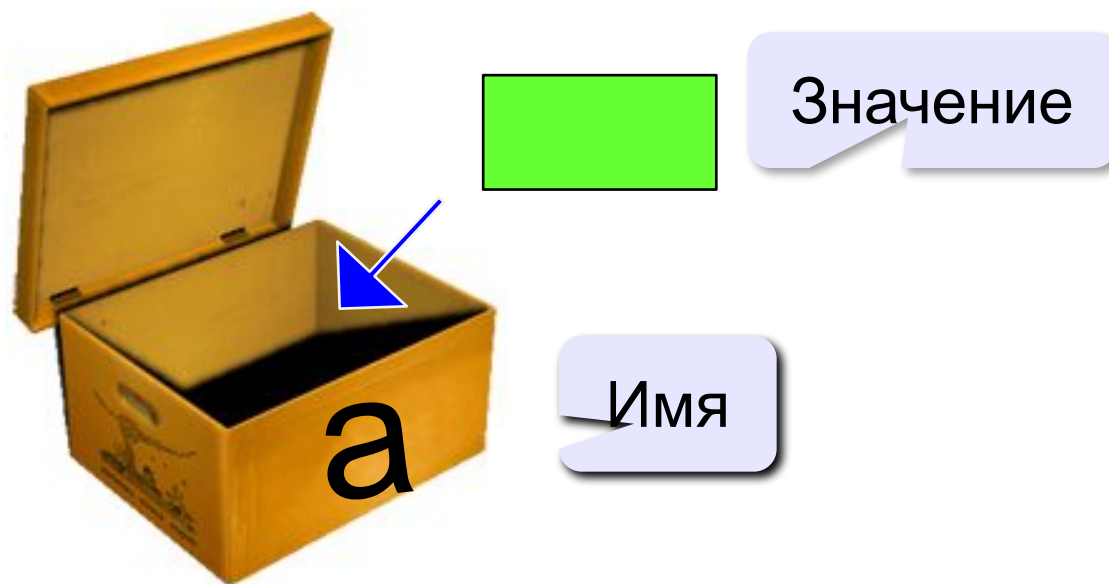
**Псевдокод** – алгоритм на русском языке с элементами языка программирования.



**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**

# Типы переменных

```
a = 4  
print ( type (a) )  
<class 'int'>
```

целое число (*integer*)

```
a = 4.5  
print ( type (a) )  
<class 'float'>
```

вещественное число

```
a = "Вася"  
print ( type (a) )  
<class 'str'>
```

символьная строка

```
a = True  
print ( type (a) )  
<class 'bool'>
```

логическая



# Зачем нужен тип переменной?

---

Тип определяет:

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

# Как записать значение в переменную?

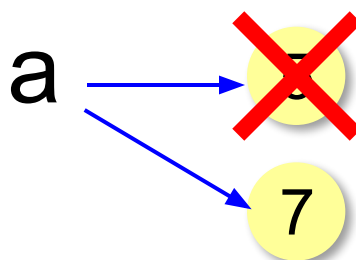
оператор  
присваивания



При записи нового значения  
старое удаляется из памяти!

a = 5

a = 7

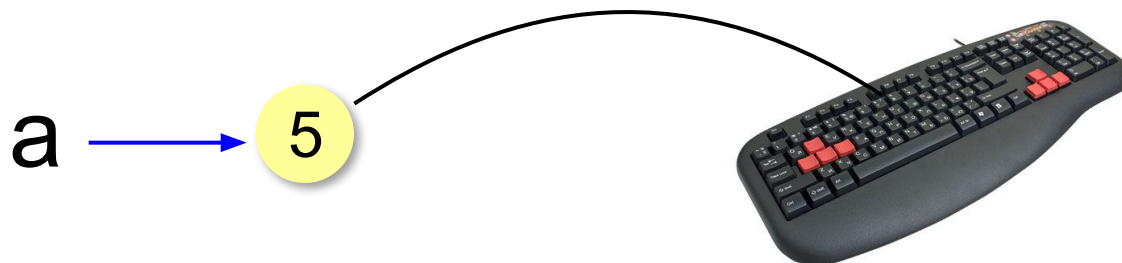


**Оператор** – это команда языка  
программирования (инструкция).

**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 ()
```

ввести строку с клавиатуры  
и связать с переменной a

```
b = input ()
```

```
c = a + b
```

```
print ( c )
```

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()`

целые

применить

разделить строку на части по пробелам

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

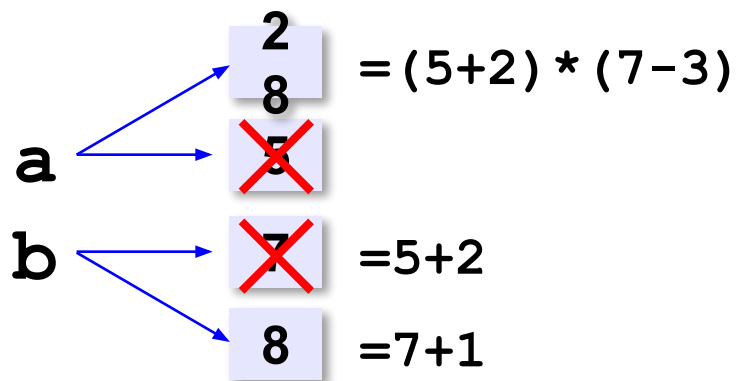


Что не так?

```
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

Протокол:

КОМПЬЮТЕР

Введите два целых числа

25 30

ПОЛЬЗОВАТЕЛЬ

25 + 30 = 55

# Форматный вывод

целое

```
a = 123
```

```
print ( "{:5d}".format(a) )
```

123

5 знаков

```
a = 5
```

```
print ( "{:5d}{:5d}{:5d}".format  
        (a, a*a, a*a*a) )
```

5 25 125

5 знаков 5 знаков 5 знаков

# Программирование на языке Python

## § 56. Вычисления

# Типы данных

---

- `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'>
```

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

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

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

перенос на  
следующую строку

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
```

# 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  $\geq 0$

$$-7 = (-4) * 2 + 1$$



# 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

# 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 · 10<sup>2</sup>

# Вещественные числа

## Экспоненциальный формат:

```
x = 1. / 30000
```

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

```
x = 12345678.
```

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

 $3,333333 \cdot 10^{-5}$ 

```
3.333333e-05
```

```
1.234568e+07
```

```
x = 123.456
```

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

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

 $1,234568 \cdot 10^7$ 

```
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

```
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
```

# 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

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

318458191041

209938992481

# Линейный конгруэнтный генератор

$$X = (a * X + b) \% c \quad | \quad \text{интервал от } 0 \text{ до } c-1$$

$$X = (X + 3) \% 10 \quad | \quad \text{интервал от } 0 \text{ до } 9$$

$$X = 0 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 2 \rightarrow 5 \rightarrow 8$$

$$8 \rightarrow 1 \rightarrow 4 \rightarrow 7 \rightarrow 0$$

зерно

защипливание



Важно 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

---

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

**Пример:**

Введите три целых числа:

5 7 8

$$5+7+8=20$$

$$5*7*8=280$$

$$(5+7+8)/3=6.667$$

«В»: Ввести с клавиатуры координаты двух точек (А и В) на плоскости (вещественные числа). Вычислить длину отрезка АВ.

**Пример:**

Введите координаты точки А:

5.5 3.5

Введите координаты точки В:

1.5 2

$$\text{Длина отрезка АВ} = 4.272$$

# Задачи

---

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

**Пример:**

Получено число 123.

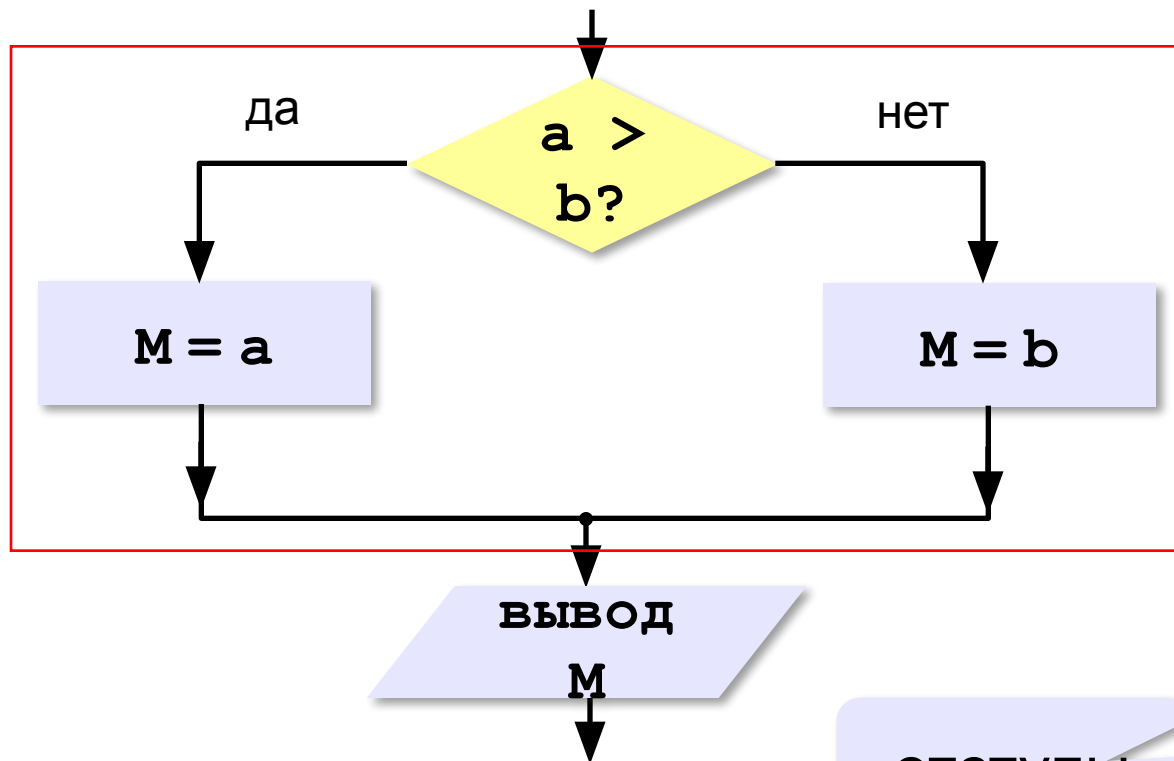
Его цифры 1, 2, 3.

# Программирование на языке Python

## § 57. Ветвления

# Условный оператор

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



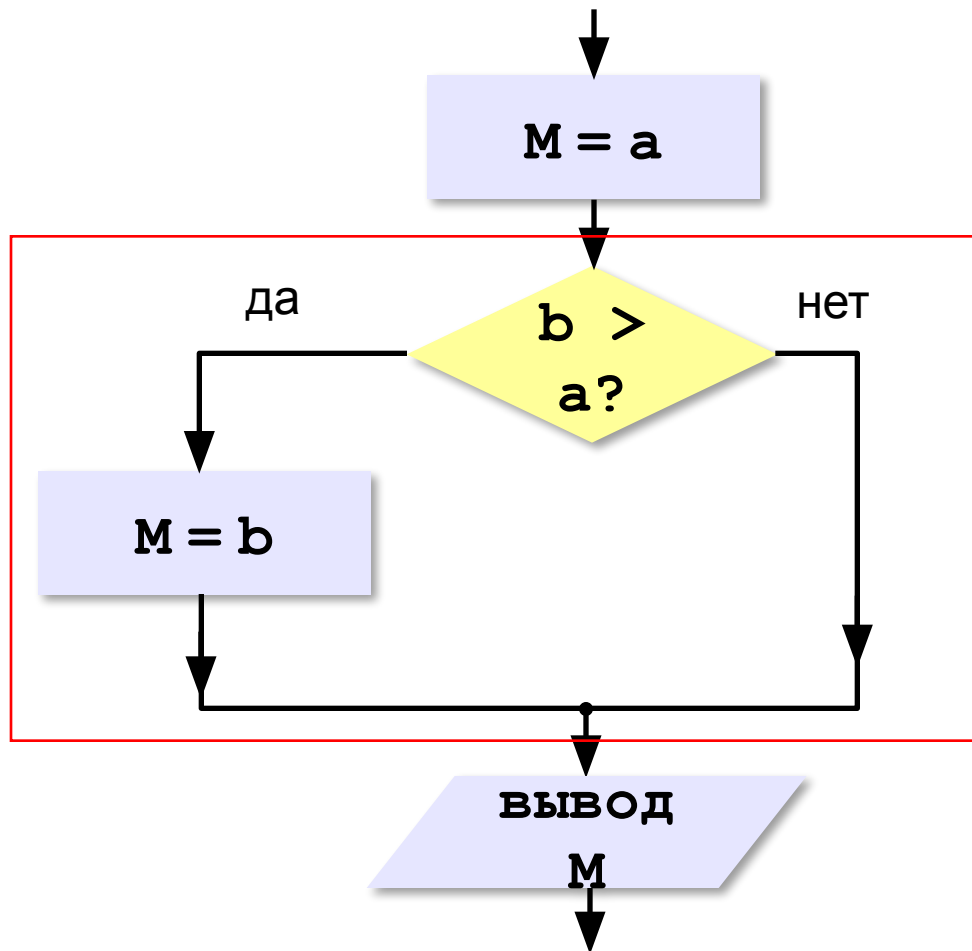
полная  
форма  
ветвления

? 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

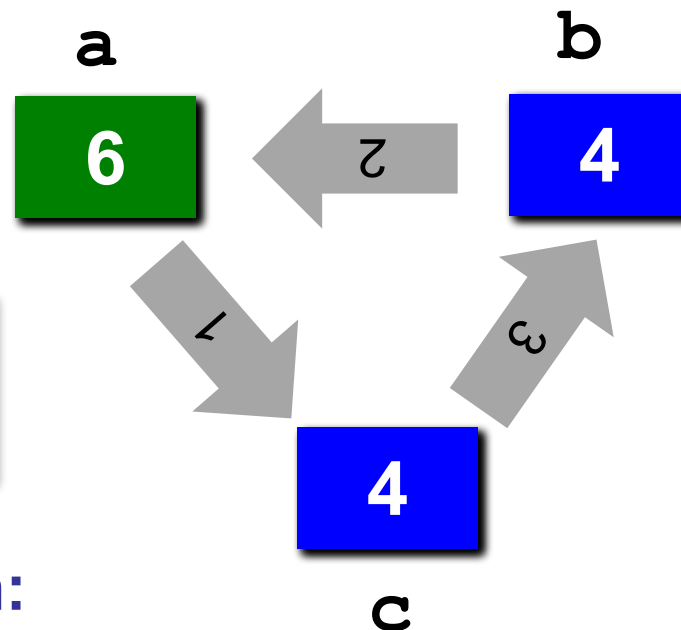
```
if a < b:
```

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

?

Зачем нужен?

вложенный  
условный оператор

# Каскадное ветвление

---

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

---

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

**Пример:**

Введите три целых числа:

1 5 4

Максимальное число 5

**«В»:** 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

---

**«А»:** 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

---

**«В»:** 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.

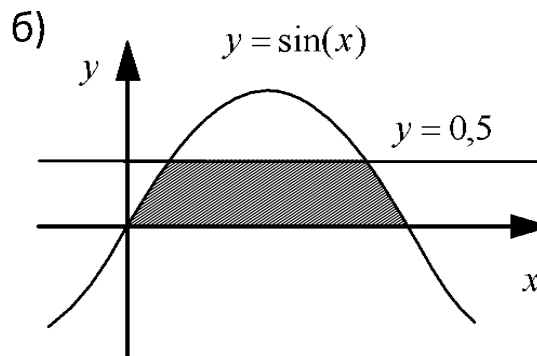
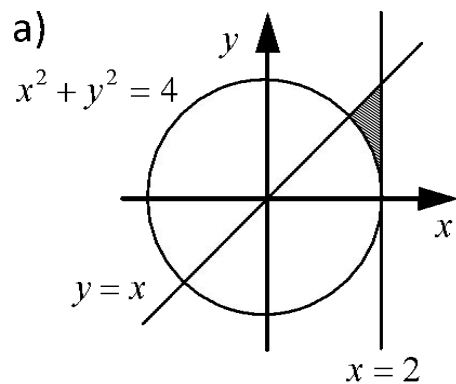
**Пример:**

Введите возраст: **22**

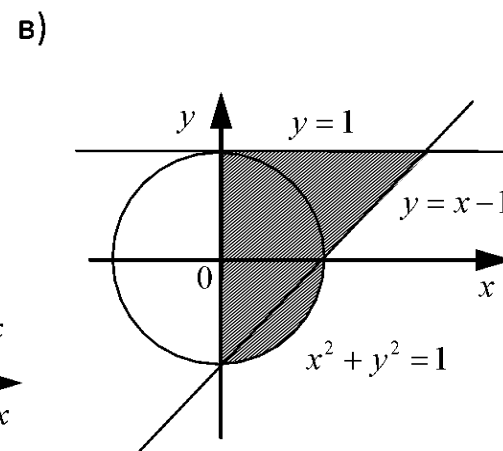
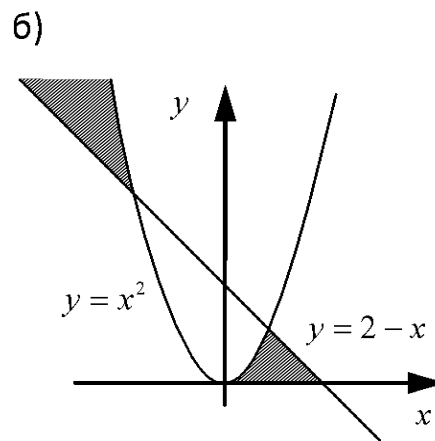
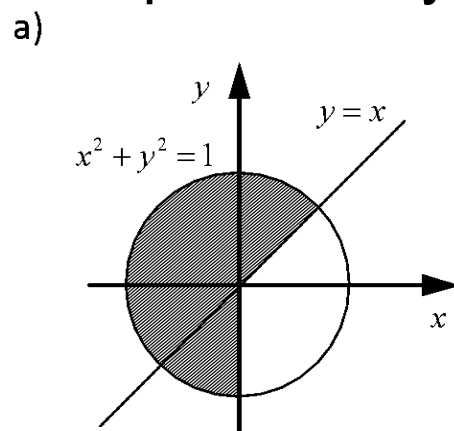
Вам 22 года.

# Задачи

«А»: Напишите условие, которое определяет заштрихованную область.

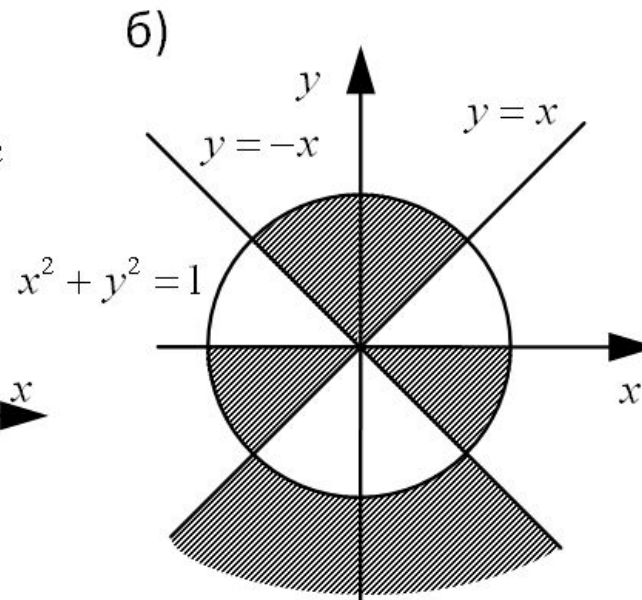
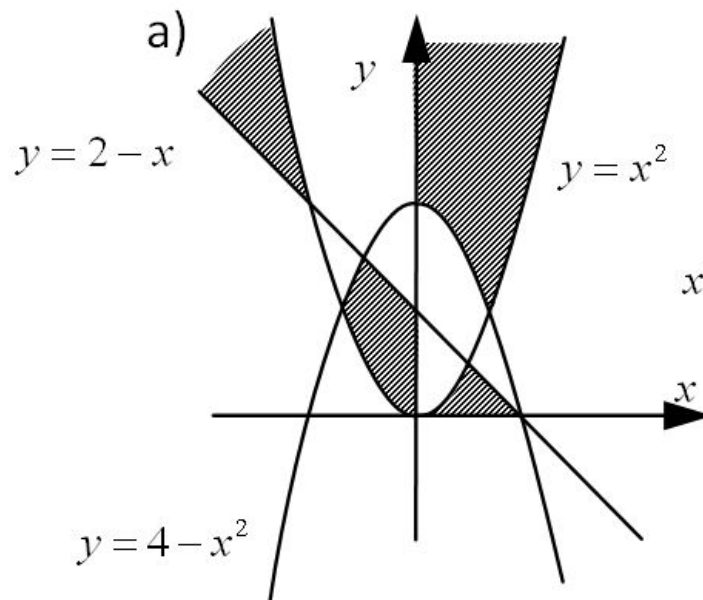


«В»: Напишите условие, которое определяет заштрихованную область.



# Задачи

«С»: Напишите условие, которое определяет заштрихованную область.



# Программирование на языке Python

## § 58. 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

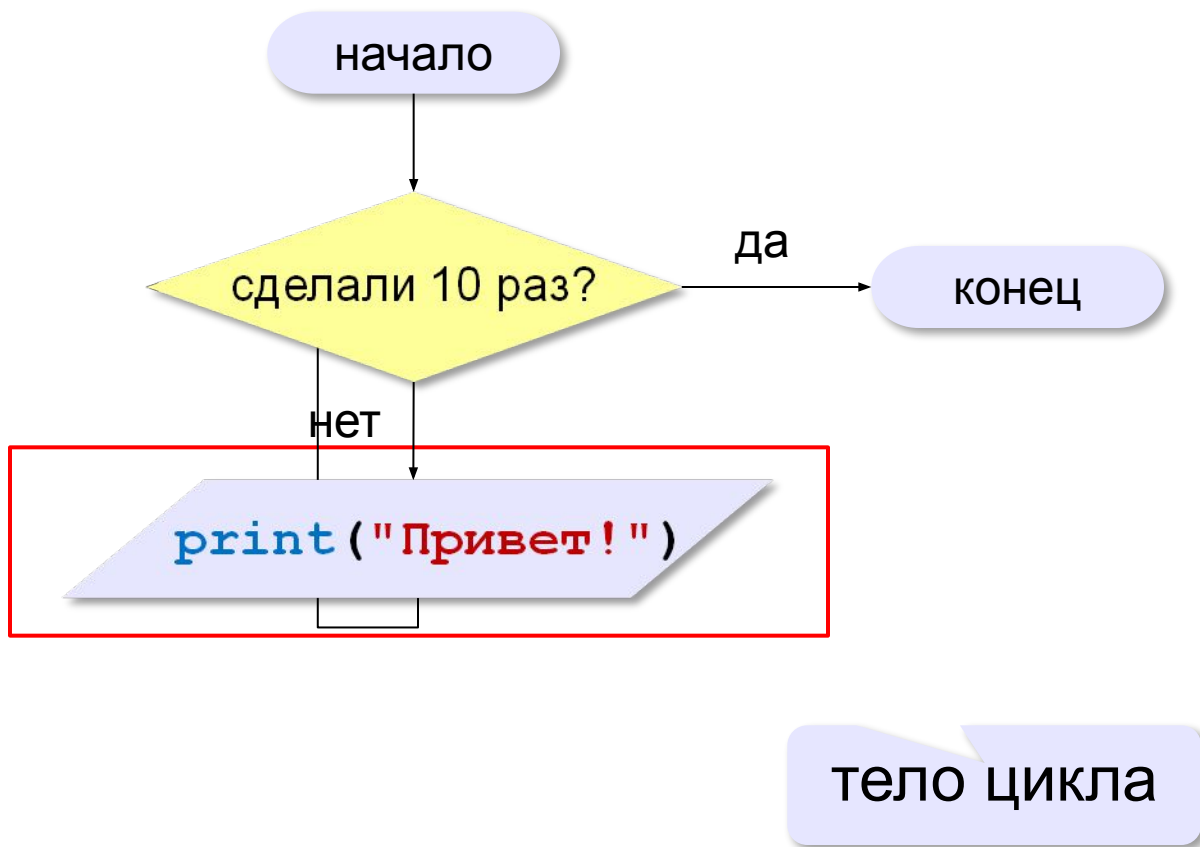
---

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



Что плохо?

# Блок-схема цикла



# 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
```

# Цикл с условием

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!

# Цикл с условием

---

При известном количестве шагов:

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

Заикливание:

```
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
```

**защелкивание**

## 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**

# Задачи

---

**«А»:** 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**

$10 * 10 = 100$

$11 * 11 = 121$

$12 * 12 = 144$

**«В»:** 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

---

**«С»:** 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

---

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

**Пример:**

Введите натуральное число:

**12345**

Сумма цифр 15.

**«В»:** 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 .**

# 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 )
```



Что получится?

1, 3, 5, 7, 9

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

100

81

64

49

36

25

16

9

4

1

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

---

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

«В»: 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

---

«С»: 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?

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

```
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

---

**«А»:** 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

**«В»:** 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<sub>2</sub>

?

Как вывести первую цифру?

n := **1** 0 1 1 0 0 1 0<sub>2</sub>    разряды  
           7 6 5 4 3 2 1 0

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
```

178  $\Rightarrow$  10110010

| n   | k   | ВЫВОД |
|-----|-----|-------|
| 178 | 128 | 1     |
|     |     |       |
|     |     |       |
|     |     |       |
|     |     |       |



Результат зависит от n!

# Процедура с параметрами

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

локальная  
переменная

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

```
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 )
```

работаем с  
глобальной  
переменной

1

# Неправильная процедура

```
x = 5; y = 10
```

```
xSum ()
```



Что плохо?

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



- 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:

|          |          |
|----------|----------|
| <b>x</b> | <b>y</b> |
| 5        | 10       |
| <b>z</b> | <b>w</b> |
| 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:

|               |               |    |
|---------------|---------------|----|
| <b>a</b>      | <b>b</b>      |    |
| <del>25</del> | <del>30</del> | 15 |
|               |               | 20 |
|               |               | 52 |

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

# Задачи

---

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

**Пример:**

Введите  $N$ :

10

-----

«В»: 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**

**ММХІІІ**

# Программирование на языке Python

## § 60. Функции

# Что такое функция?

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

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

**Алгоритм:**

```
сумма = 0
пока n != 0:
    сумма += n % 10
    n = n // 10
```

# Сумма цифр числа

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

передача  
результата

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

```
# сохранить в переменной  
n = sumDigits(12345)
```

```
# сразу вывод на экран  
print( sumDigits(12345) )
```



Что плохо?

# 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

---

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

**Пример:**

Введите два натуральных числа:

**7006652 112307574**

$\text{НОД}(7006652, 112307574) = 1234.$

**«В»:** 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
```

кортеж – набор  
элементов

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

q[0]

q[1]



# Задачи

---

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

**Пример:**

**Введите три натуральных числа:**

**10 15 5**

**5 10 15**

**«В»:** 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

---

Задача. Найти все простые числа в диапазоне от 2 до 1000.

```
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() )  
if isPrime(n):  
    print ( n, "- простое число" )
```

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

# Задачи

---

**«А»:** 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

---

**«В»:** 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**

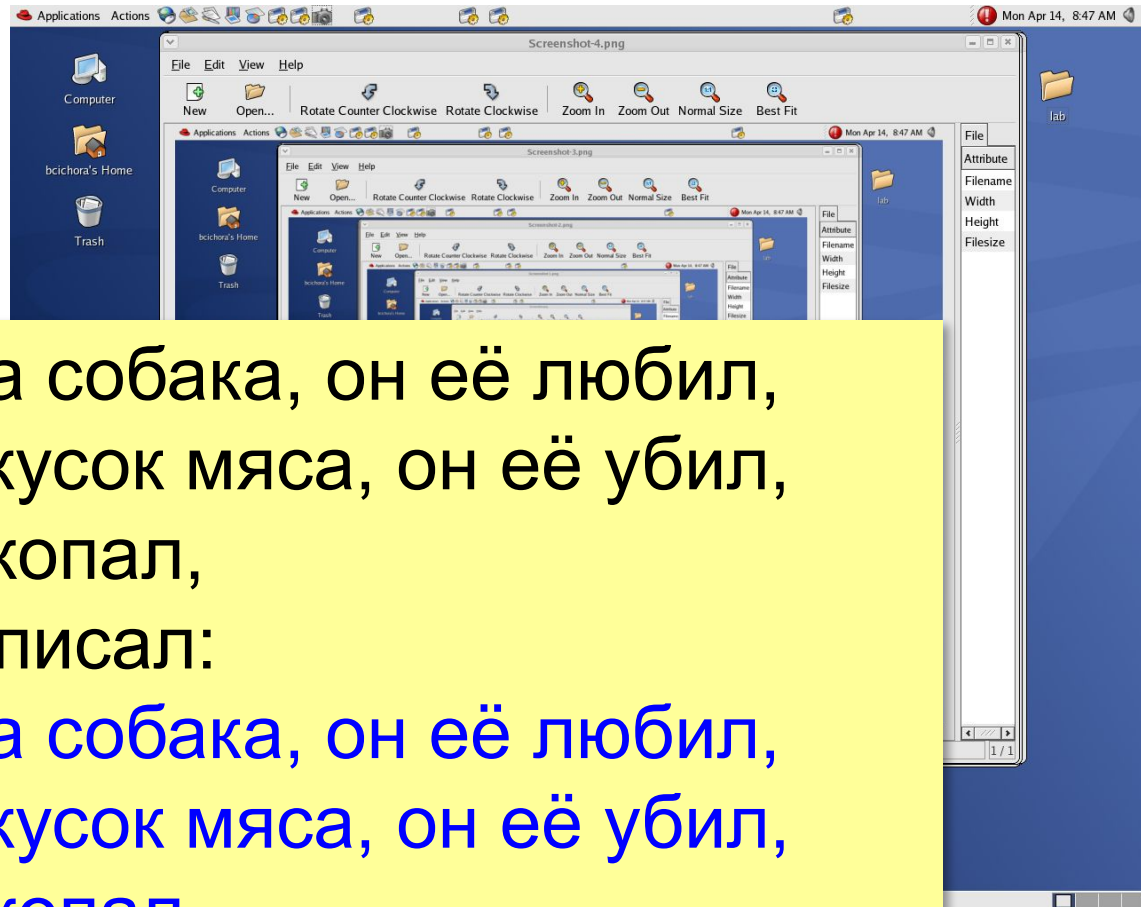
Число 19 не гиперпростое.



# 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:

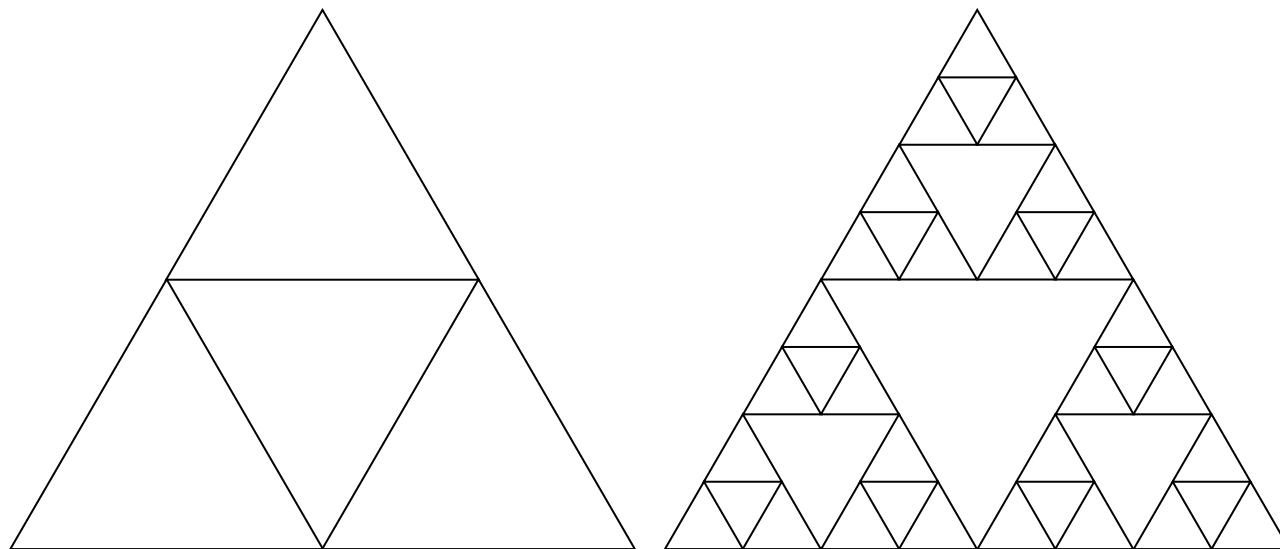
- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$  при  $n > 2$

**1, 1, 2, 3, 5, 8, 13, 21, 34, ...**

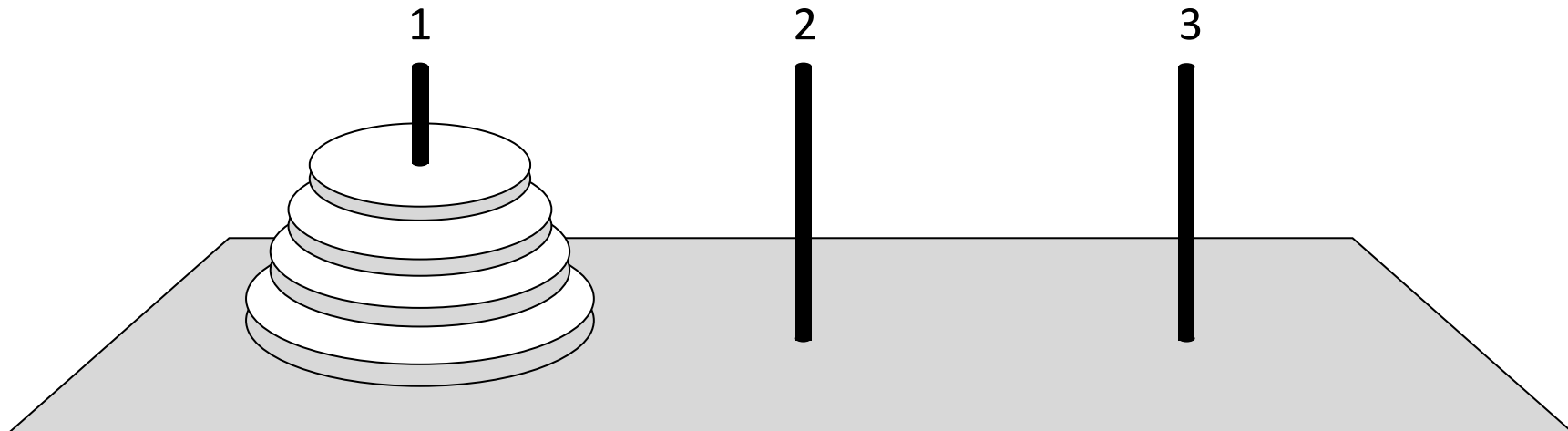
# Фракталы

**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 )
```

рекурсия `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



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 )
```

# Вывод двоичного кода 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

```
printBin ( 0 )
```



**?** Как без рекурсии?



# Вычисление суммы цифр числа

```
def sumDig ( n ) :  
    sum = n % 10  
    if n >= 10 :  
        sum += sumDig ( n // 10 )  
    return sum
```

последняя цифра

рекурсивный вызов

**?** Где условие окончания рекурсии?

sumDig ( 1234 )

4 + sumDig ( 123 )

4 + 3 + sumDig ( 12 )

4 + 3 + 2 + sumDig ( 1 )

4 + 3 + 2 + 1

# 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

---

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

**Пример:**

Введите два натуральных числа :

**7006652 112307574**

**НОД (7006652 , 112307574) = 1234 .**

**«В»:** 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
```

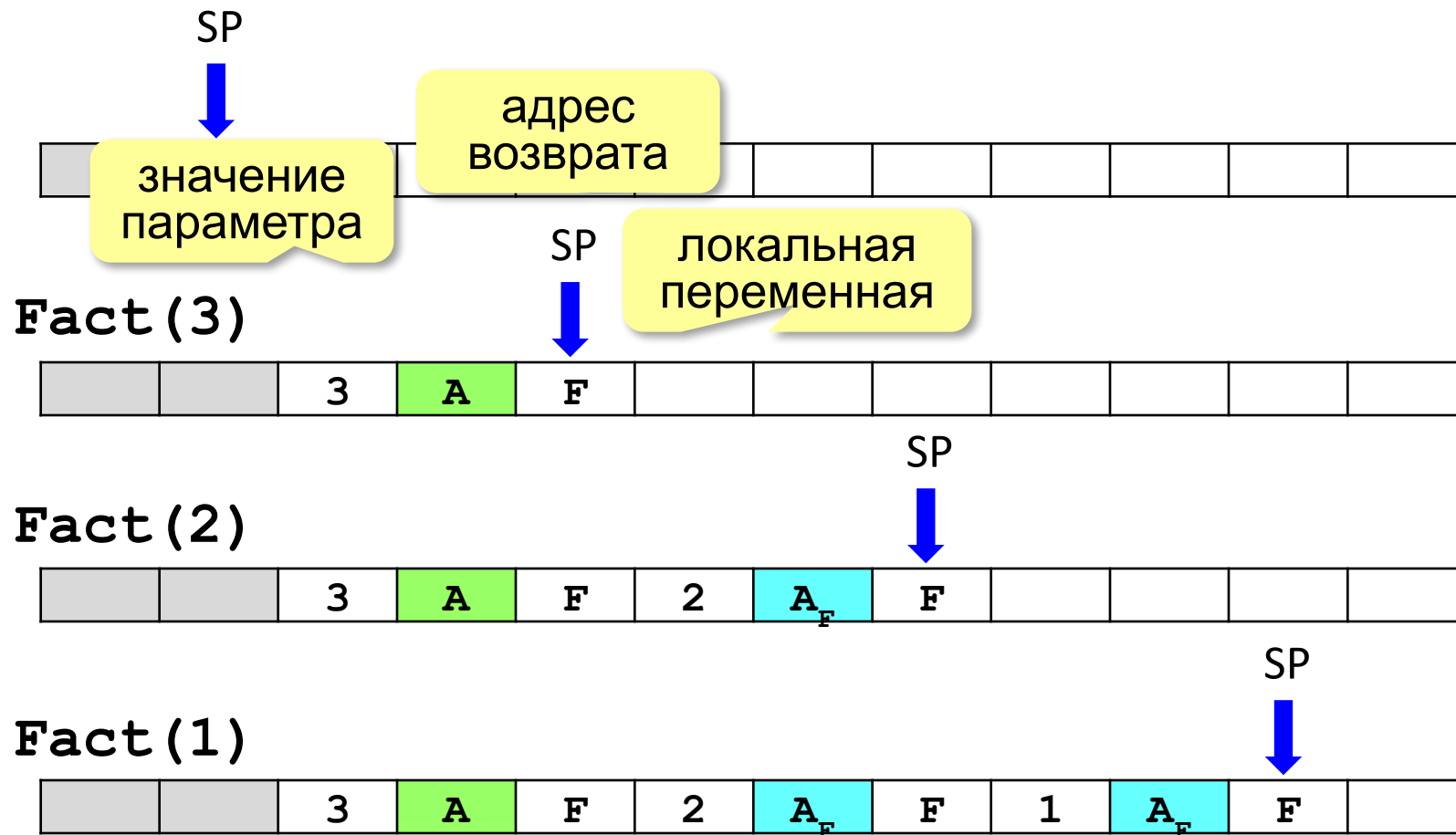
```
-> N = 3  
    -> N = 2  
        -> N = 1  
            <- N = 1  
        <- N = 2  
    <- N = 3
```



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
```

# Конец фильма

---

**ПОЛЯКОВ Константин Юрьевич**

д.т.н., учитель информатики

ГБОУ СОШ № 163, г. Санкт-Петербург

[kpolyakov@mail.ru](mailto:kpolyakov@mail.ru)

**ЕРЕМИН Евгений Александрович**

к.ф.-м.н., доцент кафедры мультимедийной

дидактики и ИТО ПГГПУ, г. Пермь

[eremin@pspu.ac.ru](mailto:eremin@pspu.ac.ru)



# Источники иллюстраций

---

1. [old-moneta.ru](http://old-moneta.ru)
2. [www.random.org](http://www.random.org)
3. [www.allruletka.ru](http://www.allruletka.ru)
4. [www.lotterypros.com](http://www.lotterypros.com)
5. [logos.cs.uic.edu](http://logos.cs.uic.edu)
6. [ru.wikipedia.org](http://ru.wikipedia.org)
7. иллюстрации художников издательства «Бином»
8. авторские материалы