Программирование на языке Python

Программирование на языке Python

Алгоритм и его свойства

Что такое алгоритм?

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

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

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

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

Свойства алгоритма

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

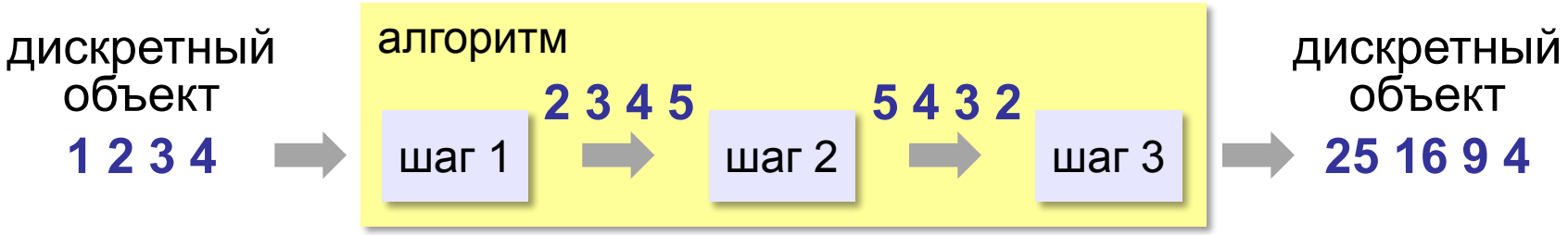
Детерминированность (определённость) — при каждом запуске алгоритма с одними и теми же исходными данными получается один и тот же результат.

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**

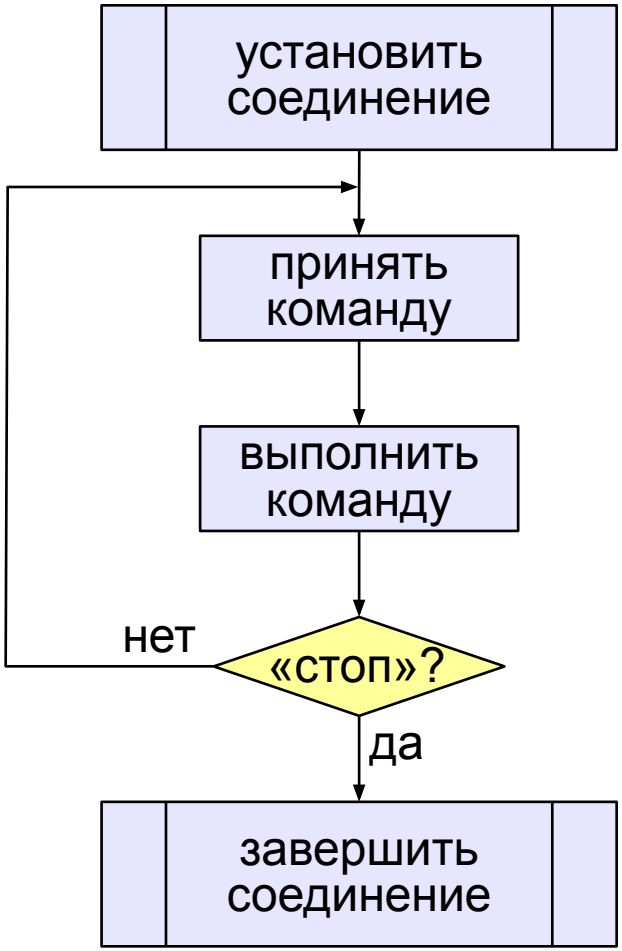
```
установить соединение
пока не принята команда «стоп»
   принять команду
   выполнить команду
завершить сеанс связи
```

- **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'
закрытьСоединение
```

Программирование на языке Python

Простейшие программы

Простейшая программа

```
# Это пустая программа
```

комментарии после # не обрабатываются

? Что делает эта программа?

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

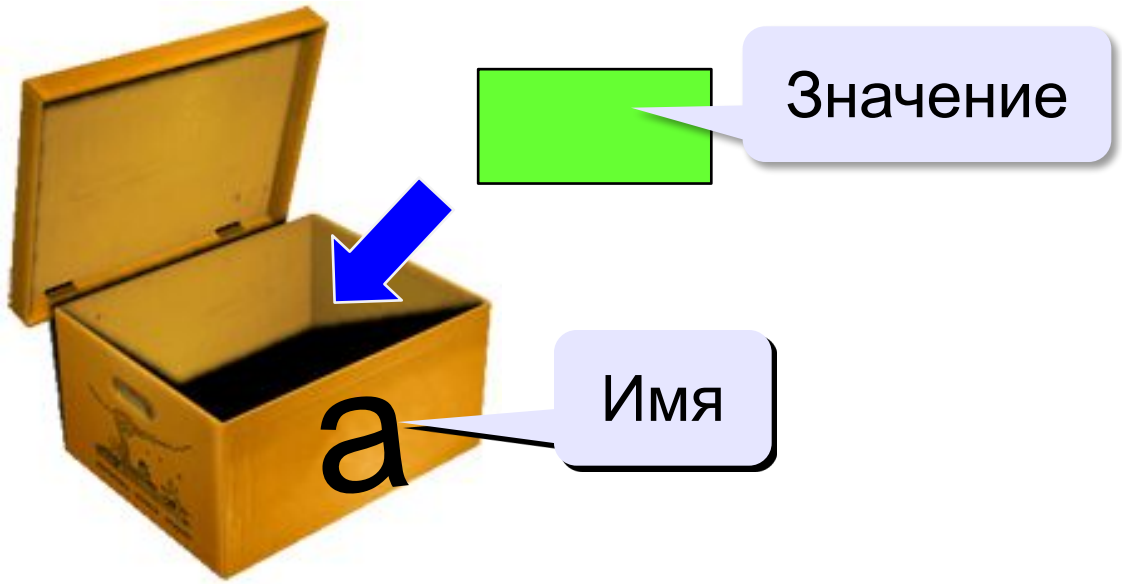
```
ввести два числа
вычислить их сумму
вывести сумму на экран
```

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

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?

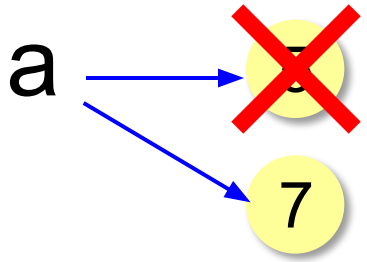
Тип определяет:

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

Как записать значение в переменную?

оператор присваивания

```
a = 5
a = 7
```

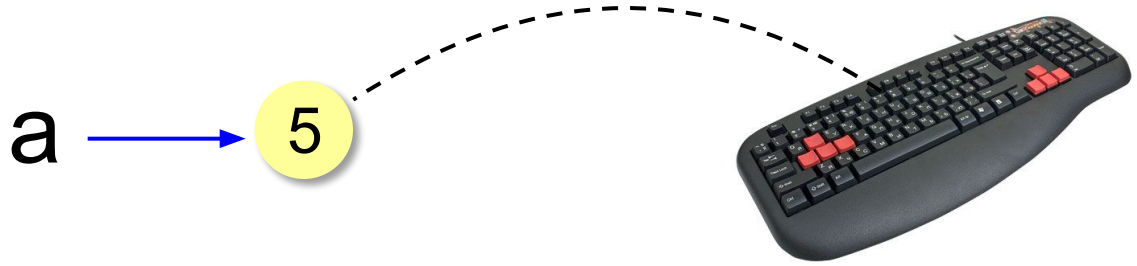


При записи нового значения старое удаляется из памяти!

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!

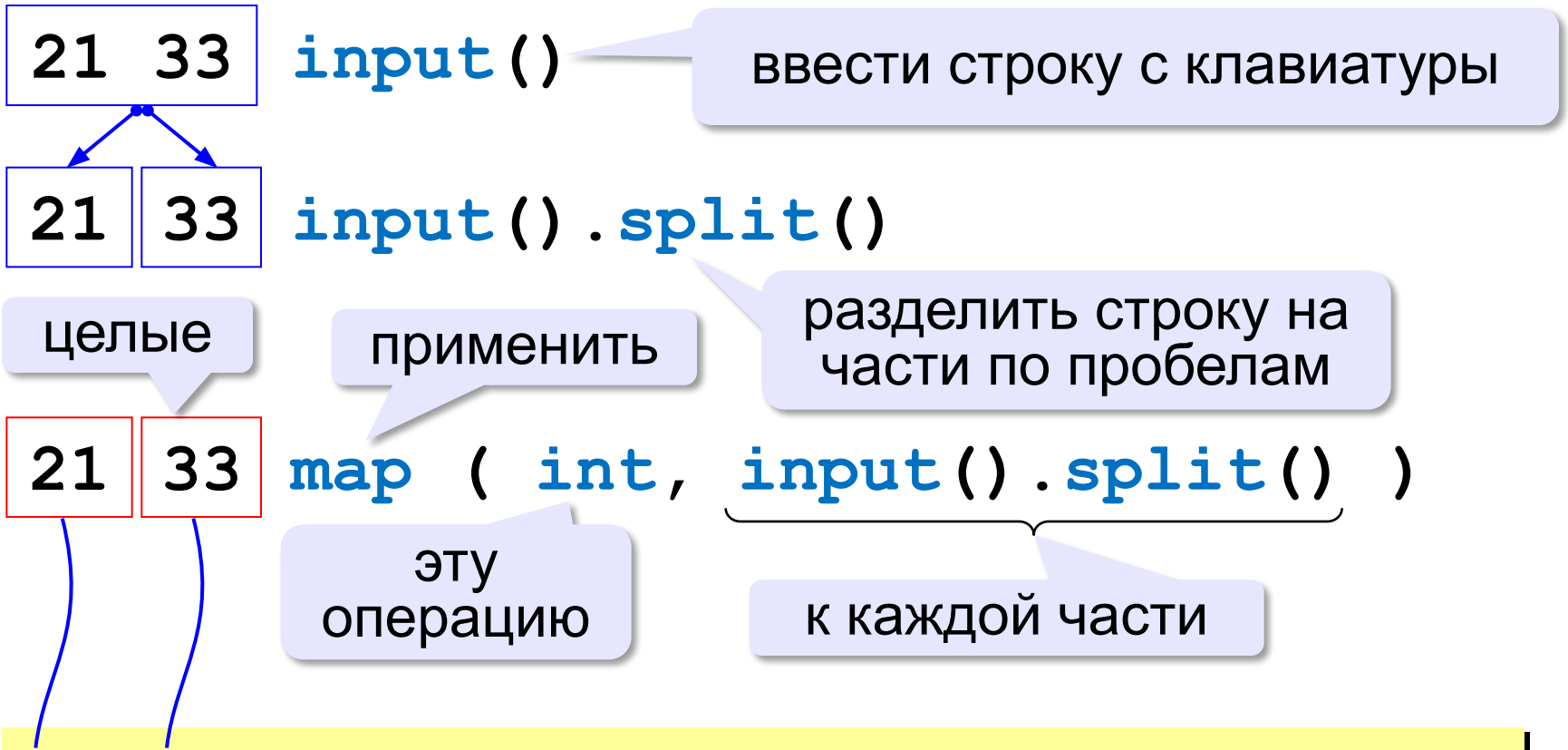
преобразовать в целое число

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

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

Ввод двух значений в одной строке

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




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

Ввод с подсказкой

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

подсказка

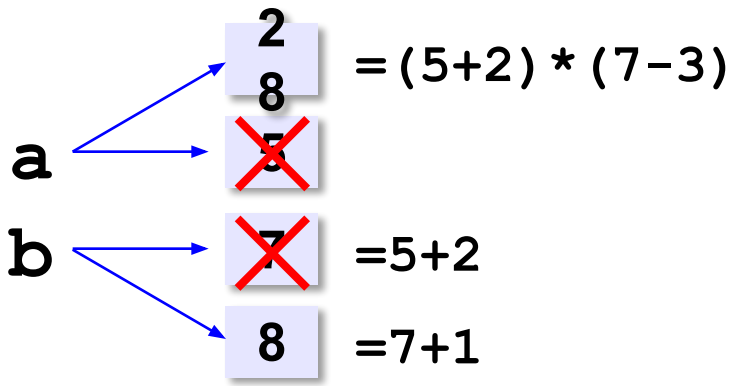
```
Введите число: 26
```

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
25 + 30 = 55
```

пользователь

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

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'>
```

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
```

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
```

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

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

Вещественные числа

Экспоненциальный формат:

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

```
3.333333e-05
1.234568e+07
```

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

$1{,}234568 \cdot 10^{7}$

```
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

```
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: 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 случайных чисел

```
from random import *
```

подключить все!

англ. *random – случайный*

Целые числа на отрезке [a,b]:

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

Генератор на [0,1):

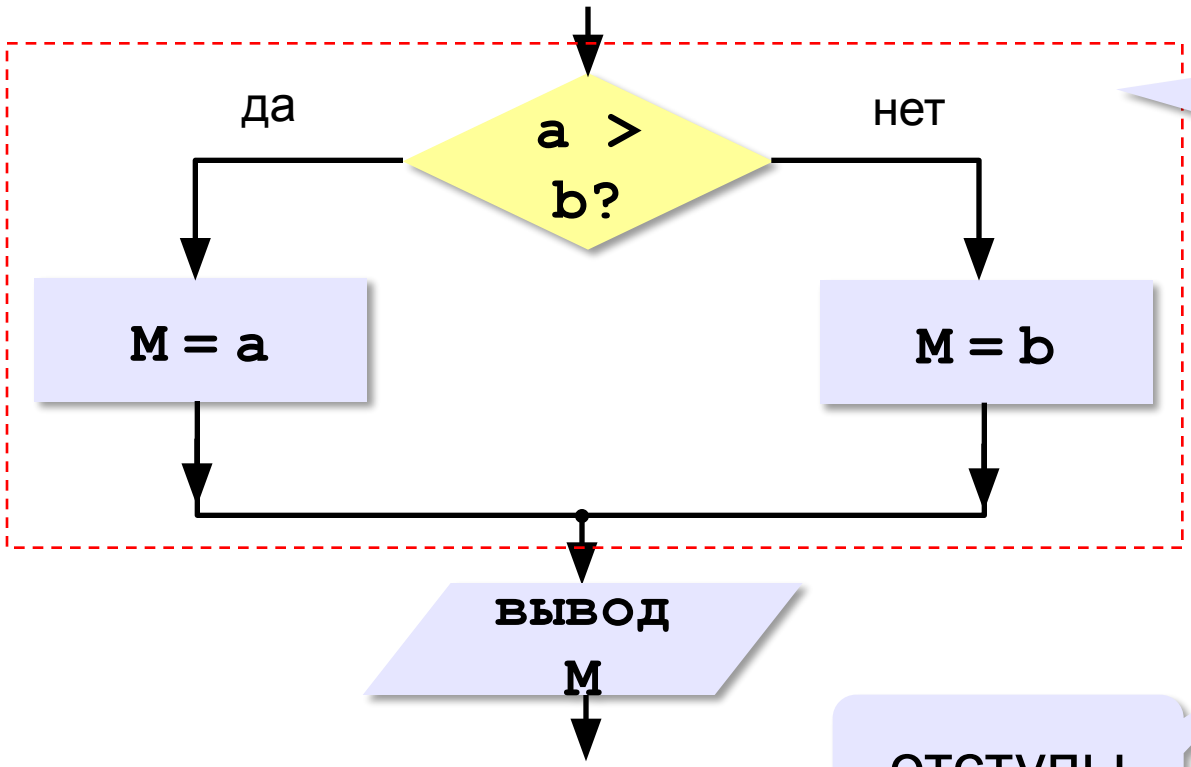
```
X = random(); # псевдослучайное число
Y = random()  # это уже другое число!
```

Программирование на языке Python

Ветвления

Условный оператор

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



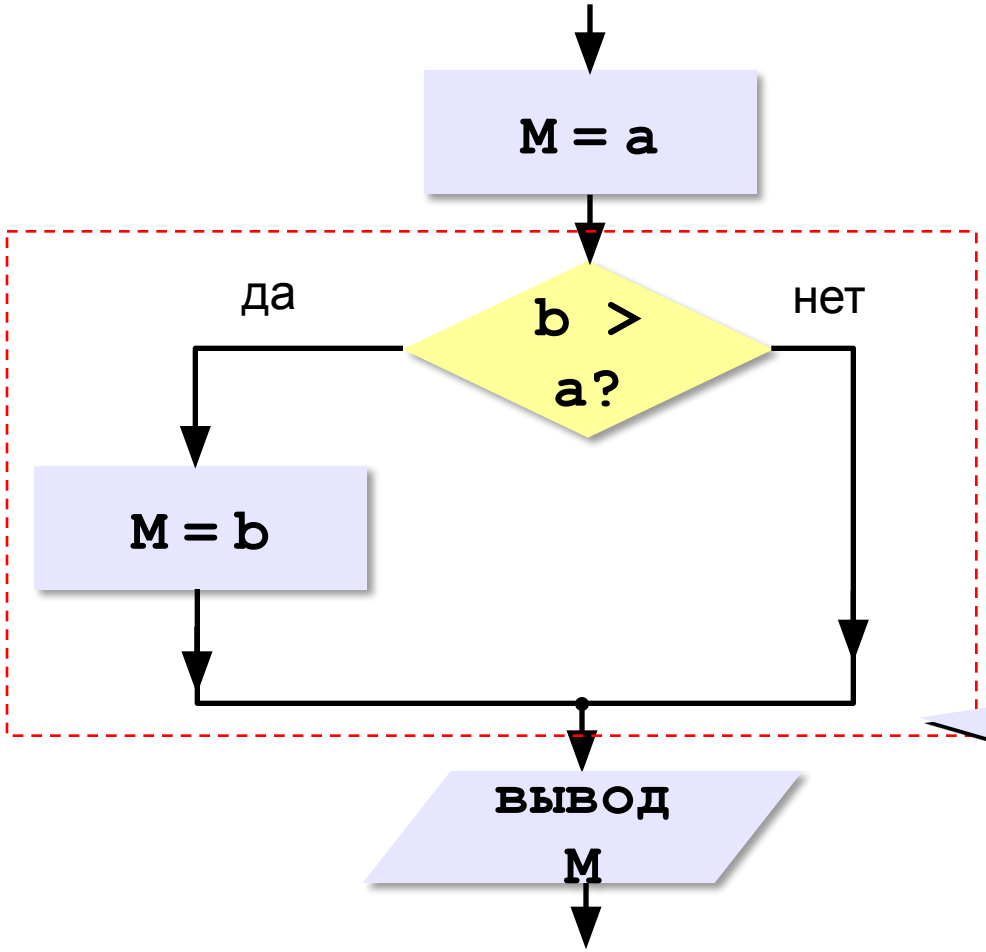
полная форма ветвления

? 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:

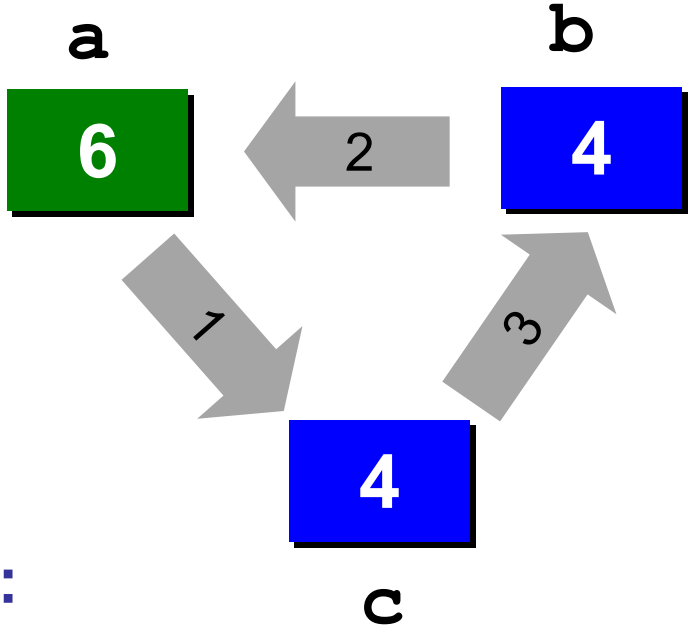
```
M = max(a, b)
```

```
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")
```

вложенный условный оператор

Зачем нужен?

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%.
```

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»)

Программирование на языке Python

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».

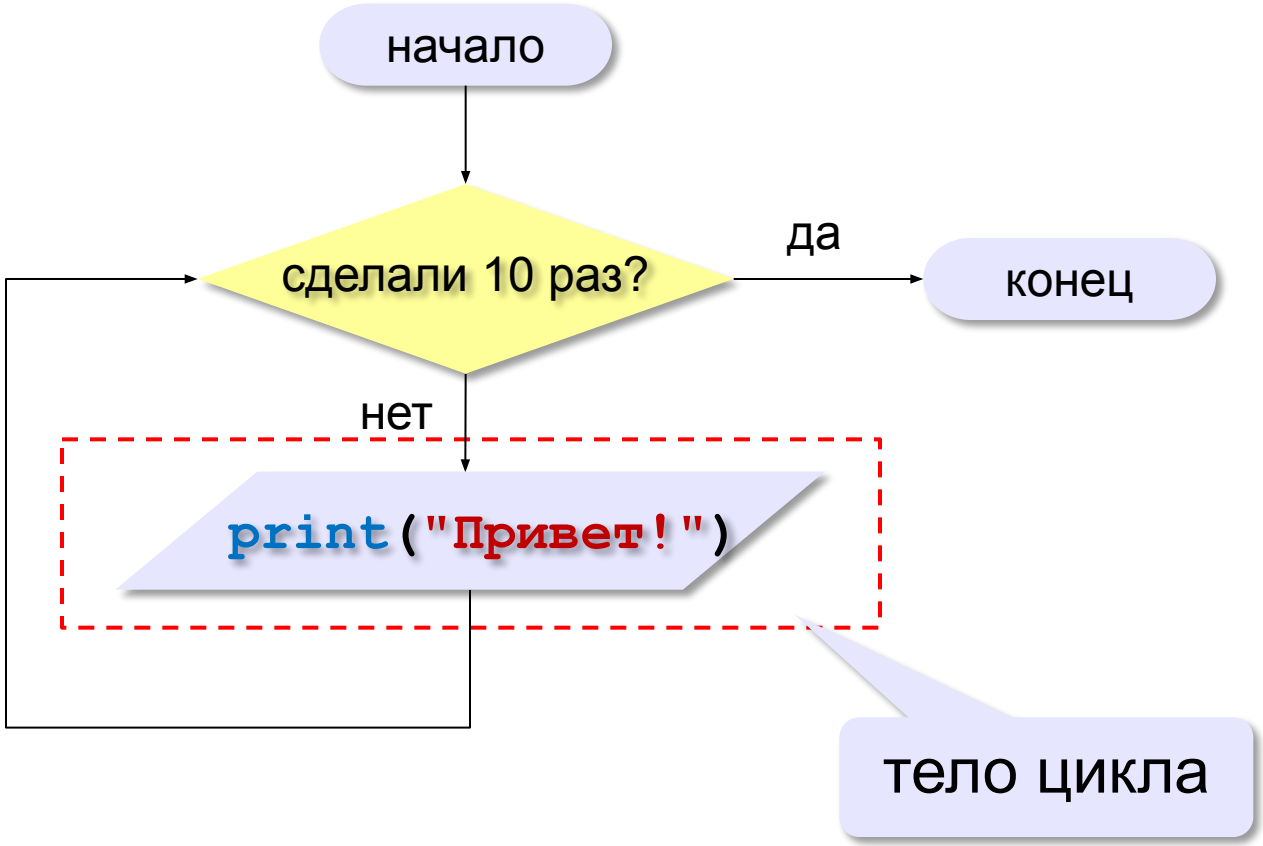
Можно ли решить известными методами?

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
```

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

результат операции автоматически сравнивается с нулём!

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?

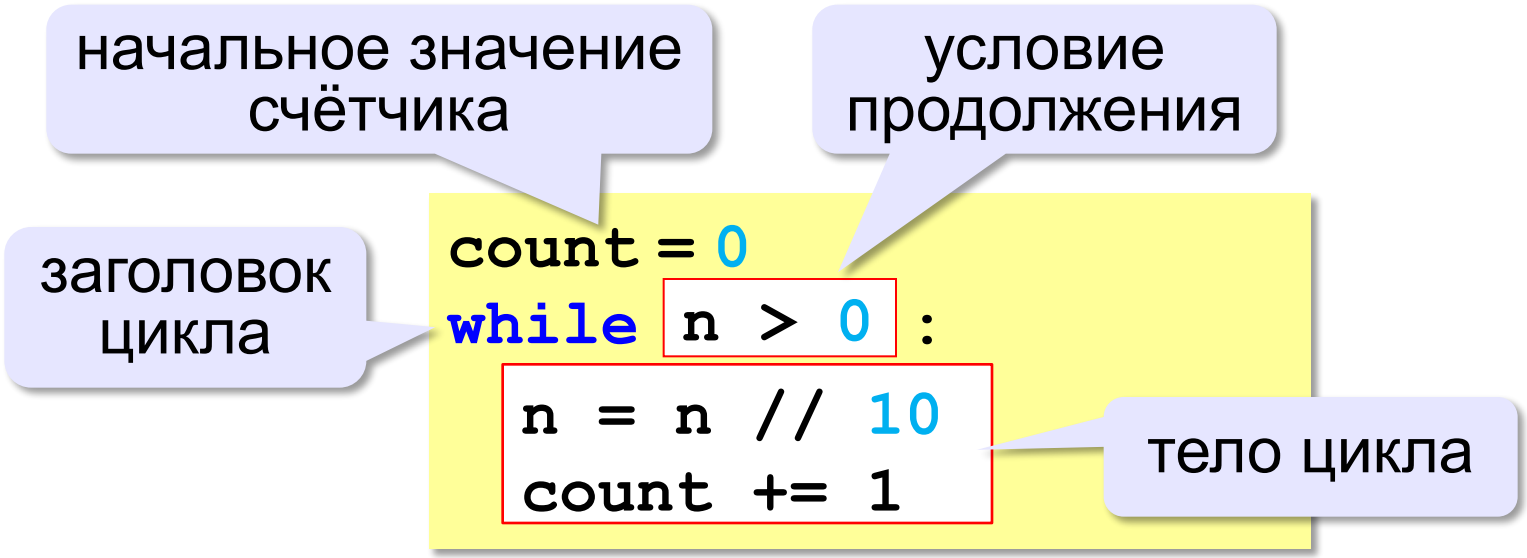
```
n = n // 10
```

? Как увеличить счётчик на 1?

```
счётчик = счётчик + 1
```

```
счётчик += 1
```

Цикл с условием



! Цикл с предусловием – проверка на входе в цикл!

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

Цикл с постусловием

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

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

бесконечный цикл

тело цикла

условие выхода

прервать цикл

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

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

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?

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

выйти из цикла

если вышли по условию

Программирование на языке Python

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

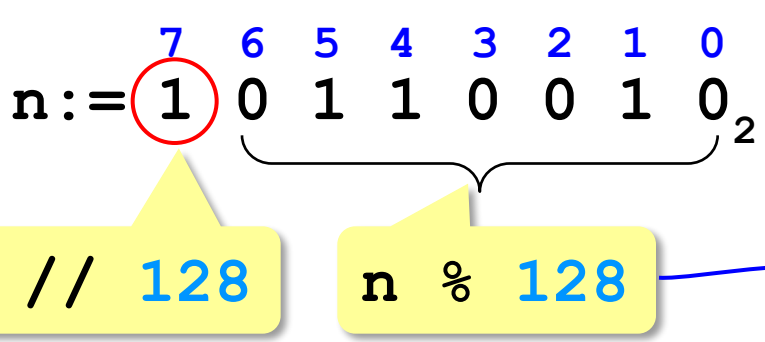
Задача. Вывести на экран запись целого числа (0..255) в 8-битном двоичном коде.

много раз!

Алгоритм:

178 ⇒ 10110010_2

? Как вывести первую цифру?



? 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 ⇒ 10110010

n	k	вывод
178	128	1

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

работаем с глобальной переменной

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

1

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 17 22	10 3 30

```
15
20
52
```

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

Программирование на языке Python

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
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?

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

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

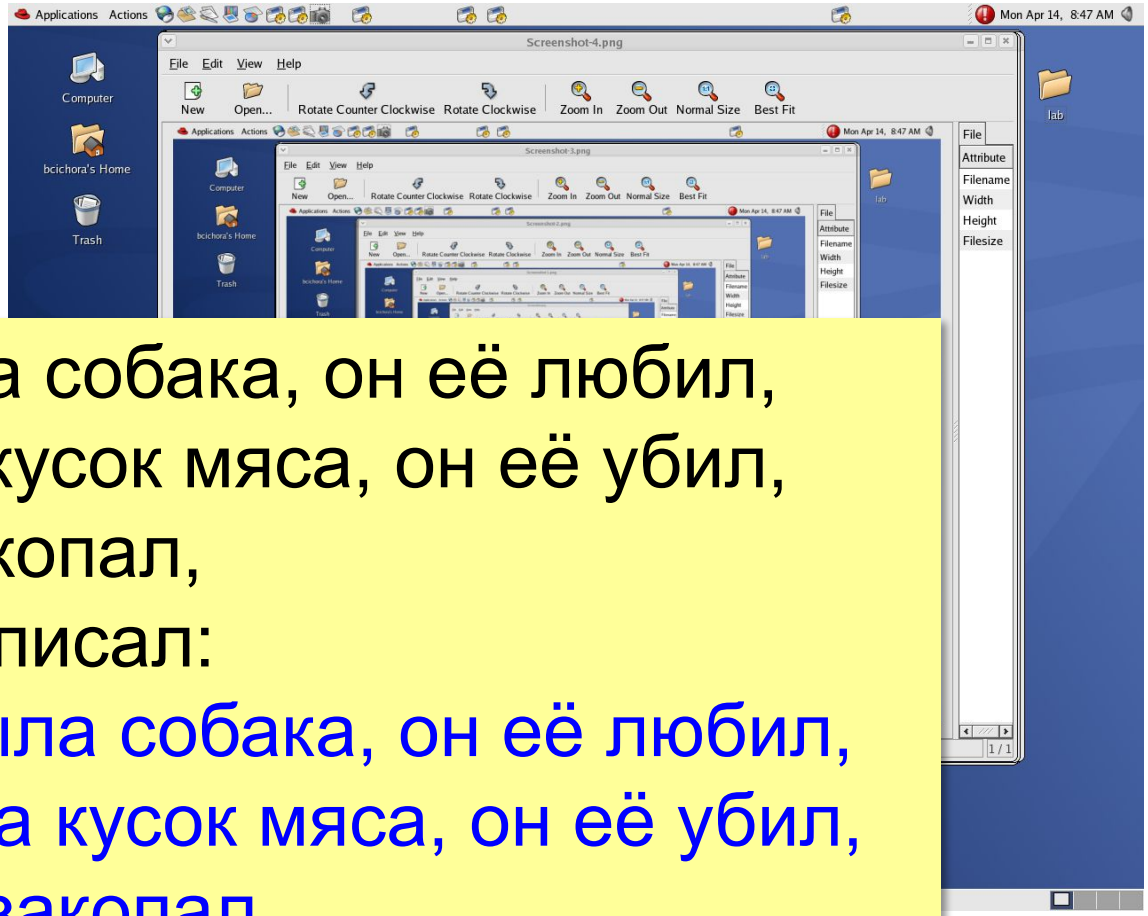
Функция, возвращающая логическое значение, может использоваться везде, где и логическая величина!

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

Программирование на языке Python

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:

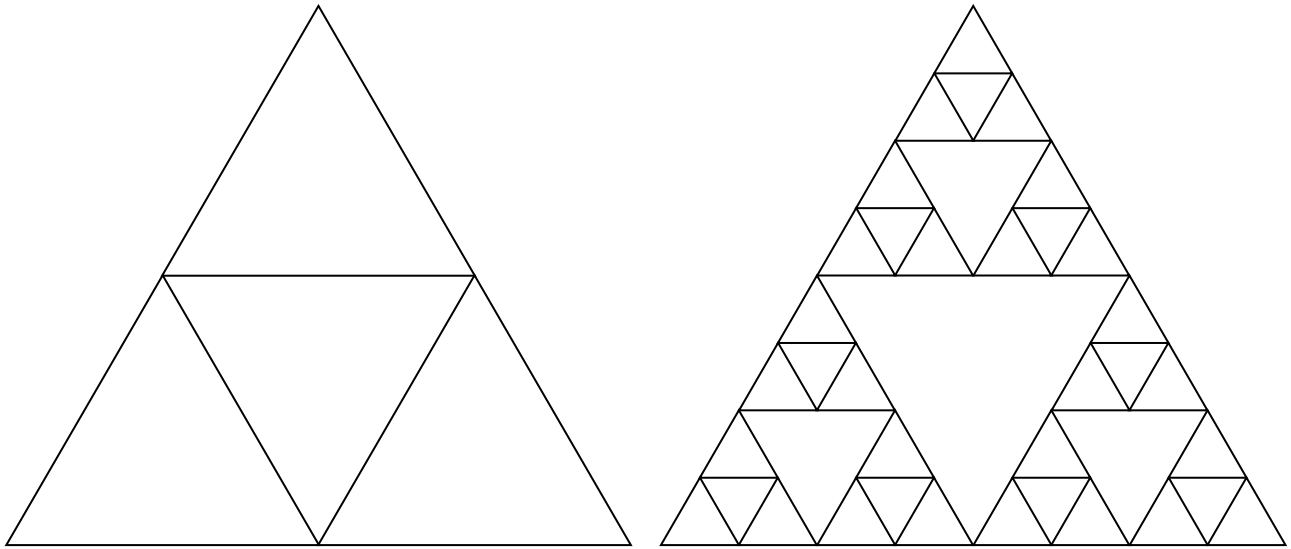
- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$ при $n > 2$

1, 1, 2, 3, 5, 8, 13, 21, 34, …

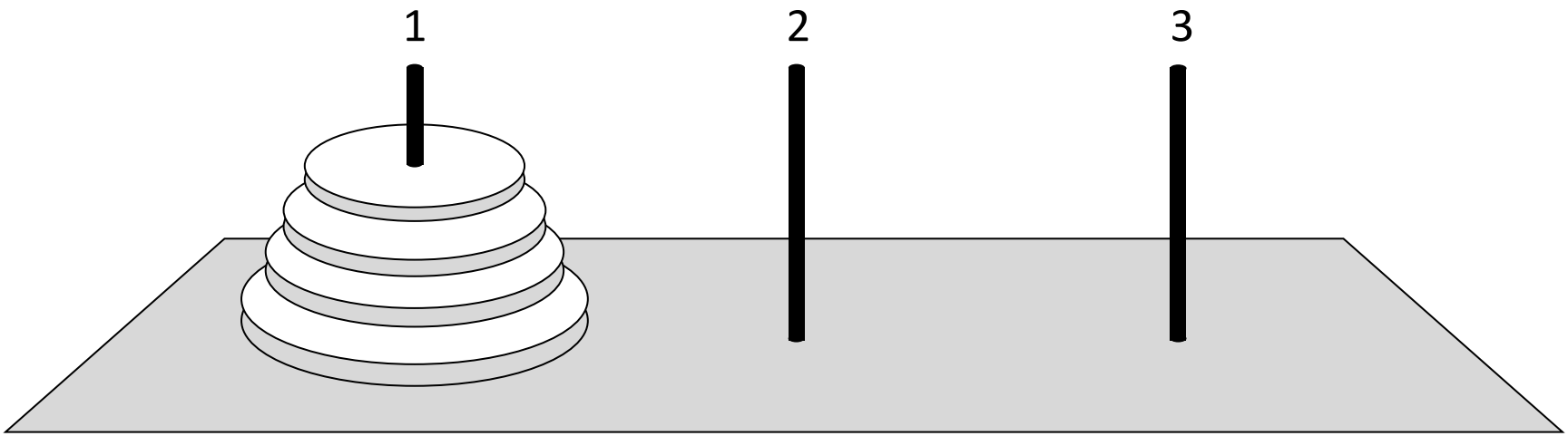
Фракталы

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 )
```

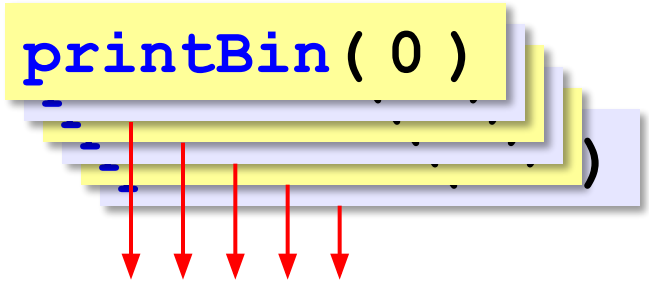
Вывод двоичного кода числа

```
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



? 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

Как работает рекурсия?

Факториал:

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

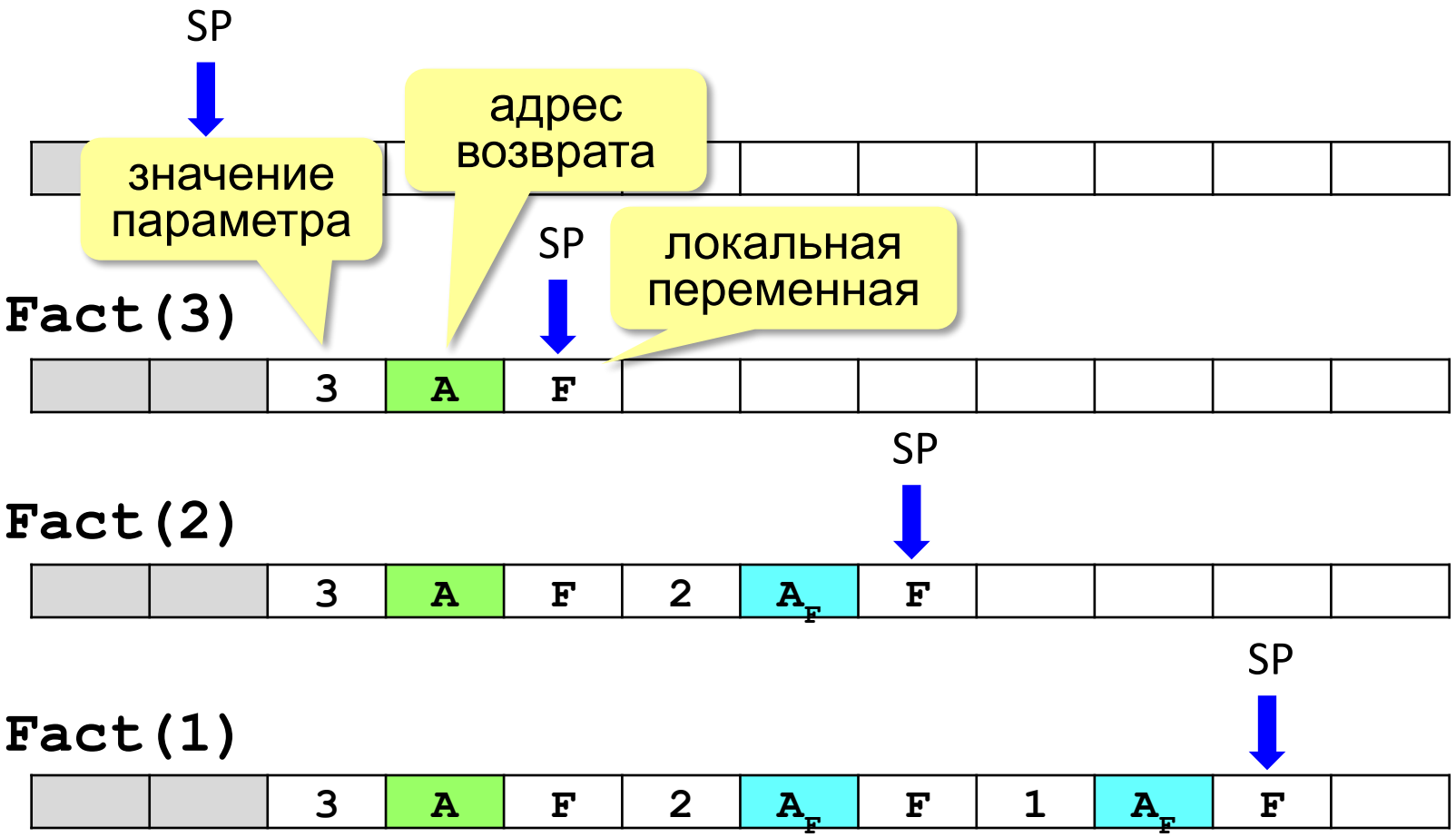
```
def Fact(N):
  print ( "->", N )
  if N <= 1: F = 1
  else:
    F = N * Fact ( N - 1 )
  print ( "<-", N )
  return F
```

```
-> 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
```