

Программирование Python

Программирование (Python)

Введение

Что такое программирование?

Программирование — это создание программ для компьютеров. Этим занимаются **программисты**.

Чем занимаются **программисты**:

анализ задачи (выделение исходных данных, связей между ними, этапов решения задачи)

системные аналитики

разработка **алгоритмов**

алгоритмисты

написание и отладка **программ**

кодировщики

тестирование программ

тестировщики

написание **документации**

технические писатели

Направления в программировании

системный программист

операционные системы,
утилиты, драйверы

прикладной программист

прикладные программы, в
т.ч. для мобильных
устройств

веб-программист

веб-сайты

программист баз данных

системы управления
базами данных

Простейшая программа

```
# Это пустая программа
```



Что делает эта программа?

комментарии после #
не обрабатываются

кодировка utf-8
по умолчанию)

```
# coding: utf-8
```

```
# Это пустая программа
```

```
"""
```

```
Это тоже комментарий
```

```
"""
```

Вывод на экран

оператор
вывода

Оператор — это команда
языка программирования.

```
print ( "Привет!" )
```

```
print ( "Привет", Вася! )
```



Что плохо?

```
print ( "Привет, Вася!" )
```

вся строка в
кавычках

Переход на новую строку

```
print ( "Привет, Вася!" )  
print ( "Привет, Петя!" )
```

Результат:

```
Привет, Вася!  
Привет, Петя!
```

переход на новую строку автоматически

Нужно в одной строке:

```
Привет, Вася!Привет, Петя!
```

Решение:

```
print ( "Привет, Вася!" , end="" )  
print ( "Привет, Петя!" )
```

после вывода данных ничего не выводить

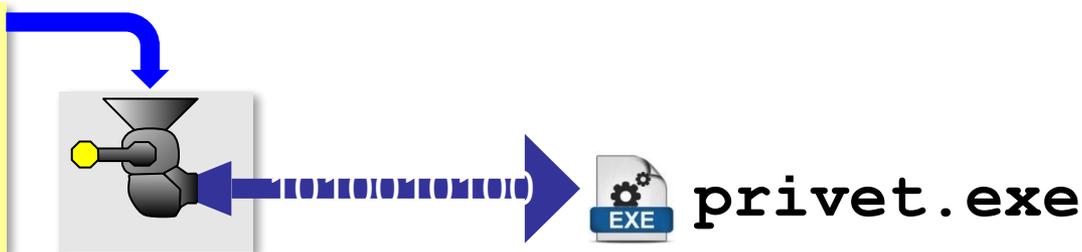
Системы программирования

Системы программирования — это средства для создания новых программ.

Транслятор — это программа, которая переводит тексты программ, написанных программистом, в машинные коды (команды процессора).

- **компилятор** — переводит всю программу в машинные коды, строит исполняемый файл (**.exe**)

```
program Hello;  
begin  
  write('Привет!')  
end.
```



- **интерпретатор** — сам выполняет программу по частям (по одному оператору).



Python – интерпретатор!

Системы программирования

Отладчик — это программа для поиска ошибок в других программах.

- **пошаговый режим** — выполнение программы по шагам (по одному оператору)
- **просмотр значений переменных** во время выполнения программы
- **точки останова** – операторы в программе, перед выполнением которых нужно остановиться.

Среда программирования (IDE):

- редактор текста программ
- транслятор
- отладчик

Программирование (Python)

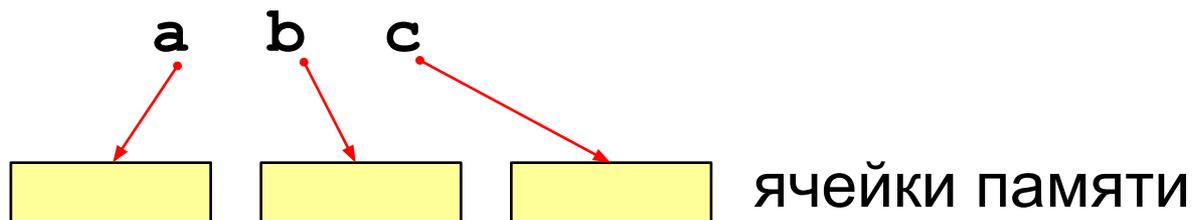
Линейные программы

Зачем нужны переменные?

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

Где запомнить?

Переменная — это величина, которая имеет имя, тип и значение. Значение переменной может изменяться во время выполнения программы.



Имена переменных

Идентификатор — это имя программы или переменной.

a b c

заглавные и строчные
буквы **различаются**

МОЖНО использовать

- латинские буквы (A-Z, a-z)
- 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

Работа с переменными

Присваивание (запись значения)

```
a = 5
```

оператор
присваивания

$a \leftarrow 5$

```
a = 5  
a = 18
```

? Что будет храниться в *a*?

Вывод на экран

```
print(a)
```

? В чём разница?

```
c = 14  
print(c)
```

14

```
c = 14  
print("c")
```

c

Работа с переменными

Изменение значения

```
i = i + 1
```

увеличить на 1

$$i \leftarrow i + 1$$

```
a = 4
b = 7
a = a + 1
b = b + 1
a = a + b
b = b + a
a = a + 2
b = b + a
```

a	b
4	
	7
5	
	8
13	
	21
15	
	36

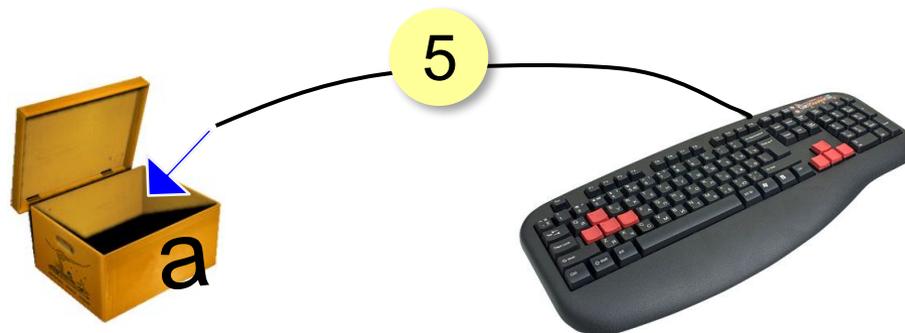
Python:

```
a, b = 4, 7
a += 1
b += 1
a += b
b += a
a += 2
b += a
```

Ввод с клавиатуры

Цель – изменить исходные данные, не меняя программу.

```
a = input()
```



1. Программа ждет, пока пользователь введет значение и нажмет *Enter*.
2. Введенное значение записывается в переменную **a**.

Ввод 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 с подсказкой

Введите число: 26

```
a = input("Введите число: ")
```



Что не так?

подсказка

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



Что будет?

преобразовать
в целое число

Введите число: Qu-Qu

ValueError: invalid literal for int() with base 10: 'Qu-Qu'

Ввод вещественных чисел

```
print( "Введите число:" )  
x = float( input() )
```

или так:

```
x = float( input("Введите число:") )
```

Программа сложения чисел

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



Что плохо?

ожидание:

Введите два числа:

5

7

5+7=12

реальность:

5

7

12



Как улучшить диалог?

Арифметические выражения

$$a \leftarrow \frac{c + b - 1}{2} \cdot d$$

Линейная запись (в одну строку):

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

Операции: + -

* – умножение

/ – деление

** – возведение в степень ($x^2 \rightarrow x**2$)

Порядок выполнения операций

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
```

Частное и остаток

// – деление нацело (остаток отбрасывается)

% – остаток от деления

175 сек = 2 мин 55 сек



Как получить 2 и 55?

```
t = 175
```

```
m = t // 60
```

```
s = t % 60
```

Частное и остаток

 Что получится?

```
n = 123
d = n // 10
k = n % 10
```

При делении на 10 нацело отбрасывается последняя цифра числа.

Остаток от деления на 10 – это последняя цифра числа.

Операторы // и %

```
a = 1234
d = a % 10; print( d )
a = a // 10
d = a % 10; print( d )
a = a // 10
d = a % 10; print( d )
a = a // 10
d = a % 10; print( d )
a = a // 10 :
```

4

3

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

Математические функции

загрузить
модуль `math`

= подключить математические
функции

```
import math
# квадратный корень
print( math.sqrt(25) )
r = 50 # радиус окружности
print( 2*math.pi*r )
print( math.pi*r**2 )
```



Что считаем?

число π

Операции с вещественными числами

$$1/3 = 0,33333\dots$$

бесконечно много знаков



Большинство вещественных чисел хранятся в памяти компьютера с ошибкой!

```
x = 1/2
y = 1/3
z = 5/6 # 5/6=1/2+1/3
print(x+y-z)
```



`-1.110223e-016`

Датчик случайных чисел

Целые числа на отрезке:

ПОДКЛЮЧИТЬ функцию `randint`
ИЗ модуля `random`

```
from random import randint
K = randint(1, 6) # отрезок [1, 6]
L = randint(1, 6) # это уже другое число!
```

англ. *integer* – целый
random – случайный



Не нужно имя модуля!

```
K = random.randint(1, 6)
```

Датчик случайных чисел

Вещественные числа:

```
from random import random, uniform
x = random()           # полуинтервал [0,1)
y = 7*random()
z = 7*random()+5
```

Вещественные числа на отрезке [a, b]:

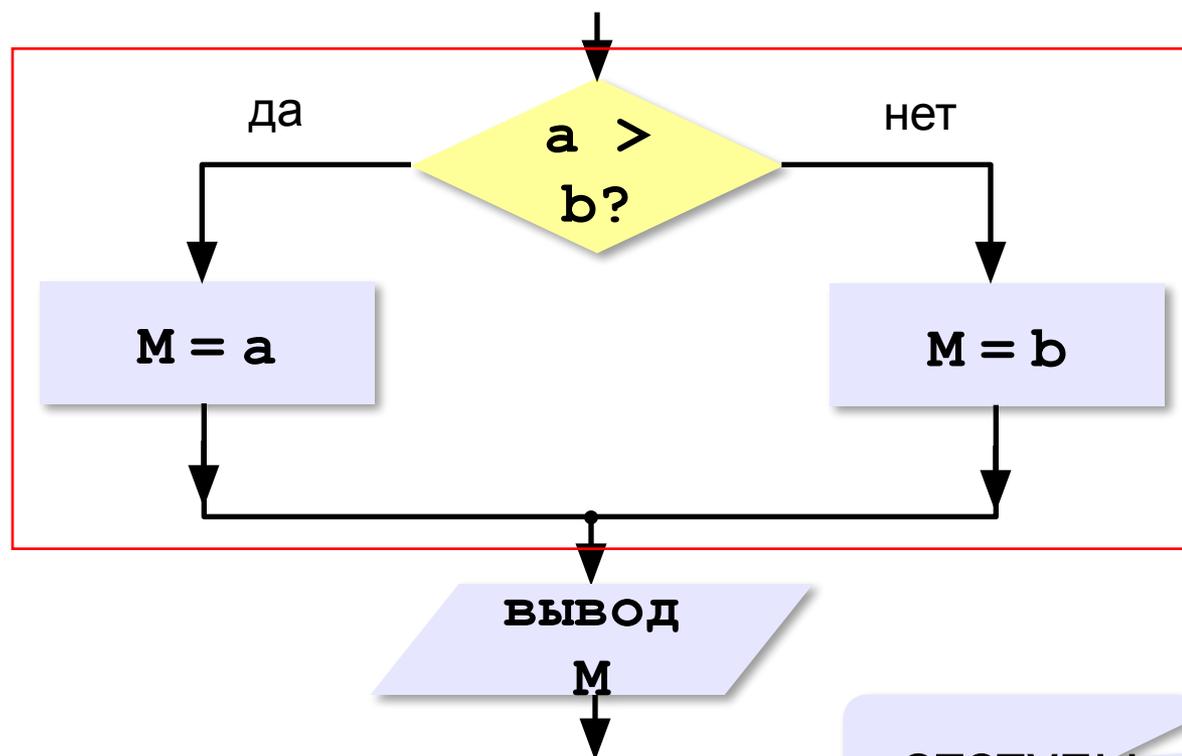
```
from random import uniform
x = uniform(1.5, 2.8)  # [1,5; 2,8]
y = uniform(5.25, 12.75) # [5,25; 12,75]
```

Программирование (Python)

§ 19. Ветвления

Выбор наибольшего из двух чисел

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

Вариант 1. Программа

```
print("Введите два целых числа")  
a = int(input())  
b = int(input())
```

```
if a > b:
```

```
    M = a
```

```
else:
```

```
    M = b
```

полная форма
условного
оператора

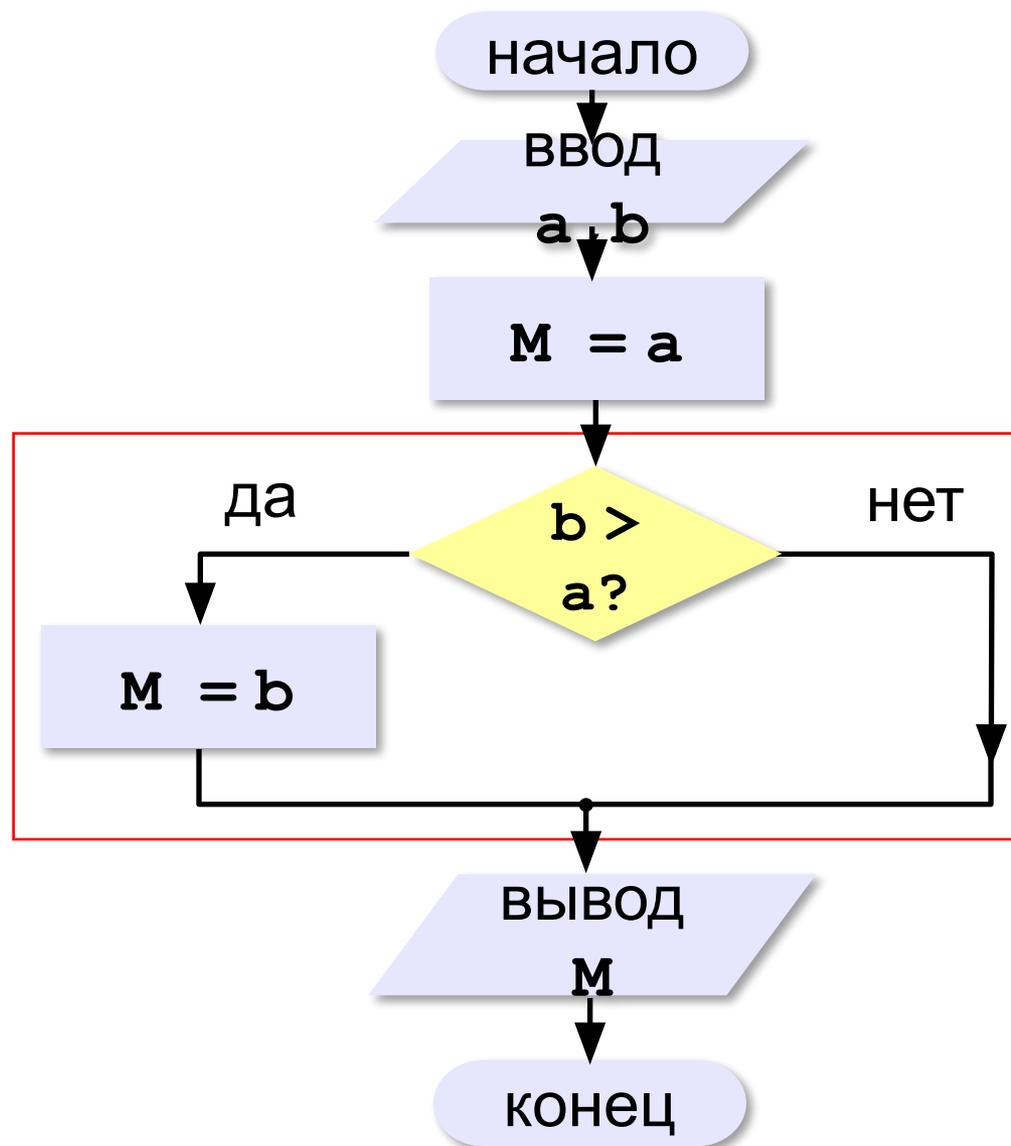
```
print("Наибольшее число", M)
```

Решение в стиле Python:

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

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

Выбор наибольшего из двух чисел-2



неполная
форма
ветвления

Вариант 2. Программа

```
print("Введите два целых числа")  
a = int(input())  
b = int(input())  
M = a  
if b > a:  
    M = b  
print("Наибольшее число", M)
```

неполная форма
условного
оператора

Примеры

Поиск минимального:

```
if a < b:  
    M = a  
if b < a:  
    M = b
```



Что плохо?



Когда работает неверно?

Примеры

```
if a < b:
```

```
    c = a
```

```
    a = b
```

```
    b = c
```



Что делает эта программа?



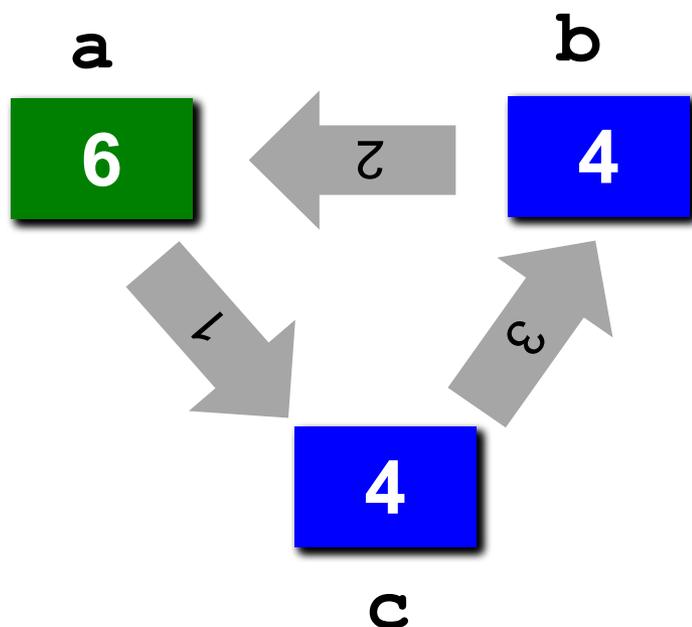
В чём отличие?

```
if a < b:
```

```
    c = a
```

```
    a = b
```

```
    b = c
```



Решение в стиле Python:

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

В других языках программирования

Паскаль:

```
if a < b then begin
    c := a;
    a := b;
    b := c;
end;
```

C:

```
if (a < b) {
    c = a;
    a = b;
    b = c;
}
```

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%.

Примеры

Задача. Вывести "Да", если число в переменной a – двузначное.

```
if 10 <= a and a <= 99:  
    print("Да")
```

Задача. Вывести "Да", если число в переменной a – двузначное и делится на 7.

```
if 10 <= a and a <= 99 and (a % 7) == 0:  
    print("Да")
```

Сложные условия: «ИЛИ»

Задача. Самолёт летает по понедельникам и четвергам.
Ввести номер дня недели и определить, летает ли в этот день самолёт.

Особенность: надо проверить, выполняется ли **одно из двух** условий:

день = 1

день = 4

```
if d == 1 or d == 4 :  
    print ("Летает")  
else:  
    print ("Не летает")
```

сложное
условие

or «ИЛИ»: выполнение **хотя бы одного**
из двух условий!

Ещё пример

Задача. Фирма набирает сотрудников от 25 до 40 лет включительно. Ввести возраст человека и определить, подходит ли он фирме (вывести ответ "подходит" или "не подходит"). Использовать «ИЛИ».

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

Сложные условия: «НЕ»

```
if not (a < b):  
    print("Старт!")
```



Как без «НЕ»?

not «НЕ»: если выполняется обратное условие

```
if a >= b:  
    print("Старт!")
```

Простые и сложные условия

Простые условия (отношения)

равно

< <= > >= == !=

не равно

Сложное условие – это условие, состоящее из нескольких простых условий (отношений), связанных с помощью логических операций:

- **and** – одновременное выполнение условий

`v >= 25 and v <= 40`

- **or** – выполнение хотя бы одного из условий

`v <= 25 or v >= 40`

- **not** – отрицание, обратное условие

`not (v > 25)` ⇔

`v <= 25`

Порядок выполнения операций

- выражения в скобках
- <, <=, >, >=, =, !=
- not
- and
- or

```
      4      1      6      2      5      3  
if not a > 2 or c != 5 and b < a:  
    ...
```

Сложные условия

Истинно или ложно

`not (a > b)`

Да

`a < b and b < c`

Да

`a > c or b > c`

Нет

`a < b and b > c`

Нет

`a > c and b > d`

Нет

`not (a >= b) or c == d`

Да

`a >= b or not (c < b)`

Да

`a > c or b > c or b > a`

Да

a

2

b

3

c

4