Программирование на языке Python

Общие сведения о языке Python

История

Python был представлен сотрудником голландского института Гвидо ван Россумом (Guido van Rossum) в 1991 году, когда он работал над распределенной ОС Амеба. Ему требовался расширяемый язык, который бы обеспечил поддержку системных вызовов. За основу были взяты ABC и Модула-3. В качестве названия он выбрал Python в честь комедийных серий BBC "Летающий цирк Монти-Питона". С тех пор Python развивался при поддержке тех организаций, в которых Гвидо работал. Особенно активно язык совершенствуется в настоящее время, когда над ним работает не только команда создателей, но и целое сообщество программистов со всего мира.

Философия языка

>>> **import this**
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
……...

Красивое лучше, чем уродливое.
Явное лучше, чем неявное.
Простое лучше, чем сложное.
Сложное лучше, чем запутанное.
Плоское лучше, чем вложенное.
Разреженное лучше, чем плотное.
Читаемость имеет значение.
Особые случаи не настолько особые, чтобы нарушать правила.
……..

Простейшая программа

```
# Это пустая 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

«B»: Вывести на экран текст «лесенкой»

```
Вася
    пошел
        гулять
```

«C»: Вывести на экран рисунок из букв

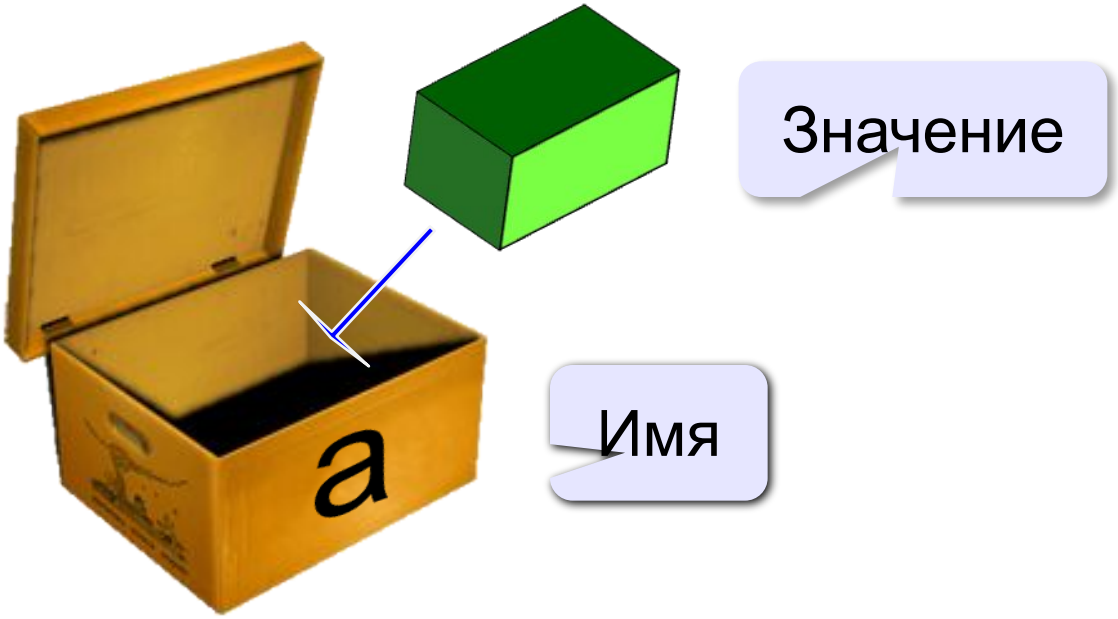
```
   Ж
  ЖЖЖ
 ЖЖЖЖЖ
ЖЖЖЖЖЖЖ
 НН  НН
 ZZZZZ
```

Программирование на языке Python

Переменные

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.~~

Типы переменных

- **Числа**

- Для хранения целых чисел Python в отличие от большинства языков использует всю доступную память.
- Вещественные числа реализованы на основе чисел с плавающей точкой двойной точности — double (64 бита). 1 бит на знак, 11 бит на показатель экспоненты и 52 бита на значащую часть (мантиссу).

Примеры: 3.0, -123.345, .76543, 23.490e23.

- **Логические: True, False.**

Логический тип на самом деле является лишь подтипом целого, значение False соответствует нулю, True — любому ненулевому целому числу.

Типы переменных

- ***Упорядоченные последовательности***

— *строки:* последовательность литералов (символов). Строковые значения должны быть заключены в одинарные и двойные кавычки.
Примеры: 'a', ‘фис', '234g 3654___', "dont".

— *списки:* последовательность произвольных элементов, разделяемых запятыми и взятая в квадратные скобки. Пустой список— [ ].
Примеры: [1, 2, 3], ['Name', 'Surname', Age].

— *кортежи:* последовательность произвольных элементов, разделяемых запятыми, которая может быть взята в круглые скобки. Пустой кортеж обязательно должен быть взят в скобки: (), кортеж из одного элемента обязательно должен содержать запятую после единственного элемента (4,).
Примеры: (2, 3) , (‘abc', 345)

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'>
```

Преобразование типов переменных

```
int(n)- преобразует в целое
float(n)- преобразует в десятичное
str(n)- преобразует в символьное
```

```
a = 4.5
print(str(a))
```

> > > “4.5”

строка

Тип определяет:

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

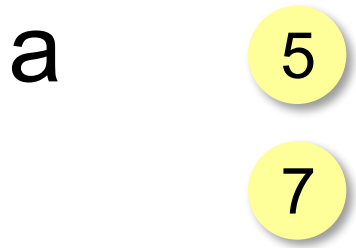
запись значения в переменную?

оператор присваивания

При записи нового значения старое удаляется из памяти!

```
a = 5
```

```
a = 7
```

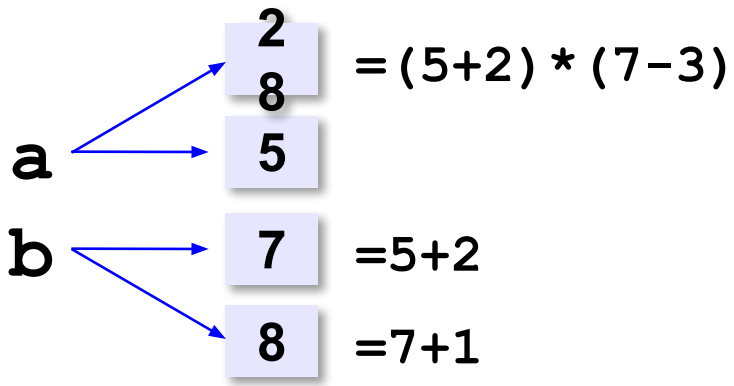


Оператор – это команда языка программирования (инструкция).

Оператор присваивания – это команда для записи нового значения переменной.

Изменение значений переменной

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



Программирование на языке Python

Ввод данных с клавиатуры и простейшие операторы

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 с подсказкой

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

```
Введите число: 26
```

подсказка

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

– для python 3.x обязательно!!!

Сложение чисел: простое решение

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

Сложение чисел: полное решение

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

подсказка

компьютер

Протокол:

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

пользователь

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 = 9.0; b = 6.0
x = 3.0 / 4     # = 0.75
x = a / b       # = 1.5
x = -3.0 / 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
```

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

Программирование на языке Python

Вывод данных.
Формат вывода.

Вывод данных

```
print ( a )
```

значение переменной

```
print ( "Ответ: ", a )
```

значение и текст

перечисление через запятую

```
print ( "Ответ: ", a+b )
```

вычисление выражения

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

```
2 + 3 = 5
```

через пробелы

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

```
2+3=5
```

убрать разделители (для Python 3)

Целые числа

%d — целое

```
>>> n = 10.57
>>> print "%d" % (n)
10
```

%5d — целое, записанное в поле из пяти символов (в данном случае перед цифрой стоит три пробела)

```
>>> print "%5d" % (n)
   10
```

%-5d — целое, записанное в поле из пяти символов, но выравненное влево (там справа ещё три пробела)

```
>>> print "%-5d" % (n)
10
```

%05d — целое, записанное в поле из пяти символов, с нулями впереди

```
>>> print "%05d" % (n)
00010
```

Вещественные числа

%5.1f — вещественное с плавающей запятой, с одним знаком после запятой, шириной в поле шириной пять знаков (в данном случае один пробел перед цифрой и округление)

```
>>> print "%5.1f" % (n)
 10.6
```

%.3f — вещественное с плавающей запятой, с тремя знаками после запятой, в поле минимально необходимой ширины

```
>>> print "%.3f" % (n)
10.570
```

Вещественные числа

%e — вещественное с плавающей запятой, записанное в научной нотации

```
>>> print "%e" % (n)
1.057000e+01
```

%E — то же что *%e*, но для обозначения экспоненты используется заглавная *E*

```
>>> print "%E" % (n)
1.057000E+01
```

%11.3e — вещественное с плавающей запятой, записанное в научной нотации, с 3мя знаками после запятой в поле из 11ти знаков (в данном случае перед числом 2 пробела)

```
>>> print "%11.3e" % (n)
  1.057e+01
```

%.3e — вещественное с плавающей запятой, записанное в научной нотации, в поле минимально необходимой ширины

```
>>> print "%.3e" % (n)
1.057e+01
```

Строки

%s — строка. Если аргумент не строка, то производится автоматическое конвертирование его в строку путём str(n)

```
>>> print "%s" % (n)
10.57
```

%20s — строка из 20 символов, цифра выравнена вправо.

```
>>> print "%20s" % (n)
               10.57
```

%-20s — строка из 20 символов, цифра выравнена влево.

```
>>> print "%-20s" % (n)
10.57
```

Стандартные функции

`abs(x)` – модуль числа
`int(x)` – преобразование к целому числу

```
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
```

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!
```

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:

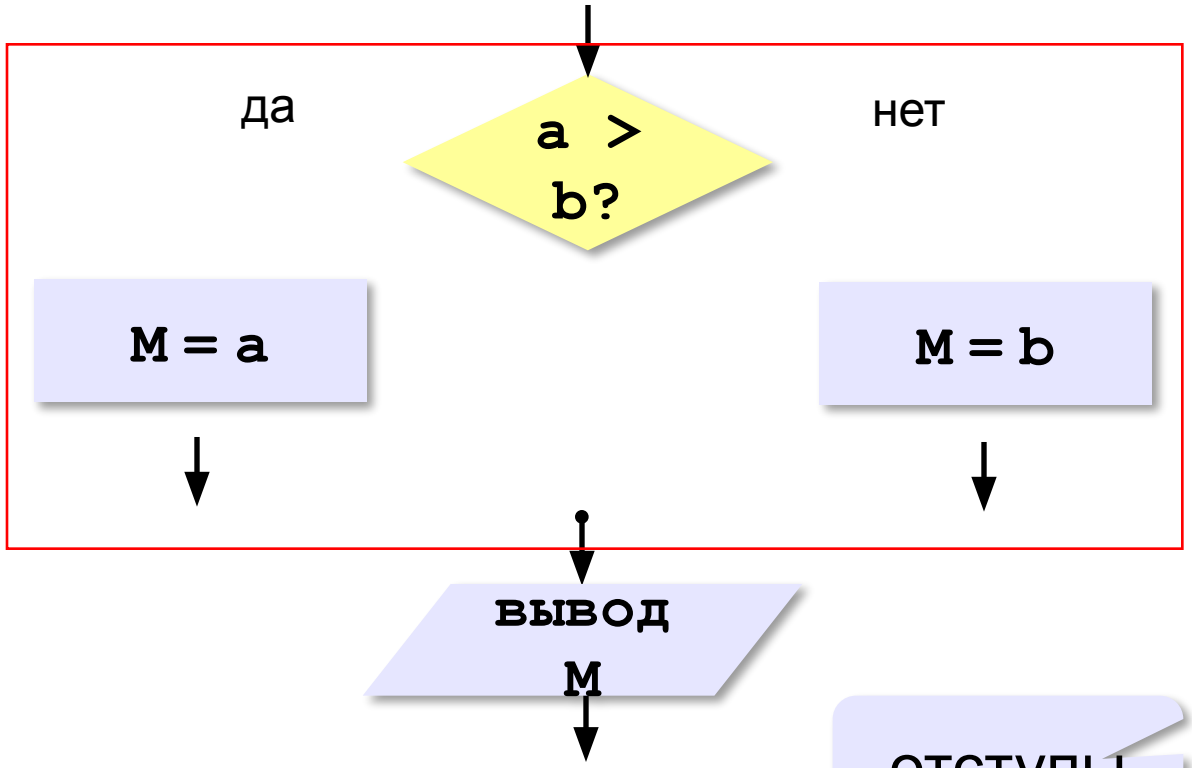
```
Получено число 123.
Его цифры 1, 2, 3.
```

Программирование на языке Python

Условные операторы

Условный оператор

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



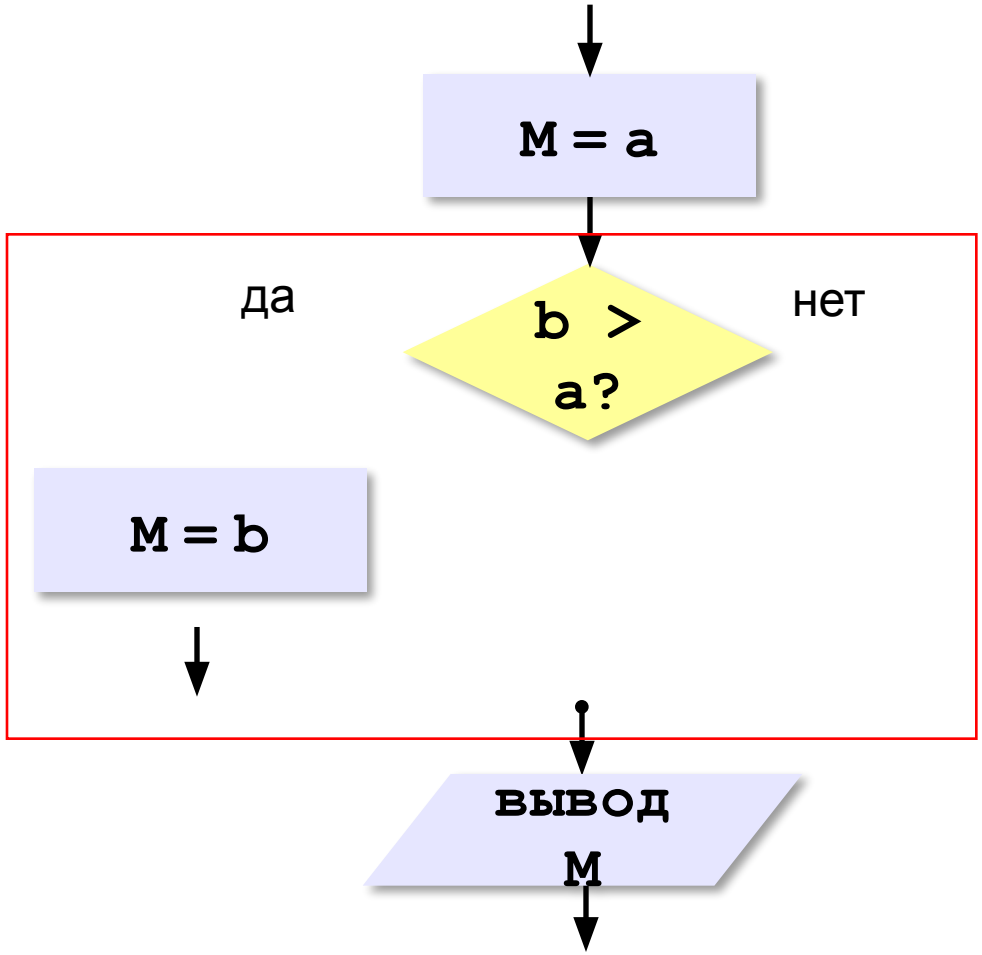
полная форма ветвления

? Если a = b?

отступы

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

Условный оператор: неполная форма




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

неполная форма ветвления

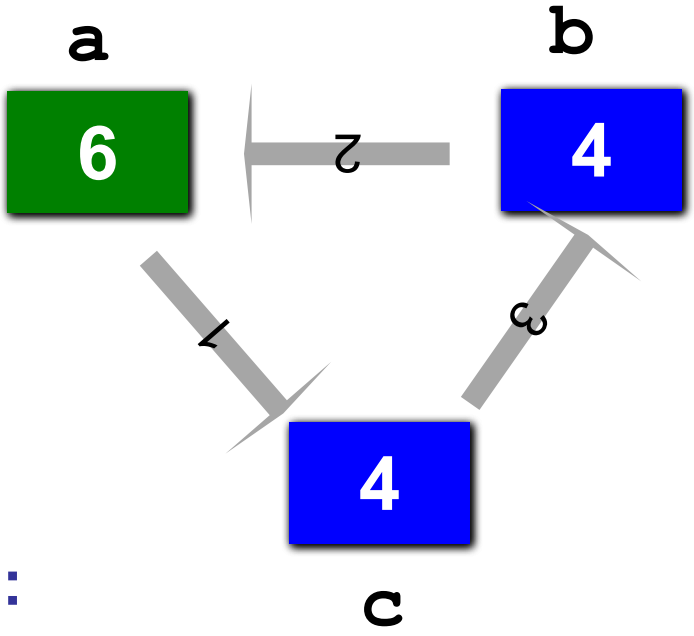
Решение в стиле Python:

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

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

Условный оператор

```
if a > b:
   c = a
   a = b
   b = c
```



? 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?

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

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?

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

«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

«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.
```

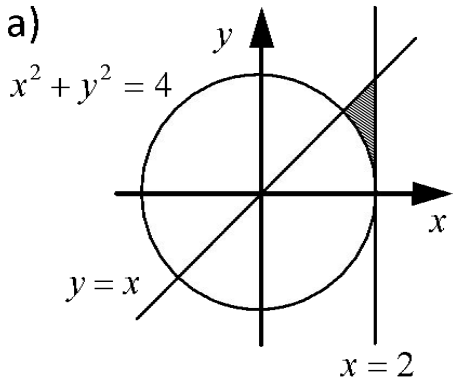
Пример:

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

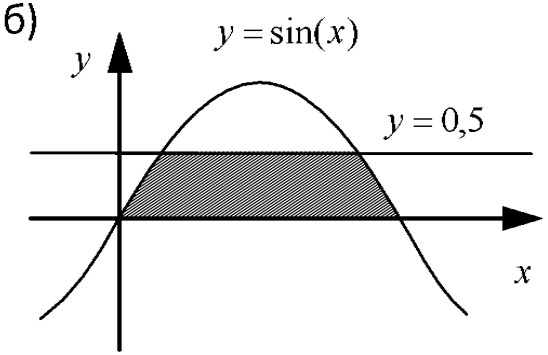
Задачи

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

а)

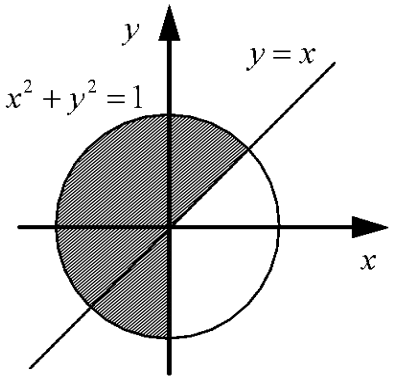


б)

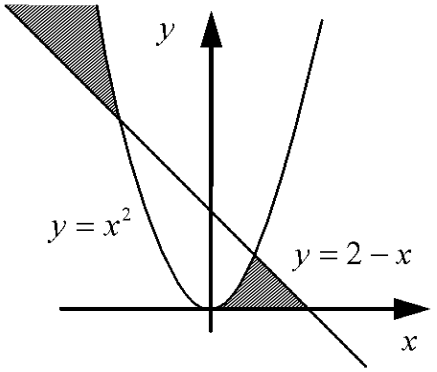


«В»: Напишите условие, которое определяет заштрихованную область.

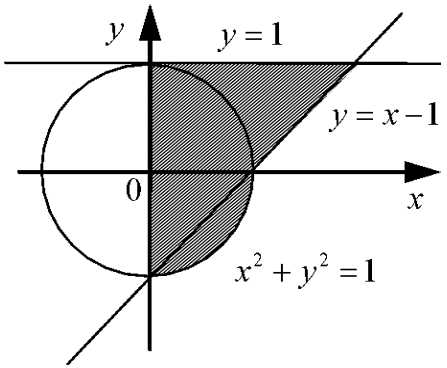
а)



б)

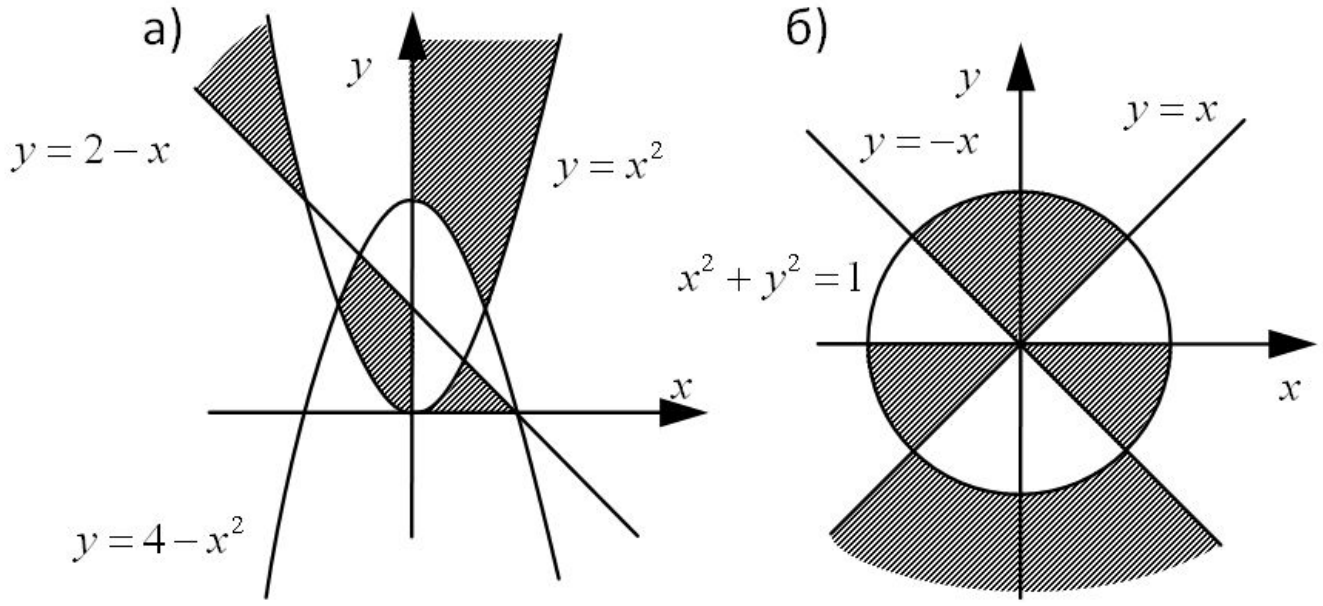


в)



Задачи

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



Программирование на языке Python

Циклические алгоритмы

Что такое цикл?

Цикл – это многократное выполнение одинаковых действий.

Два вида циклов:

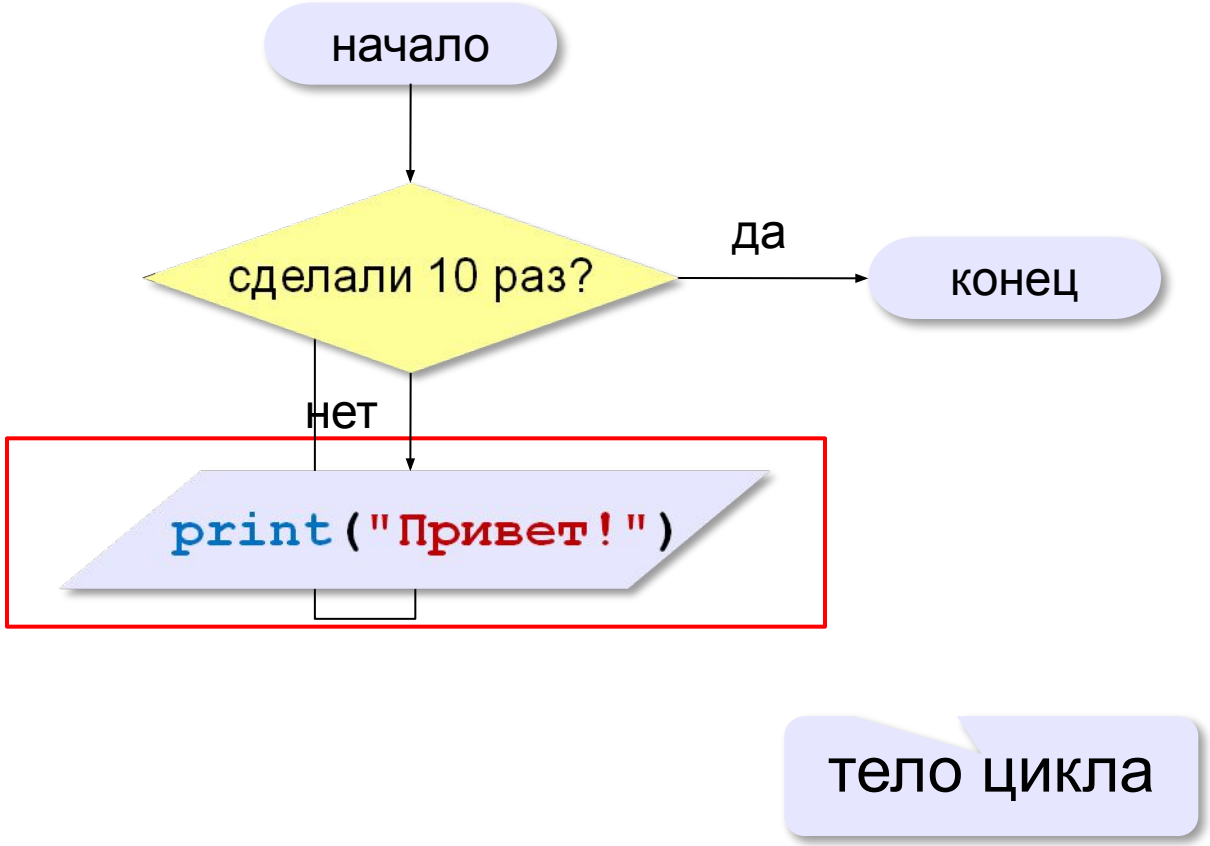
- цикл с **известным** числом шагов (сделать 10 раз)
- цикл с **неизвестным** числом шагов (делать, пока не надоест)

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
```

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**.

бесконечный цикл

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

тело цикла

условие выхода

прервать цикл

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

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-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.
```

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 = 0
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

«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?

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

```
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

Символьные строки

Символьные строки

Начальное значение:

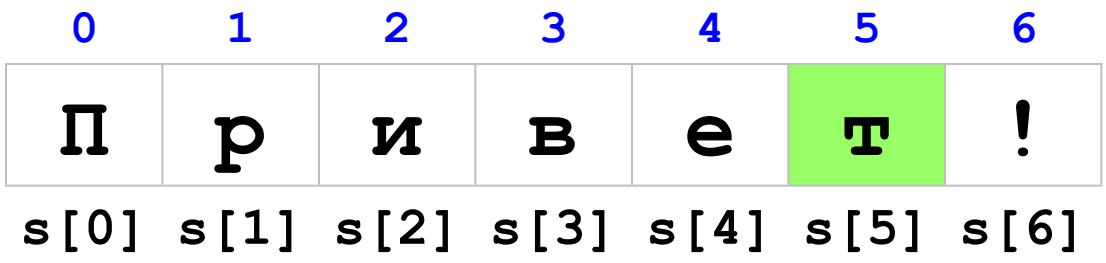
```
s = "Привет!"
```

Вывод на экран:

```
print ( s )
```

```
print ( s[5] )
```

```
print ( s[-2] )
```



s[len(s)-2]

Длина строки:

```
n = len ( s )
```

Строка – это последовательность символов!

Символьные строки

Ввод с клавиатуры:

```
s = input ( "Введите имя: " )
```

Изменение строки:

```
s[4] = "a"
```

Строка – это неизменяемый объект!

... но можно составить новую строку:

```
s1 = s + "a"
```

Символьные строки

Задача: заменить в строке все буквы "а" на буквы "б".

```
s = input( "Введите строку:" )
s1 = ""    # строка-результат
for c in s:
    if c == "а":
        c = "б"
    s1 = s1 + c
print( s1 )
```

перебрать все символы в строке

добавить символ к строке-результату

Задачи

«А»: Ввести с клавиатуры символьную строку и заменить в ней все буквы «а» на «б» и все буквы «б» на «а» (заглавные на заглавные, строчные на строчные).

Пример:

```
Введите строку:
ааббААББссСС
Результат:
ббааББААссСС
```

Задачи

«В»: Ввести с клавиатуры символьную строку и определить, сколько в ней слов. Словом считается последовательности непробельных символов, отделенная с двух сторон пробелами (или стоящая с краю строки). Слова могут быть разделены несколькими пробелами, в начале и в конце строки тоже могут быть пробелы.

Пример:

```
Введите строку:
  Вася  пошел      гулять
Найдено слов: 3
```

Задачи

«С»: Ввести с клавиатуры символьную строку и найдите самое длинное слово и его длину. Словом считается последовательности непробельных символов, отделенная с двух сторон пробелами (или стоящая с краю строки). Слова могут быть разделены несколькими пробелами, в начале и в конце строки тоже могут быть пробелы.

Пример:

```
Введите строку:
  Вася   пошел  гулять
Самое длинное слово: гулять, длина 6
```

Операции со строками

Методы строк.

Формат: ***строка.метод(параметры)***

Например:

S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.replace(str1, str2)	Замена строки str1 на строку str2
S.count(str, [start],[end])	Возвращает количество непересекающихся повторений подстроки в диапазоне [начало, конец]
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError

Операции со строками

Объединение *(конкатенация)* :

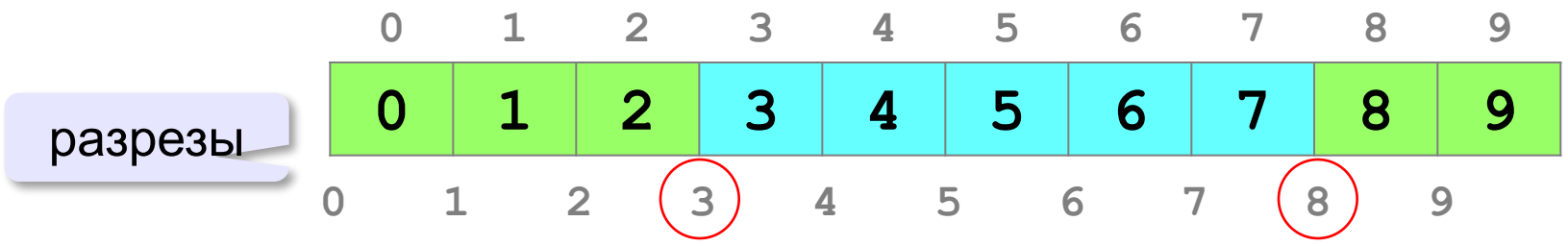
```
s1 = "Привет"
s2 = "Вася"
s  = s1 + ", " + s2 + "!"
```

"Привет, Вася!"

Срезы:

Срез – это механизм гибкого управления строкой на основе индексации

```
s = "0123456789"
s1 = s[3:8]              # "34567"
```



Операции со строками

Срезы:

```
s = "0123456789"
s1 = s[:8]                  # "01234567"
```

от начала строки

```
s = "0123456789"
s1 = s[3:]                  # "3456789"
```

до конца строки

```
s1 = s[::-1]                # "9876543210"
```

реверс строки

Операции со строками

Срезы с отрицательными индексами:

```
s = "0123456789"
s1 = s[:-2]              # "01234567"
```

N-2

```
s = "0123456789"
s1 = s[-6:-2]            # "4567"
```

N-6 N-2

Операции со строками

Удаление:

```
s = "0123456789"
s1 = s[:3] + s[9:]   # "0129"
```

"012" "9"

Вставка:

```
s = "0123456789"
s1 = s[:3] + "ABC" + s[3:]
```

"012ABC3456789"

Стандартные функции

Верхний/нижний регистр:

```
s = "aAbBcC"
s1 = s.upper()   # "AABBCC"
s2 = s.lower()   # "aabbcc"
```

Проверка на цифры:

```
s = "abc"
print ( s.isdigit() )   # False
s1 = "123"
print ( s1.isdigit() )   # True
```

… и много других.

Поиск в строках

```
s = "Здесь был Вася."
n = s.find ( "с" )    # n = 3
if n >= 0:
  print ( "Номер символа", n )
else:
  print ( "Символ не найден." )
```

Находит первое слева вхождение подстроки!

Поиск с конца строки:

```
s = "Здесь был Вася."
n = s.rfind ( "с" )     # n = 12
```

Пример обработки строк

Задача: Ввести имя, отчество и фамилию. Преобразовать их к формату «фамилия-инициалы».

Пример:

```
Введите имя, отчество и фамилию:
Василий Алибабаевич Хрюндиков
Результат:
Хрюндиков В.А.
```

Алгоритм:

- найти первый пробел и выделить имя
- удалить имя с пробелом из основной строки
- найти первый пробел и выделить отчество
- удалить отчество с пробелом из основной строки
- «сцепить» фамилию, первые буквы имени и фамилии, точки, пробелы…

Пример обработки строк

```
print ( "Введите имя, отчество и фамилию:" )
s = input()
n = s.find ( " " )
name = s[:n]     # вырезать имя
s = s[n+1:]
n = s.find ( " " )
name2 = s[:n]      # вырезать отчество
s = s[n+1:]        # осталась фамилия
s = s + " " + name[0] + "." + name2[0] + "."
print ( s )
```

Пример обработки строк

Решение в стиле Python:

```
print ( "Введите имя, отчество и фамилию:" )
s = input()
fio = s.split()
s = fio[2] + " " + fio[0][0] + "." + fio[1][0] + "."
print ( s )
```

```
Василий Алибабаевич Хрюндиков
 fio[0]    fio[1]     fio[2]
```

Задачи

«А»: Ввести с клавиатуры в одну строку фамилию, имя и отчество, разделив их пробелом. Вывести фамилию и инициалы.

Пример:

```
Введите фамилию, имя и отчество:
Иванов Петр Семёнович
П.С. Иванов
```

Задачи

«B»: Ввести адрес файла и «разобрать» его на части, разделенные знаком "/". Каждую часть вывести в отдельной строке.

Пример:

```
Введите адрес файла:
C:/Фото/2013/Поход/vasya.jpg
C:
Фото
2013
Поход
vasya.jpg
```

Задачи

«С»: Напишите программу, которая заменяет во всей строке одну последовательность символов на другую.

Пример:

```
Введите строку:
(X > 0) and (Y < X) and (Z > Y) and (Z <> 5)
Что меняем: and
Чем заменить: &
Результат
(X > 0) & (Y < X) & (Z > Y) & (Z <> 5)
```

Преобразования «строка» – «число»

Из строки в число:

```
s = "123"
N = int ( s )          # N = 123
s = "123.456"
X = float ( s )        # X = 123.456
```

Из числа в строку:

```
N = 123
s = str ( N )                 # s = "123"
s = "{:5d}".format(N)         # s = "  123"

X = 123.456
s = str ( X )                 # s = "123.456"
s = "{:7.2f}".format(X)       # s = " 123.46"
s = "{:10.2e}".format(X)      # s = "  1.23e+02"
```

Задачи

«A»: Напишите программу, которая вычисляет сумму трех чисел, введенную в форме символьной строки. Все числа целые.

Пример:

```
Введите выражение:
12+3+45
Ответ: 60
```

«B»: Напишите программу, которая вычисляет выражение, состоящее из трех чисел и двух знаков (допускаются только знаки «+» или «–»). Выражение вводится как символьная строка, все числа целые.

Пример:

```
Введите выражение:
12-3+45
Ответ: 54
```

Задачи

«С»: Напишите программу, которая вычисляет выражение, состоящее из трех чисел и двух знаков (допускаются знаки «+», «–», «*» и «/»). Выражение вводится как символьная строка, все числа целые. Операция «/» выполняется как целочисленное деление.

Пример:

```
Введите выражение:
12*3+45
Ответ: 81
```

Задачи

«D»: Напишите программу, которая вычисляет выражение, состоящее из трех чисел и двух знаков (допускаются знаки «**+**», «**–**», «*****» и «**/**») **и круглых скобок**. Выражение вводится как символьная строка, все числа целые. Операция «**/**» выполняется как целочисленное деление (**div**).

Пример:

```
Введите выражение:
2*(3+45)+4
Ответ: 100
```

Программирование на языке Python

Списки

Списки

```
students = ['Ivan', 'Masha', 'Sasha']
for student in students:
    print("Hello, " + student + "!")
```

```
Hello, Ivan!
Hello, Masha!
Hello, Sasha!
```

students[0]	'Ivan'
students[1]	'Masha'
students[2]	'Sasha'

Изменение списков

В отличии от изученных типов данных (**int**, **float**, **str**) списки (**list**) являются изменяемыми.

Можно изменить конкретный элемент списка:

```
students = ['Ivan', 'Masha', 'Sasha']
students[1] = 'Oleg'
print(students)
```

```
['Ivan', 'Oleg', 'Sasha']
```

Доступ к элементам списка

```
students = ['Ivan', 'Masha', 'Sasha']
```

Длина списка: `len(students)`

Результат: 3

`students[0]`	`'Ivan'`	`students[-1]`	`'Sasha'`
`students[1]`	`'Masha'`	`students[-2]`	`'Masha'`
`students[2]`	`'Sasha'`	`students[-3]`	`'Ivan'`

`students[:2]`

`students[::-1]`

Операции со списками

+

```
students = ['Ivan', 'Masha', 'Sasha']
teachers = ['Oleg', 'Alex']
students + teachers
```

Результат: ['Ivan', 'Masha', 'Sasha', 'Oleg', 'Alex']

*

```
[0, 1] * 4
```

Результат: [0, 1, 0, 1, 0, 1, 0, 1]

Операции со списками

```
A = [1, 3, 4, 23, 5]
```

```
A = [1, 3] + [4, 23] + [5]
```

```
[1, 3, 4, 23, 5]
```

```
A = [0]*10
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
A = list( range(10) )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

методы списков

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.pop(i)	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort()	Сортирует список
list.reverse()	Разворачивает список
list.clear()	Очищает список

Добавление элементов в список

```
students = ['Ivan', 'Masha', 'Sasha']
students.append('Olga')
```

Результат: ['Ivan', 'Masha', 'Sasha', 'Olga']

```
students += ['Olga']
```

Результат: ['Ivan', 'Masha', 'Sasha', 'Olga', 'Olga']

```
students += ['Boris', 'Sergey']
```

Результат: ['Ivan', 'Masha', 'Sasha', 'Olga', 'Olga', 'Boris', 'Sergey']

Пустой список: students = []

Вставка элемента:

```
students = ['Ivan', 'Masha', 'Sasha']
students.insert(1, 'Olga')
```

Результат: ['Ivan', 'Olga', 'Masha', 'Sasha']

Удаление элемента из списка

```
students = ['Ivan', 'Masha', 'Sasha']
students.remove('Sasha')
```

Результат: ['Ivan', 'Masha']

```
del students[0]
```

Результат: **‘Masha’**

Поиск элемента в списке

```
students = ['Ivan', 'Masha', 'Sasha']
if 'Ivan' in students:
    print('Ivan is here!')
if 'Ann' not in students:
    print('Ann is out')


ind = students.index('Sasha')
```

Результат: 2

```
ind = students.index('Ann')
```

Результат: ValueError: 'Ann' is not in list

Сортировка списка

Не изменяя порядка изначального списка

```
students = ['Sasha', 'Ivan', 'Masha']
ordered_students = sorted(students)
```

Результат ['Ivan', 'Masha', 'Sasha']

min()

max()

Изменяя сам список

```
students.sort()
```

Результат ['Ivan', 'Masha', 'Sasha']

Список в обратном порядке

```
students = ['Sasha', 'Ivan', 'Masha']
students.reverse()
```

Результат: ['Masha', 'Ivan', 'Sasha']

```
reversed(students)
```

```
students[::-1]
```

Присвоение списков

```
a = [1, 'A', 2]
b = a
a[0] = 42
Значение a: [42, 'A', 2]
Значение b: [42, 'A', 2]
b[2] = 30
Значение b: [42, 'A', 30]
Значение a: [42, 'A', 30]
```

Генераторы списков

```
A=[ i for i in range(10) ]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
A=[ i*i for i in range(10) ]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
from random import randint
A = [ randint(20,100) for x in range(10)]
```

```
A=[ i*i for i in range(10) if i%2==0
```

```
[0, 2, 4, 6, 8]
```

условие отбора

```
A=list( range(10) )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Двумерные списки

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
a[1]
```

```
[4, 5, 6]
```

```
a[1][1]
```

```
5
```

Программирование на языке Python

Массивы

Что такое массив?

Массив – это группа переменных одного типа, расположенных в памяти рядом (в соседних ячейках) и имеющих общее имя. Каждая ячейка в массиве имеет уникальный номер (индекс).

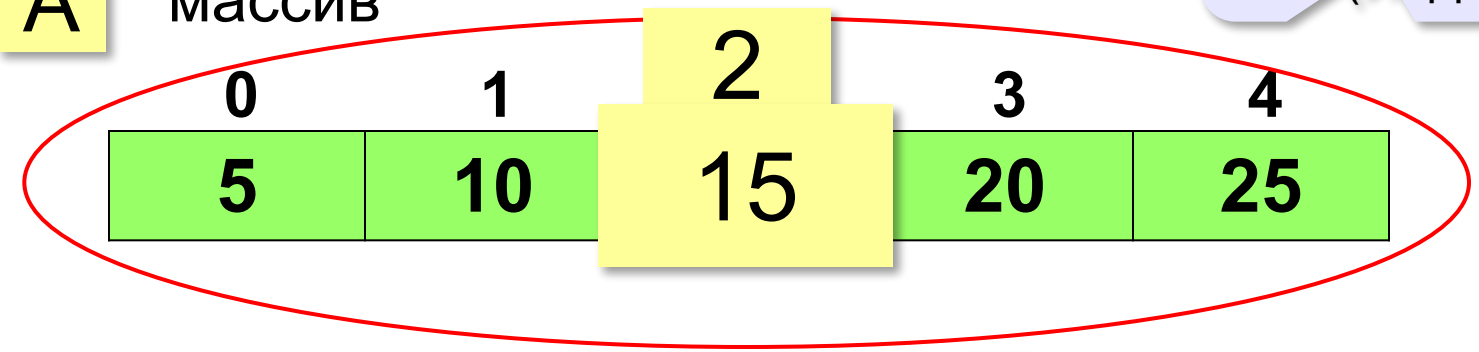
Надо:

- выделять память
- записывать данные в нужную ячейку
- читать данные из ячейки

Что такое массив?

Массив = **таблица**!

A массив



НОМЕР элемента массива (ИНДЕКС)

`A[0]` `A[1]` `A[4]`

ЗНАЧЕНИЕ элемента массива

НОМЕР (ИНДЕКС) элемента массива: 2

`A[2]`

ЗНАЧЕНИЕ элемента массива: 15

Ввод массива с клавиатуры

Ввод без подсказок:

```
A = [ int(input()) for i in range(N) ]
```

Ввод с подсказками:

```
A=[int(input("A["+str(i)+"]=")) for i in range(N)]
```

Ввод в одной строке: (для Python 3)

```
data = input()      # "1 2 3 4 5"
s = data.split()    # ["1","2","3","4","5"]
A = [ int(x) for x in s ]
                    # [1,2,3,4,5]
```

Вывод массива на экран

Как список:

```
print ( A )
```

```
[1, 2, 3, 4, 5]
```

В строчку через пробел:

записать как строку

```
s = [ str(x) for x in A]
print ( " ".join( s ) )
```

соединить через пробел

```
1 2 3 4 5
```

Для Python 3:

```
for i in range(N):
  print ( A[i], end = " " )
```

или так:

```
for x in A:
  print ( x, end = " " )
```

Подсчёт нужных элементов

Задача. В массиве записаны данные о росте баскетболистов. Сколько из них имеет рост больше 180 см, но меньше 190 см?

```
count = 0
for x in A:
  if 180 < x and x < 190:
    count += 1
```

Задачи

«A»: Заполните массив случайными числами в интервале [0,100] и найдите среднее арифметическое его значений.

Пример:

```
Массив:
1 2 3 4 5
Среднее арифметическое 3.000
```

«B»: Заполните массив случайными числами в интервале [0,100] и подсчитайте отдельно среднее значение всех элементов, которые <50, и среднее значение всех элементов, которые ≥50.

Пример:

```
Массив:
3 2 52 4 60
Ср. арифм. элементов [0,50): 3.000
Ср. арифм. элементов [50,100]: 56.000
```

Задачи

«С»: Заполните массив из N элементов случайными числами в интервале [1,N] так, чтобы в массив обязательно вошли все числа от 1 до N (постройте случайную перестановку).

Пример:

```
Массив:
3 2 1 4 5
```

Программирование на языке Си

Алгоритмы обработки массивов

Поиск в массиве

Вариант с досрочным выходом:

номер найденного элемента

X-искомый элемент
N – количество элементов массива

```
nX = -1
for i in range ( N ):
  if A[i] == X:
    nX = i
    break
if nX >= 0:
  print ( "A[", nX, "]=", X, sep = "" )
else:
  print ( "Не нашли!" )
```

досрочный выход из цикла

Поиск в массиве

Варианты в стиле Python:

```
for i in range( N ):
  if A[i]==X:
    print( "A[", i, "]=", X)
    break
else:
  print( "Не нашли!" )
```

если не было досрочного выхода из цикла

```
if X in A:
  nX=A.index(X)
  print( "A[", nX, "]=", X)
else:
  print( "Не нашли!" )
```

Задачи

«А»: Заполните массив случайными числами в интервале [0,5]. Введите число X и найдите все значения, равные X.

Пример:

```
Массив:
1 2 3 1 2
Что ищем:
2
Нашли: A[1]=2, A[4]=2
```

Пример:

```
Массив:
1 2 3 1 2
Что ищем:
6
Ничего не нашли.
```

Задачи

«B»: Заполните массив случайными числами в интервале [0,5]. Определить, есть ли в нем элементы с одинаковыми значениями, стоящие рядом.

Пример:

```
Массив:
1 2 3 3 2 1
Есть: 3
```

Пример:

```
Массив:
1 2 3 4 2 1
Нет
```

Задачи

«С»: Заполните массив случайными числами. Определить, есть ли в нем элементы с одинаковыми значениями, не обязательно стоящие рядом.

Пример:

```
Массив:
3 2 1 3 2 5
Есть: 3, 2
```

Пример:

```
Массив:
3 2 1 4 0 5
Нет
```

Максимальный элемент

```
M=A[0]
for i in range(1,N):
  if A[i]>M:
    M=A[i]
print( M )
```

Если `range(N)`?

Варианты в стиле Python:

```
M=A[0]
for x in A:
  if x>M:
    M=x
```

Как найти его номер?

```
M=max( A )
```

Максимальный элемент и его номер

```
M = A[0]; nMax = 0
for i in range(1,N):
    if A[i] > M:
        M = A[i]
        nMax = i
print ( "A[", nMax, "]=", M, sep = "" )
```

? Что можно улучшить?

! По номеру элемента можно найти значение!

```
nMax = 0
for i in range(1,N):
    if A[i] > A[nMax]
        nMax = i
print ( "A[", nMax, "]=", A[nMax], sep = "" )
```

Максимальный элемент и его номер

Вариант в стиле Python:

```
M = max(A)
nMax = A.index(M)
print( "A[", nMax, "]=", M, sep="" )
```

номер заданного элемента (первого из…)

Задачи

«A»: Заполнить массив случайными числами и найти минимальный и максимальный элементы массива и их номера.

Пример:

```
Массив:
1 2 3 4 5
Минимальный элемент: A[1]=1
Максимальный элемент: A[5]=5
```

«B»: Заполнить массив случайными числами и найти два максимальных элемента массива и их номера.

Пример:

```
Массив:
5 5 3 4 1
Максимальный элемент: A[1]=5
Второй максимум: A[2]=5
```

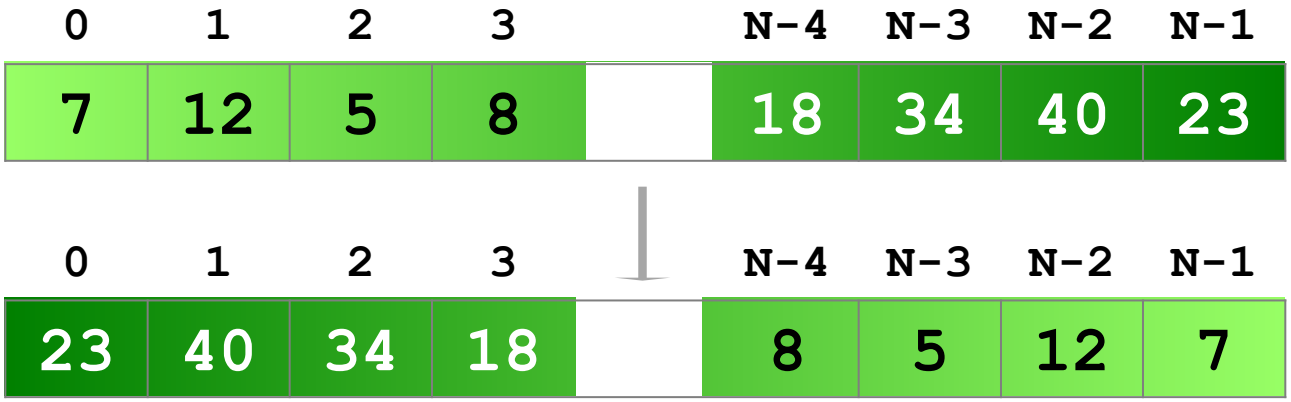
Задачи

«С»: Введите массив с клавиатуры и найдите (за один проход) количество элементов, имеющих максимальное значение.

Пример:

```
Массив:
3 4 5 5 3 4 5
Максимальное значение 5
Количество элементов 3
```

Реверс массива



«Простое» решение:

остановиться на середине!

```
for i in range( N//2 ):
    поменять местами A[i] и A[N-1-i]
```

? Что плохо?

Реверс массива

```
for i in range(N//2):
    c=A[i]
    A[i]=A[N-1-i]
    A[N-1-i]=c
```

Варианты в стиле Python:

```
for i in range(N//2):
    A[i],A[N-i-1]=A[N-i-1],A[i]
```

```
A.reverse()
```

Срезы в Python

разрезы: 0 1 2 3 N-4 N-3 N-2 N-1 N

```
A[1:3] → [12, 5]
A[2:3] → [5]
A[:3] → A[0:3] → [7, 12, 5]
```

с начала

```
A[3:N-2] → [8,…,18,34]
A[3:] → A[3:N] → [8,…,18,34,40,23]
```

до конца

копия массива

```
A[:] → [7,12,5,8,…,18,34,40,23]
```

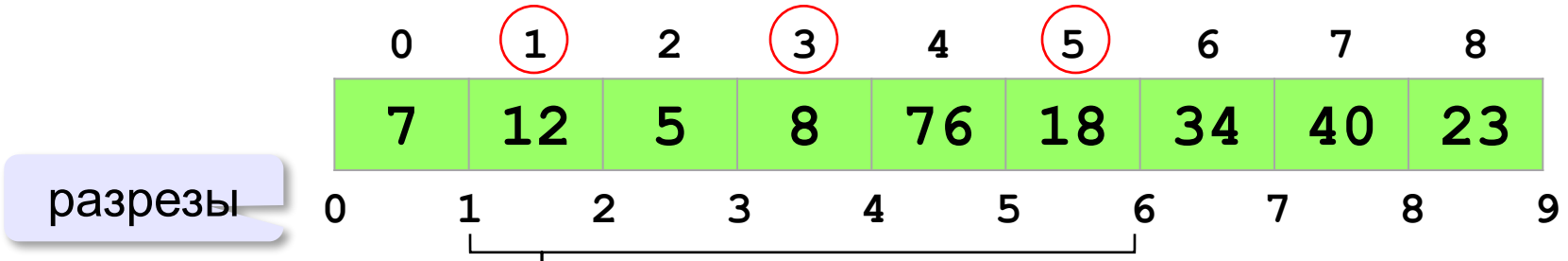
Срезы в Python – отрицательные индексы

разрезы


```
A[1:-1]
A[1:N-1]   ⟶  [12,5,8,…,18,34,40]

A[-4:-2]
A[N-4:N-2] ⟶  [18, 34]
```

Срезы в Python – шаг



шаг

```
A[1:6:2] — [12, 8, 18]
  A[::3] — [7, 8, 34]
A[8:2:-2] — [23, 34, 76]
 A[::-1] — [23,40,34,18,76,8,5,12,7]
```

реверс! `A.reverse()`

Задачи

«A»: Заполнить массив случайными числами и выполнить циклический сдвиг элементов массива вправо на 1 элемент.

Пример:

```
Массив:
1 2 3 4 5 6
Результат:
6 1 2 3 4 5
```

«B»: Массив имеет четное число элементов. Заполнить массив случайными числами и выполнить реверс отдельно в первой половине и второй половине.

Пример:

```
Массив:
1 2 3 4 5 6
Результат:
3 2 1 6 5 4
```

Задачи

«С»: Заполнить массив случайными числами в интервале [-100,100] и переставить элементы так, чтобы все положительные элементы стояли в начала массива, а все отрицательные и нули – в конце. Вычислите количество положительных элементов.

Пример:

```
Массив:
20 -90 15 -34 10 0
Результат:
20 15 10 -90 -34 0
Количество положительных элементов: 3
```

Отбор нужных элементов

Задача. Отобрать элементы массива **A**, удовлетворяющие некоторому условию, в массив **B**.

Простое решение:

```
B = []
сделать для i от 0 до N-1
  если условие выполняется для A[i] то
    добавить A[i] к массиву B
```

```
B = []
for x in A:
  if x % 2 == 0:
    B.append(x)
```

? Какие элементы выбираем?

добавить **x** в конец массива **B**

Отбор нужных элементов

Задача. Отобрать элементы массива **A**, удовлетворяющие некоторому условию, в массив **B**.

Решение в стиле Python:

перебрать все элементы A

```
B = [ x for x in A
      if x % 2 == 0 ]
```

если **x** – чётное число

Задачи

«A»: Заполнить массив случайными числами в интервале [-10,10] и отобрать в другой массив все чётные отрицательные числа.

Пример:

```
Массив A:
-5 6 7 -4 -6 8 -8
Массив B:
-4 -6 -8
```

«B»: Заполнить массив случайными числами в интервале [0,100] и отобрать в другой массив все простые числа. Используйте логическую функцию, которая определяет, является ли переданное ей число простым.

Пример:

```
Массив A:
12 13 85 96 47
Массив B:
13 47
```

Задачи

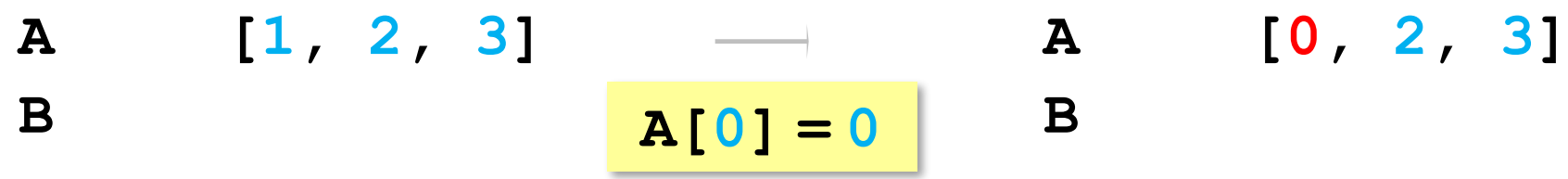
«С»: Заполнить массив случайными числами и отобрать в другой массив все числа Фибоначчи. Используйте логическую функцию, которая определяет, является ли переданное ей число числом Фибоначчи.

Пример:

```
Массив A:
12 13 85 34 47
Массив B:
13 34
```

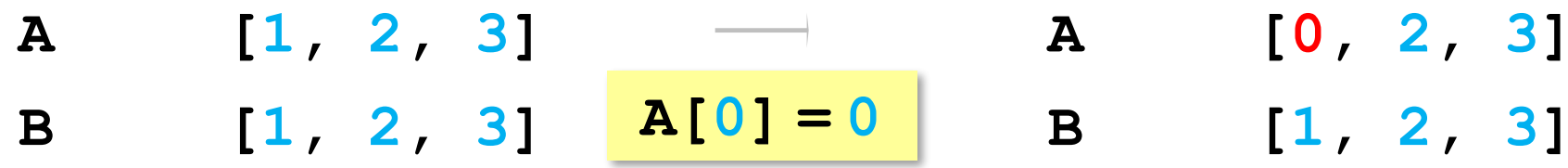
Особенности работы со списками

```
A = [1, 2, 3]
B = A
```




```
A = [1, 2, 3]
B = A[:]
```

копия массива A



Копирование списков

«Поверхностное» копирование:

```
import copy
A = [1, 2, 3]
B = copy.copy(A)
```

A [1,2,3]

B [4,5,6]

```
A = [1, 2, 3]
B = [4, 5, 6]
C = [A, B]
D = copy.copy(C)
C[1][0] = 0
```

C [A,B] A [0,2,3]

D [A,B] B [4,5,6]

! Влияет на C и D!

A

«Глубокое» копирование:

```
D = copy.deepcopy(C)
```

C [A,B] A [1,2,3]
B [4,5,6]

D [•,•] [1,2,3]
[4,5,6]

Программирование на языке Си

Сортировка

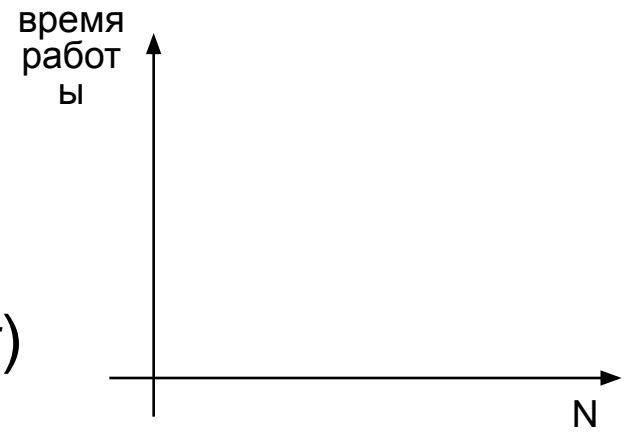
Что такое сортировка?

Сортировка – это расстановка элементов массива в заданном порядке.

…по возрастанию, убыванию, последней цифре, сумме делителей, по алфавиту, …

Алгоритмы:

- простые и понятные, но неэффективные для больших массивов
  - **метод пузырька**
  - **метод выбора**
- сложные, но эффективные
  - **«быстрая сортировка»** (*QuickSort*)
  - сортировка «кучей» (*HeapSort*)
  - сортировка слиянием (*MergeSort*)
  - пирамидальная сортировка

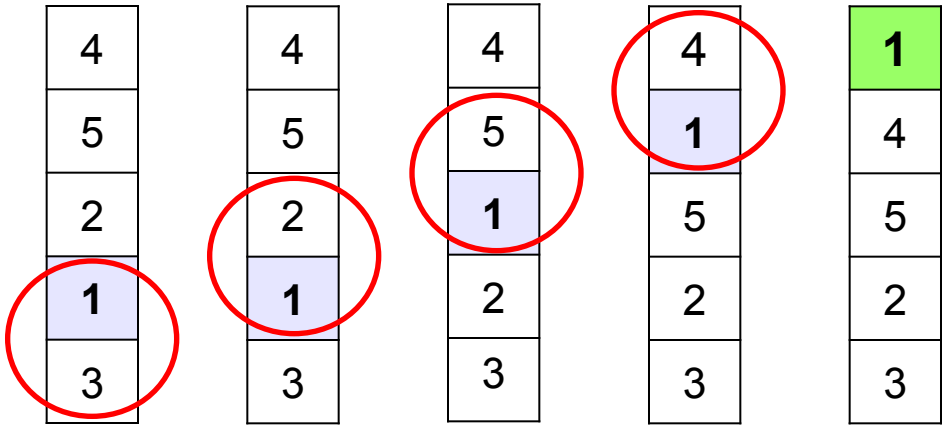


Метод пузырька (сортировка обменами)

Идея: пузырек воздуха в стакане воды поднимается со дна вверх.

Для массивов – **самый маленький** («легкий» элемент перемещается вверх (**«всплывает»**).

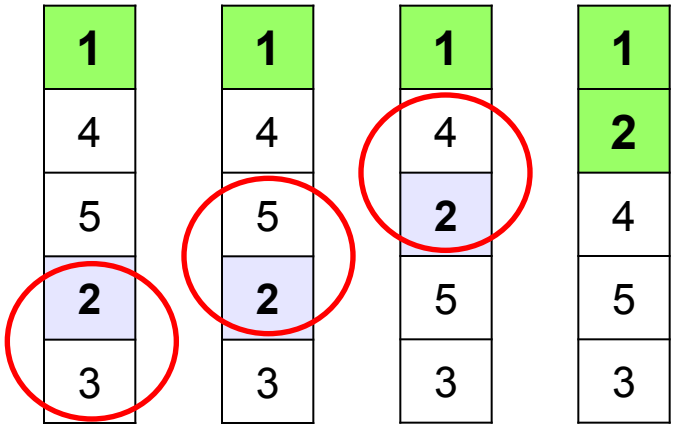
1-й проход:



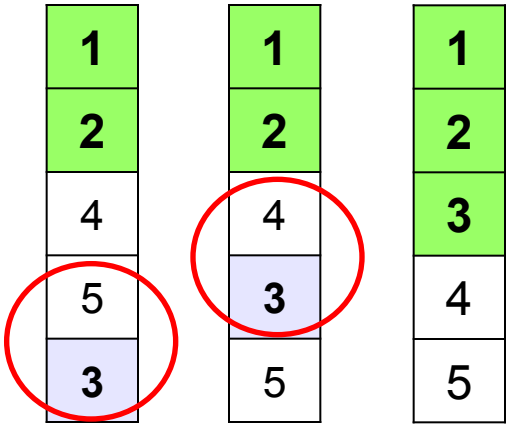
- сравниваем два соседних элемента; если они стоят «неправильно», меняем их местами
- за 1 проход по массиву один элемент (самый маленький) становится на свое место

Метод пузырька

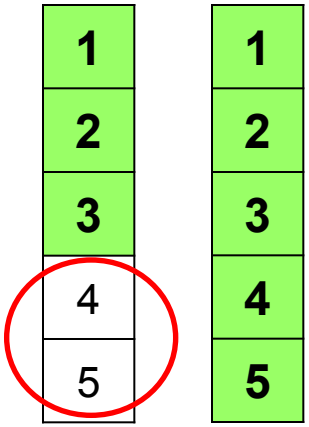
2-й проход:



3-й проход:



4-й проход:



Для сортировки массива из **N** элементов нужен **N-1** проход (достаточно поставить на свои места **N-1** элементов).

Метод пузырька

1-й проход:

```
сделать для j от N-2 до 0 шаг -1
  если A[j+1]<A[j] то
    # поменять местами A[j] и A[j+1]
```

единственное отличие!

2-й проход:

```
сделать для j от N-2 до 1 шаг -1
  если A[j+1]<A[j] то
    # поменять местами A[j] и A[j+1]
```

Метод пузырька

от N-2 до 0 шаг -1

1-й проход:

```
for j in range(N-2, -1 ,-1):
  if A[j+1]<A[j]:
    # поменять местами A[j] и A[j+1]
```

единственное отличие!

2-й проход:

```
for j in range(N-2, 0 ,-1):
  if A[j+1]<A[j]:
    # поменять местами A[j] и A[j+1]
```

Метод пузырька

```
for i in range(N-1):
  for j in range(N-2, i- ,-1):
    if A[j+1]<A[j]:   1
      A[j],A[j+1]=A[j+1],A[j]
```

? Как написать метод «камня»?

? Как сделать рекурсивный вариант?

Задачи

«A»: Напишите программу, в которой сортировка выполняется «методом камня» – самый «тяжёлый» элемент опускается в конец массива.

«B»: Напишите вариант метода пузырька, который заканчивает работу, если на очередном шаге внешнего цикла не было перестановок.

«C»: Напишите программу, которая сортирует массив по убыванию суммы цифр числа. Используйте функцию, которая определяет сумму цифр числа.

Метод выбора (минимального элемента)

Идея: найти минимальный элемент и поставить его на первое место.

```
for i in range(N-1):
  найти номер nMin минимального
      элемента из A[i]..A[N]
  if i != nMin:
    поменять местами A[i] и A[nMin]
```

Метод выбора (минимального элемента)

```
for i in range(N-1):
    nMin = i
    for j in range(i+1,N):
        if A[j] < A[nMin]:
            nMin = j
    if i != nMin:
        A[i], A[nMin] = A[nMin], A[i]
```

Задачи

«A»: Массив содержит четное количество элементов. Напишите программу, которая сортирует первую половину массива по возрастанию, а вторую – по убыванию. Каждый элемент должен остаться в «своей» половине.

Пример:

```
Массив:
5 3 4 2 1 6 3 2
После сортировки:
2 3 4 5 6 3 2 1
```

Задачи

«B»: Напишите программу, которая сортирует массив и находит количество различных чисел в нем.

Пример:

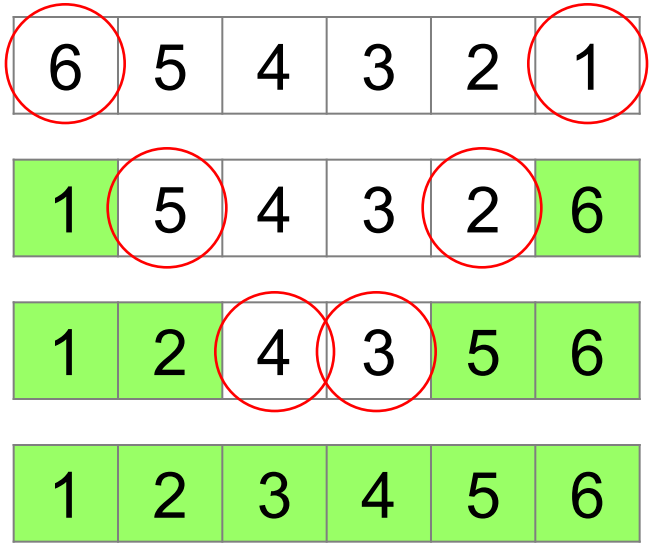
```
Массив:
5 3 4 2 1 6 3 2 4
После сортировки:
1 2 2 3 3 4 4 5 6
Различных чисел: 5
```

«C»: Напишите программу, которая сравнивает число перестановок элементов при использовании сортировки «пузырьком» и методом выбора. Проверьте ее на разных массивах, содержащих 1000 случайных элементов, вычислите среднее число перестановок для каждого метода.

Быстрая сортировка (*QuickSort*)

Ч.Э.Хоар

Идея: выгоднее переставлять элементы, который находятся дальше друг от друга.



Для массива из **N** элементов нужно всего **N/2** обменов!

Быстрая сортировка

Шаг 1: выбрать некоторый элемент массива X

Шаг 2: переставить элементы так:

`A[i] <= X`	`A[i] >= X`

при сортировке элементы не покидают « свою область»!

Шаг 3: так же отсортировать две получившиеся области

Разделяй и властвуй (англ. *divide and conquer*)

78	6	82	67	55	44	34

? Как лучше выбрать X?

Медиана – такое значение **X**, что слева и справа от него в отсортированном массиве стоит одинаковое число элементов (*для этого надо отсортировать массив…*).

Быстрая сортировка

Разделение:

1)выбрать любой элемент массива (**X=67**)

78	6	82	67	55	44	34

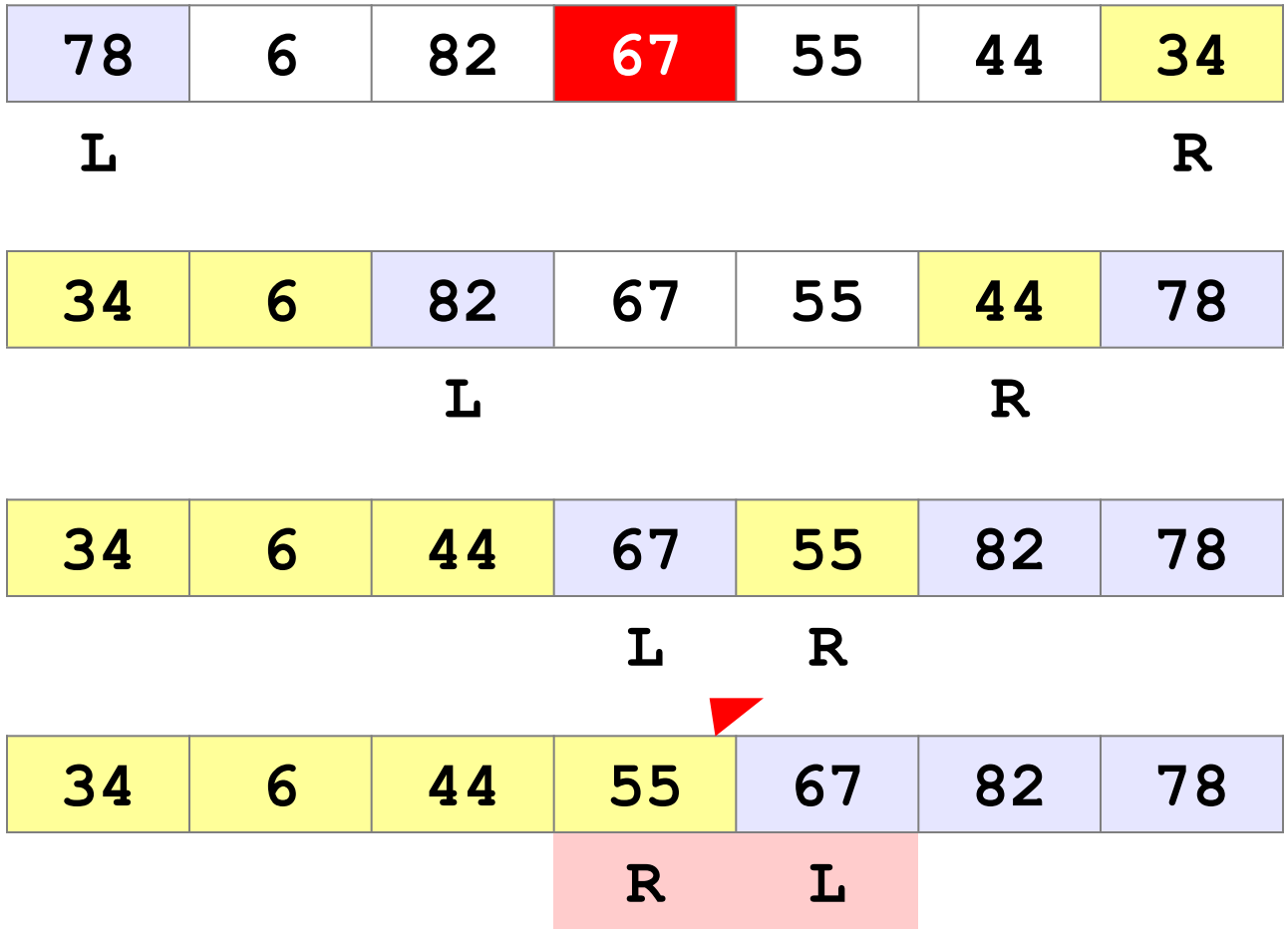
2)установить **L = 1, R = N**

3)увеличивая **L**, найти первый элемент **A[L]**, который **>= X** (должен стоять справа)

4)уменьшая **R**, найти первый элемент **A[R]**, который **<= X** (должен стоять слева)

5)если **L<=R** то поменять местами **A[L]** и **A[R]** и перейти к п. 3
иначе **стоп**.

Быстрая сортировка



L > R : разделение закончено!

Быстрая сортировка

Основная программа:

```
N = 7
A = [0]*N
# заполнить массив
qSort( A, 0, N-1 )  # сортировка
# вывести результат
```

Быстрая сортировка

массив | начало | конец

```
def qSort ( A, nStart, nEnd ):
  if nStart >= nEnd: return
  L = nStart; R = nEnd
  X = A[(L+R)//2]
  while L <= R:
    while A[L] < X: L += 1
    while A[R] > X: R -= 1
                                    L]
      L += 1; R -= 1
  qSort ( A, nStart, R )
  qSort ( A, L, nEnd )
```

разделение на 2 части

рекурсивные вызовы

Быстрая сортировка

Случайный выбор элемента-разделителя:

```
from random import randint
def qSort ( A, nStart, nEnd ):
    ...
    X = A[randint(L,R)]
    ...
```

или так:

```
from random import choice
def qSort ( A, nStart, nEnd ):
    ...
    X = choice ( A[L:R+1] )
    ...
```

Быстрая сортировка

В стиле Python:

```
from random import choice
def qSort ( A ):
   if len(A) <= 1: return A
   X = random.choice(A)
   B1 = [b for b in A if b < X]
   BX = [b for b in A if b == X]
   B2 = [b for b in A if b > X]
   return qSort(B1) + BX + qSort(B2)
```

окончание рекурсии

рекурсивные вызовы

? Что плохо?

```
Asort = qSort( A )
```

Быстрая сортировка

Сортировка массива случайных значений:

N	метод пузырька	метод выбора	быстрая сортировка
1000	0,09 с	0,05 с	0,002 с
5000	2,4 с	1,2 с	0,014 с
15000	22 с	11 с	0,046 с

Сортировка в Python

По возрастанию:

```
B = sorted( A )
```

алгоритм *Timsort*

По убыванию:

```
B = sorted( A, reverse = True )
```

По последней цифре:

```
def lastDigit ( n ):
  return n % 10
B = sorted( A, key = lastDigit )
```

или так:

```
B = sorted( A, key = lambda x: x % 10 )
```

«лямбда»-функция (функция без имени)

Сортировка в Python – на месте

По возрастанию:

```
A.sort()
```

По убыванию:

```
A.sort ( reverse = True )
```

По последней цифре:

```
def lastDigit ( n ):
  return n % 10
A.sort ( key = lastDigit )
```

или так:

```
A.sort( key = lambda x: x % 10 )
```

Задачи

«А»: Массив содержит четное количество элементов. Напишите программу, которая сортирует по возрастанию отдельно элементы первой и второй половин массива. Каждый элемент должен остаться в «своей» половине. Используйте алгоритм быстрой сортировки.

Пример:

```
Массив:
5 3 4 2 1 6 3 2
После сортировки:
2 3 4 5 6 3 2 1
```

Задачи

«B»: Напишите программу, которая сортирует массив и находит количество различных чисел в нем. Используйте алгоритм быстрой сортировки.

Пример:

```
Массив:
5 3 4 2 1 6 3 2 4
После сортировки:
1 2 2 3 3 4 4 5 6
Различных чисел: 5
```

Задачи

«C»: Напишите программу, которая сравнивает число перестановок элементов при использовании сортировки «пузырьком», методом выбора и алгоритма быстрой сортировки. Проверьте ее на разных массивах, содержащих 1000 случайных элементов, вычислите среднее число перестановок для каждого метода.

«D»: Попробуйте построить массив из 10 элементов, на котором алгоритм быстрой сортировки с с выбором среднего элемента показывает худшую эффективность (наибольшее число перестановок). Сравните это количество перестановок с эффективностью метода пузырька (для того же массива).

РАБОТА С ФАЙЛАМИ

Файл – это набор данных на диске, имеющий имя.

-*текстовые*, которые содержат текст, разбитый на строки; таким образом, из всех специальных символов в текстовых файлах могут быть только символы перехода на новую строку;

-*двоичные*, в которых могут содержаться любые данные и любые коды без ограничений; в двоичных файлах хранятся рисунки, звуки, видеофильмы и т.д.

Чтение из файла

```
inf = open('file.txt', 'r')  # open('file.txt')
s1 = inf.readline()
s2 = inf.readline()
inf.close()
```

"r" – открыть на чтение,
"w" – открыть на запись,
"a" – открыть на добавление.

```
with open('text.txt') as inf:
    s1 = inf.readline()
    s2 = inf.readline()

# здесь файл уже закрыт
```

начиная с версии 2.6

Формирование полного пути к файлу в любой ОС

```
os.path.join('.', 'dirname', 'filename.txt')
'./dirname/filename.txt'
```

Если нужно прочитать несколько данных в одной строке, разделённых пробелами, используют метод **split**. Этот метод разбивает строку по пробелам и строит список из соответствующих «слов»:

```
Fin = open ( "input.txt" )
s = Fin.readline().split()
```

Если в прочитанной строке файла были записаны числа 1 и 2, список **s** будет выглядеть так:

```
["1", "2"]
```

Построчное чтение файла

```
f=open('e:/0/qqq.top','r')
```

1)
```
while True:
    s=f.readline()
     print s
    if not s:
          break
```

2)
```
for i in f:
    print i
f.close()
```

3)
```
for s in open ('e:/0/qqq.top','r'):
        print (s)
```

Убираем ненужные символы

```
s = inf.readline().strip()
'\t abc  \n'.strip() → 'abc'
```

Генерация таблицы умножения

```
f=open('e:/0/qqq.txt','w')
a= [[i*j for j in range(1,10)] for i in
   range(1,10)]
for i in range(9):
    s=str(a[i])+'\n'
    f.write(s)
f.close()
```

Программирование на языке Python

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

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

Процедура с параметрами

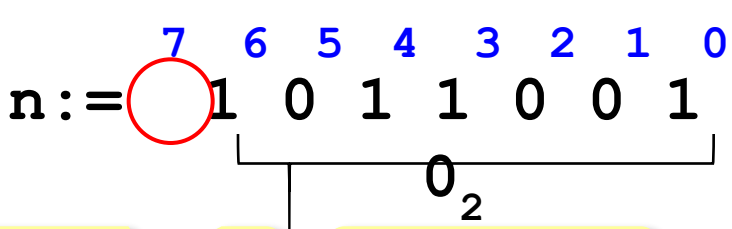
Задача. Вывести на экран запись целого числа (0..255) в 8-битном двоичном коде.

много раз!

Алгоритм:

```
178  ⇒  10110010
```

? 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

работаем с глобальной переменной

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

1

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
```

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**

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

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.
```