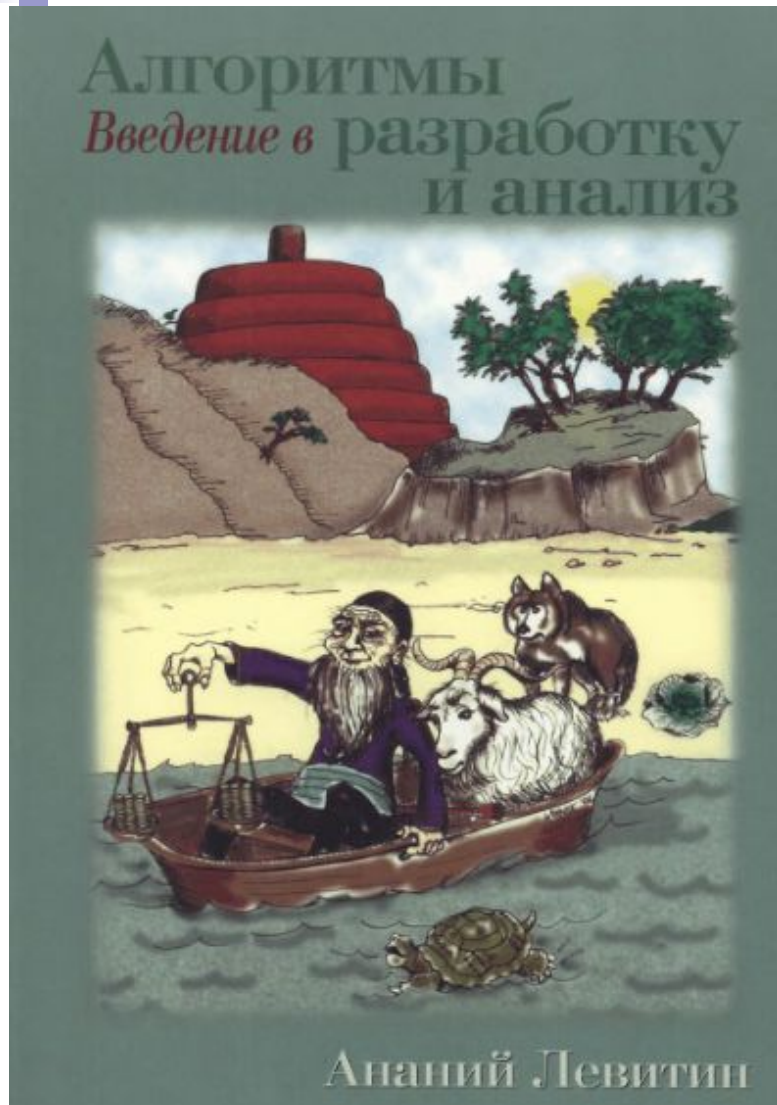




Алгоритмы

Литература



Книга ориентирована в первую очередь на начинающих разработчиков и содержит описание интересных и нетривиальных задач.

Описание алгоритмов на естественном языке дополняется псевдокодом, который позволяет реализовать алгоритм на любом языке программирования.

- *Левитин Ананий В. Алгоритмы: введение в разработку и анализ.: Пер. с англ. — М.: Вильямс, 2006. — 576 с. . [Электронный ресурс]. Режим доступа: levitin_algorithmy.divu*

Литература

1. *Левитин А.* Алгоритмы: введение в разработку и анализ.- М.: Вильямс, 2006. — 576 с.
2. *Стивенс Р.* Алгоритмы. Теория и практическое применение — М.:Издательство «Э», 2016. — 544 с.
3. *Кормен Т. Х.* Алгоритмы: Вводный курс — М.: Вильямс, 2015. — 208 с.
4. *Вирт Н.* Алгоритмы + структуры данных = программы. - М.: Мир.
5. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. — М.: МЦНМО. — 960 с.
6. *Кнут Д. Э.* - Искусство программирования. Том 1. Основные Алгоритмы.
7. *Скиена С.* Алгоритмы. Руководство по разработке. — 2-е изд.: Пер. с англ. — СПб.: БХВ-Петербург, 2011. — 720 с.



Задачи алгоритмизации

1. Построение нового или модификация некоторого ранее разработанного или определенного алгоритма.
2. Доказательство правильности алгоритма (верификация, тестирование).
3. Реализация применения разработанного или модифицированного алгоритма.
4. Анализ, оценка алгоритма по некоторым критериям его эффективности.

Алгоритм

Алгоритм – точное предписание, которое определяет процесс, ведущий от исходных данных к требуемому результату и достаточно определенное для того, чтобы ее можно было выполнить при помощи некоторого автоматического устройства.

Алгоритм – это формально описанная вычислительная процедура, получающая исходные данные, называемые также входом алгоритма или его аргументом, и выдающая результат вычислений на выход.

- ученый средневекового Востока - Муххамад ибн Муса ал-Хорезми (Магомет, сын Моисея, из Хорезма), в латинских переводах с арабского ал-Хорезми его имя определялось как algorismi.

Алгоритм

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

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

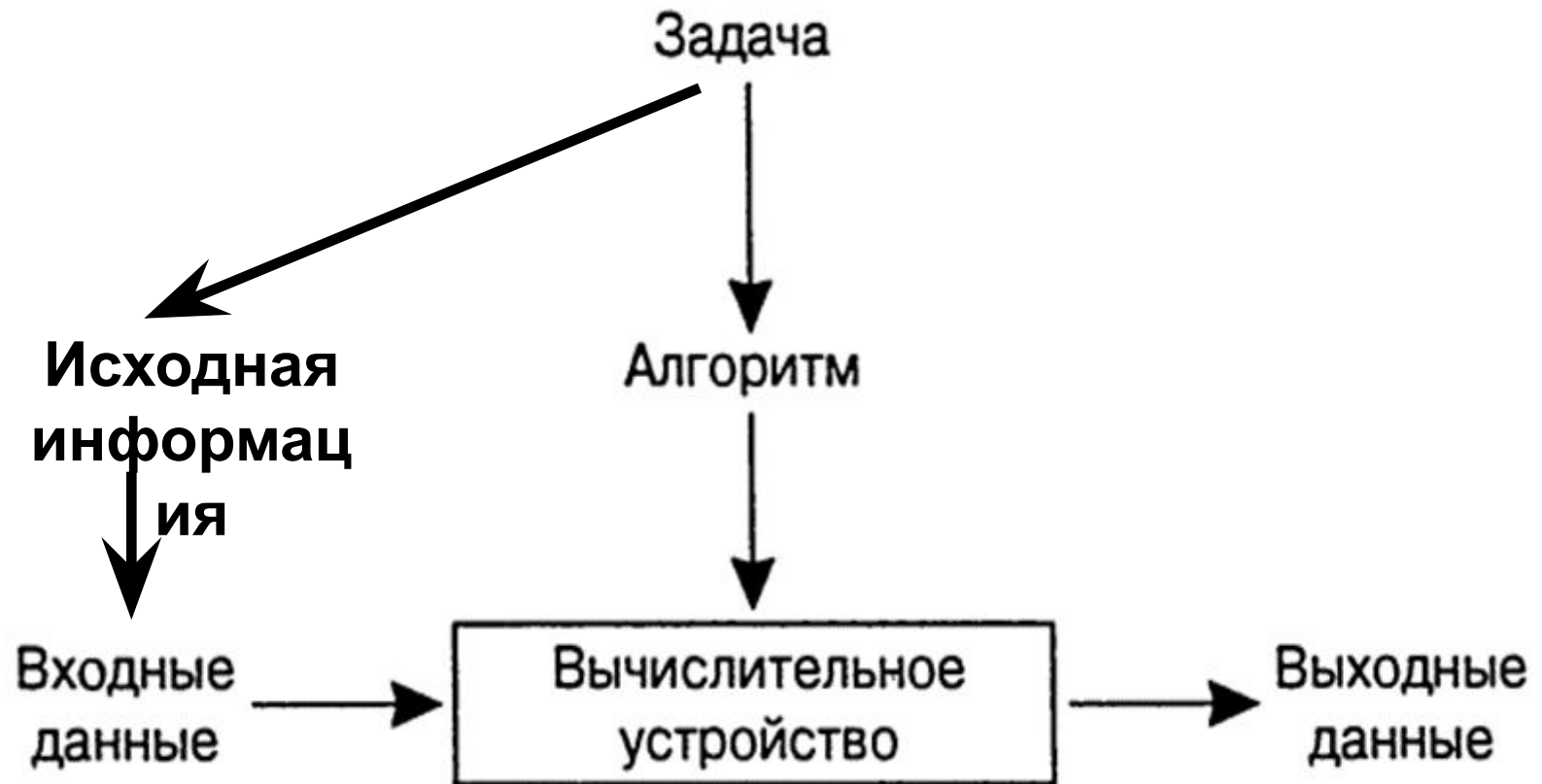
Формальные исполнители: не понимают (и не могут понять) смысл команд.

Пример. Уравнение: $y = \sin ax + b/2$

Исполнитель-человек:

- **знает**, как решать уравнения, в каком порядке выполняются арифметические операции
- **уточняет**, какие величины заданы и что необходимо вычислить
- **получает (запоминает)** заданные значения
- **использует** таблицы или калькулятор для получения значения синуса (или арксинуса)
- **формирует и выполняет** требуемую последовательность операций

Алгоритм



Существует некоторое абстрактное устройство, способное распознать инструкции и выполнить предписываемые ими действия

Виды алгоритмов

- **Детерминированный** алгоритм задает определенные действия, обозначая их в единственной и достоверной последовательности
- **Стохастический** (вероятностный) алгоритм дает программу решения задачи несколькими путями, приводящими к вероятностному достижению результата
- **Эвристический** алгоритм это такой алгоритм, в котором достижение конечного результата однозначно не определено, так же как не обозначена вся последовательность действий.

Основные требования предъявляемые к алгоритмам

- Алгоритм должен принимать данные (входные) и выдавать выходные данные – результат
- **Дискретность** – алгоритм состоит из отдельных команд, каждая из которых выполняется за конечное время
- **Конечность** (сходимость алгоритма) – решение задачи должно быть получено за конечное число шагов за конечное время
- **Детерминированность** – последовательность шагов алгоритма однозначно определена
- **Определенность** – для одних и тех же наборов исходных данных алгоритм должен выдавать один и тот же результат на любой устройстве

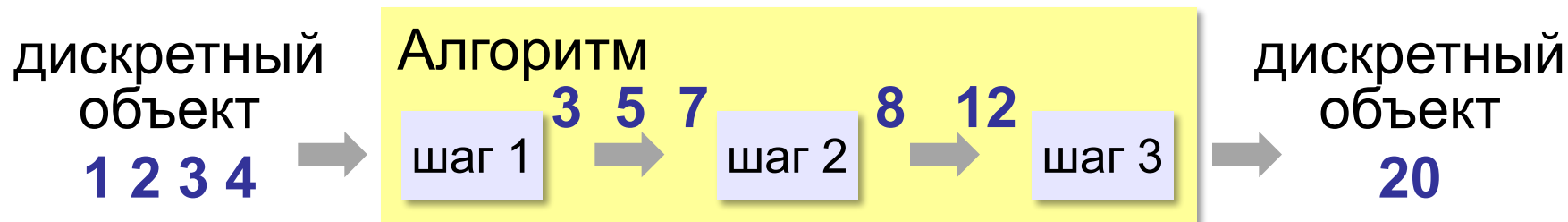
Основные требования предъявляемые к алгоритмам

- **Понятность** – алгоритм содержит только команды, входящие в систему команд исполнителя
- **Корректность** — для допустимых исходных данных алгоритм должен приводить к правильному результату
- Алгоритм предполагает наличие **механизма реализации**

Основные свойства алгоритма как процесса

- **Массовость** алгоритма – алгоритм должен описывать круг однотипных задач, исходные данные которых могут изменяться в определенных пределах
- **Эффективность** – как для решения задачи используются ограниченные ресурсы компьютера (процессорное время, объем оперативной памяти и т. д.)

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



Исполнитель алгоритма

- Исполняет алгоритм формально
- Исполняет только команды
- Не задумывается о том, какую задачу решает

Элементарность шагов

- Объем работы выполняемый на всяком шаге ограничен сверху некоторой константой, не зависящей от объема данных

Элементарные	Не элементарные
<ul style="list-style-type: none">- сложение;- вычитание;- умножение;- деление;- сравнение чисел	<ul style="list-style-type: none">- сравнение двух файлов;- проверка жесткого диска на вирусы;- архивирование папки



Основные способы описания алгоритмов

- Словесно-формульный
 - Естественный язык
 - Псевдокод
- Структурный или блок-схемный
- Табличный
- Граф-схемный

Алгоритм поиска НОД двух целых чисел

Вычисление НОД чисел m и n при помощи алгоритма Евклида

- Шаг 1. Если $n = 0$, вернуть m в качестве ответа и закончить работу; иначе перейти к шагу 2.
- Шаг 2. Поделить нацело m на n и присвоить значение остатка переменной r .
- Шаг 3. Присвоить значение n переменной m , а значение r — переменной n . Перейти к шагу 1.

Алгоритм поиска НОД двух целых чисел

Вычисление НОД чисел методом последовательного перебора

- Шаг 1. Присвоить значение функции $\min \{m, n\}$ переменной t .
- Шаг 2. Разделить m на t . Если остаток равен нулю, перейти к шагу 3; иначе перейти к шагу 4.
- Шаг 3. Разделить n на t . Если остаток равен нулю, вернуть t в качестве ответа и закончить работу; иначе перейти к шагу 4.
- Шаг 4. Вычесть 1 из t . Перейти к шагу 2.

Алгоритм поиска НОД двух целых чисел

Алгоритм 2 (Евклида)

- П1. Задать числа a и b .
- П2. Выбрать из этих чисел большее, назвать его x , меньшее назвать y .
- П3. Пока $y \neq 0$ выполнить
 - найти остаток z от деления x на y ;
 - заменить x на y ; и y на z ($z < y < x$).
- П4. Положить $d = x$.
- П5. Прекратить работу.

Алгоритм 3

- П1. Задать числа a и b .
- П2. Положить $x = a$, $y = b$.
- П3. Пока $x \neq y$ выполнить
 - если $x > y$ то положить $x = x - y$,
 - иначе положить $y = y - x$.
- П4. Положить $d = x$.
- П5. Прекратить работу.

Алгоритм поиска НОД двух целых чисел

Вычисление НОД чисел m и n школьным методом

- Шаг 1. Разложить на простые множители число m .
- Шаг 2. Разложить на простые множители число n .
- Шаг 3. Для простых множителей чисел m и n , найденных на шаге 1 и 2, выделить их общие делители. Если p является общим делителем чисел m и n и встречается в их разложении на простые множители, соответственно, p_m и p_n раз, то при выделении нужно повторить это $\min \{p_m, p_n\}$ раз.
- Шаг 4. Вычислить произведение всех выделенных общих делителей и вернуть его в качестве результата поиска НОД двух указанных чисел.

Метод определения високосного года

Чтобы определить, является ли год високосным, выполните следующие действия:

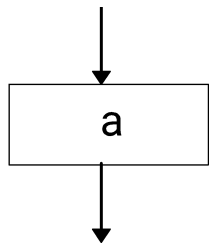
1. Если год делится на 4 без остатка, перейдите на шаг 2. В противном случае перейдите к выполнению действия 5.
2. Если год делится на 100 без остатка, перейдите на шаг 3. В противном случае перейдите к выполнению действия 4.
3. Если год делится на 400 без остатка, перейдите на шаг 4. В противном случае перейдите к выполнению действия 5.
4. Год високосный (366 дней).
5. Год не високосный год (365 дней).



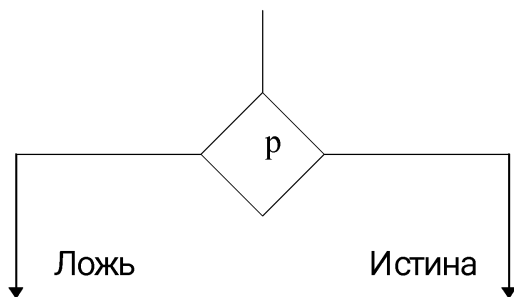
Это
алгоритм?

Представление алгоритмов в виде блок-схем

Блок - схема представляет собой двухмерный рисунок, построенный из управляющих структур. При рисовании этих структур используются специальные обозначения.



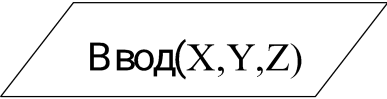
Обозначение обработки. Действие, которое необходимо выполнить, обозначается прямоугольником, в который входит и из которого выходит ровно одна линия управления. Прямоугольник называется **узлом обработки**, или **функциональным узлом**.



Обозначение проверки. Операция проверки обозначается символом, который называется **условным (предикатным) узлом**. Он представляет собой ромб, в который входит одна линия управления, а выходят две. В результате проверки помещенного внутри ромба условия (предиката) **p** выбирается один из выходов (но не оба сразу).

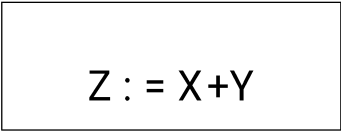
Управляющие структуры

Безальтернативные вычисления (управляющая структура "следование")



Ввод(X, Y, Z)

Ввод данных. Предписание на ввод данных содержит указание устройства ввода и имя переменной, значение которой надо ввести.



$Z := X + Y$

Изменение данных. Чаще всего предписание на изменение данных представляется в виде оператора присваивания (последовательности операторов присваивания).



Вывод(Z)

Вывод данных. Предписание на вывод данных содержит указание устройства вывода и имя переменной, значение которой следует вывести.

Управляющая структура "следование" используется в случае, когда алгоритм представляется как последовательность, элементами которой служат только действия по преобразованию данных.

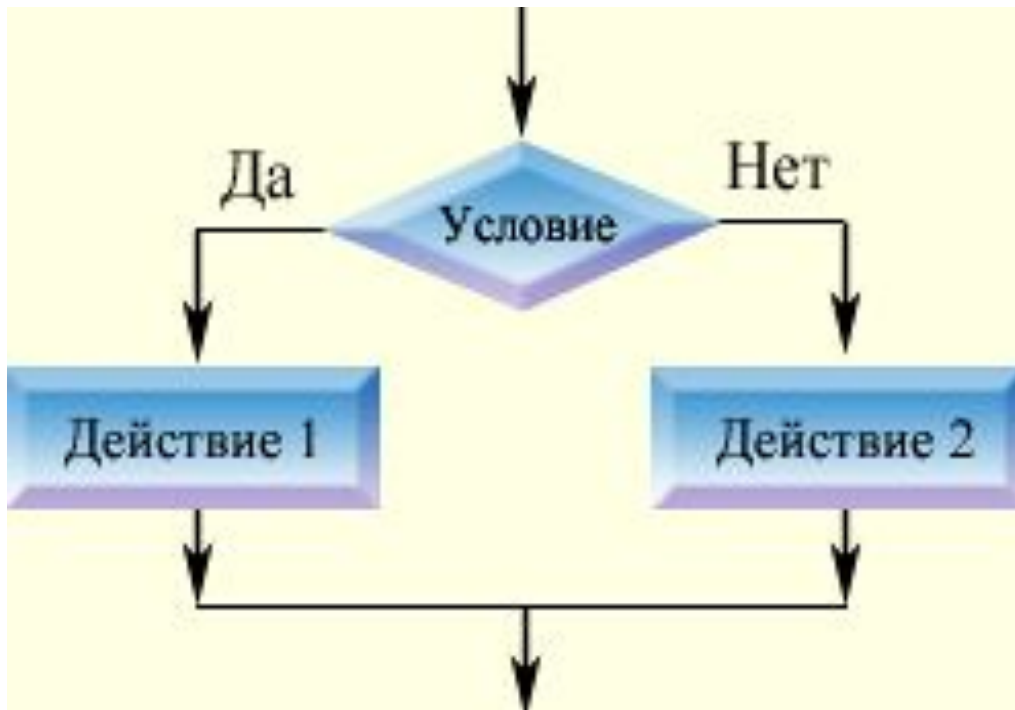
В этом случае алгоритм называют **линейным алгоритмом**.

Управляющие структуры

Пример if.py

Альтернативные вычисления (управляющая структура «выбор»)

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

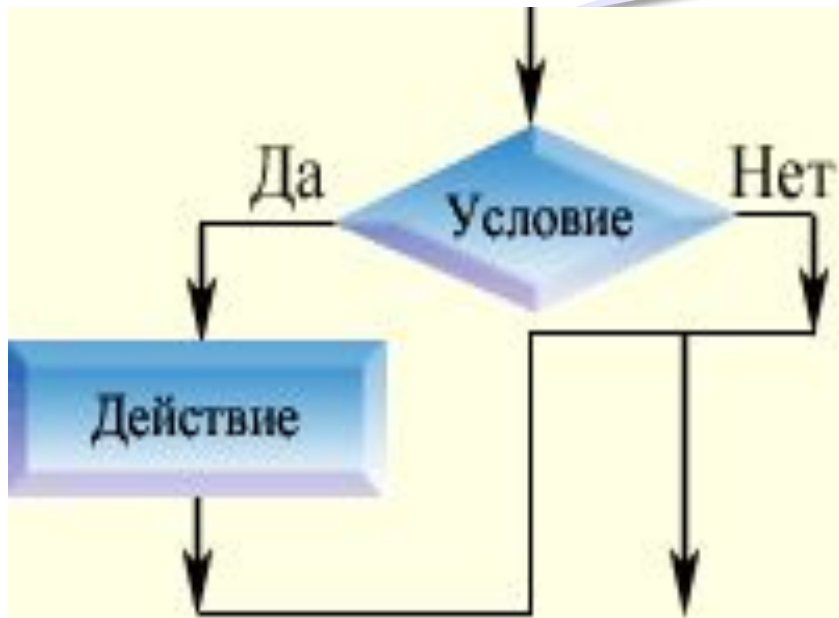


```
if a > b:  
    m = a  
else:  
    m = b
```


Управляющие структуры

Альтернативные вычисления (управляющая структура «выбор»)

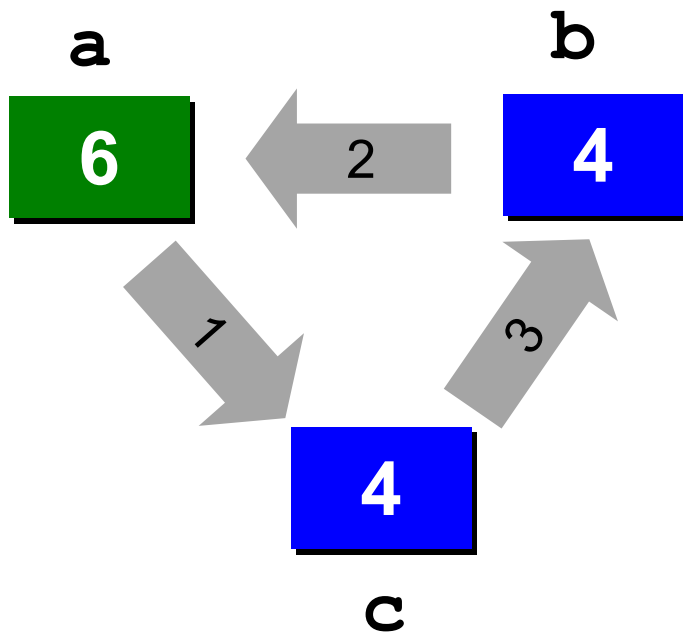
неполная форма ветвления



```
m = a
if b > a:
    m = b
```

Пример. Обмен значений

- Даны две переменные.
- Требуется обменять значения этих переменных.



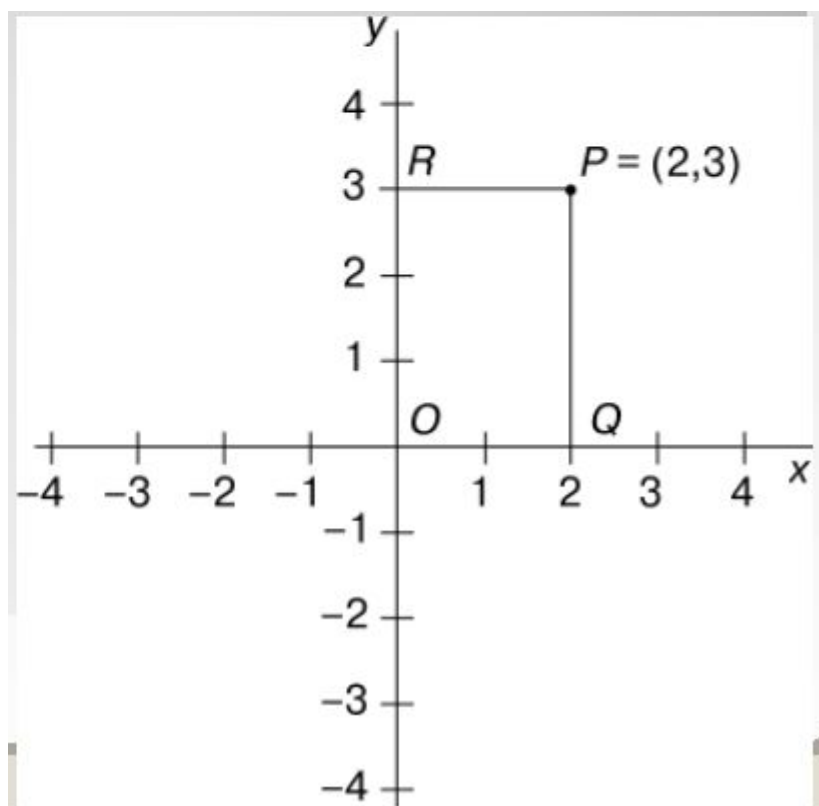
```
if a < b:  
    c = a  
    a = b  
    b = c
```

?

Можно ли обойтись без переменной **c**?

Пример.

- Дана точка в декартовой системе координат $P(x,y)$.
- Требуется составить условие определения в какой четверти находится данная точка.



Пример.

Алгоритмическое решение.

- Вариант 1. Последовательный перебор.
 - Последовательно составляются условия на принадлежность к каждой четверти.
- Вариант 2. Метод деления пополам.
 - Сначала сравнивается первая координата на условие принадлежности точки к половине системы координат.
 - Затем уточняется к какой из ее двух частей принадлежит точка

Пример.

Для решения задач выбора применяются:

- **Операции сравнения**, которые возвращают True (истина) или False (ложь)
- **Логические операторы**, применяемые для проверки одновременно несколько условий:
 - **X and Y** (Истина, если оба значения X и Y истинны)
 - **X or Y** (Истина, если хотя бы одно из значений X или Y истинно)
 - **not X** (Истина, если X ложно)

>	<	больше, меньше
>=		больше или равно
<=		меньше или равно
==		равно
!=		не равно

Приоритет :

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

Пример.

Программное решение.

- Вариант 1. Последовательный перебор.
Последовательно составляются условия на принадлежность к каждой четверти.

```
if (x > 0 && y > 0)
    "Первая четверть"
else if (x > 0 && y < 0)
    "Четвертая четверть"
else if (y > 0)
    "Вторая четверть"
else
    "Третья четверть"
```

Пример.

Пример if2.py

Программное решение.

- Вариант 2. Метод деления пополам. Сначала сравнивается первая координата на условие принадлежности точки к половине системы координат. Затем уточняется к какой из ее двух частей принадлежит точка

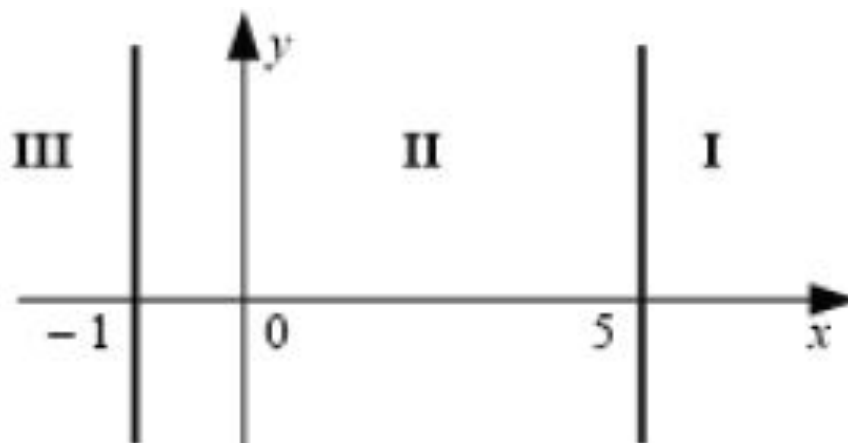
```
if (x > 0)
    if (y > 0)
        "Первая четверть"
    else
        "Четвертая четверть"
else
    if (y > 0)
        "Вторая четверть"
    else
        "Третья четверть"
```

Управляющие структуры

Альтернативные вычисления: три и более вариантов действий

Пример.

- На плоскости выделены три зоны (I, II, III). Дана координата x точки.



- Определить, в какую зону попала эта точка(допущение: x не равно границам зон -1 и 5)

Управляющие структуры

Альтернативные вычисления: три и более вариантов действий

Решение

- в программе использовать 3 неполных варианта инструкции if (без ветви else):

```
if x < -1:  
    print('В зону III')  
if x > 5:  
    print('В зону I')  
if x > -1 and x < 5:  
    print('В зону II')
```

Управляющие структуры

Пример DError_if.py

Альтернативные вычисления: три и более вариантов действий

Решение

- сэкономить одно слово if и использовать полный вариант инструкции:

```
if x < -1:  
    print('В зону III')  
if x > 5:  
    print('В зону I')  
else:  
    print('В зону II')
```

Проблема! Будьте внимательны, проверьте при $x = -8$

Управляющие структуры

Альтернативные вычисления: три и более вариантов действий

Решение

- Правильное решение использовать полный вариант инструкции:

```
x = -8
if x < -1:
    print("В зону III")
else:
    if x > 5:
        print('В зону I')
    else:
        print('В зону II')
```

Задача.

- Составьте алгоритм, который получает три числа и выводит количество одинаковых чисел в этом наборе

Пример:

Введите три числа:

5 5 5

Все числа одинаковые.

Пример:

Введите три числа:

5 7 5

Два числа одинаковые.

Пример:

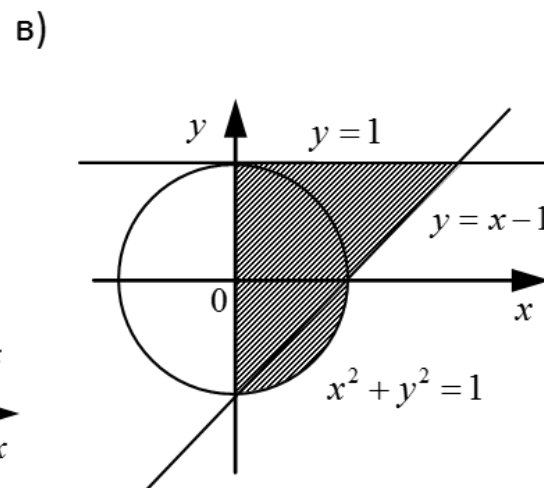
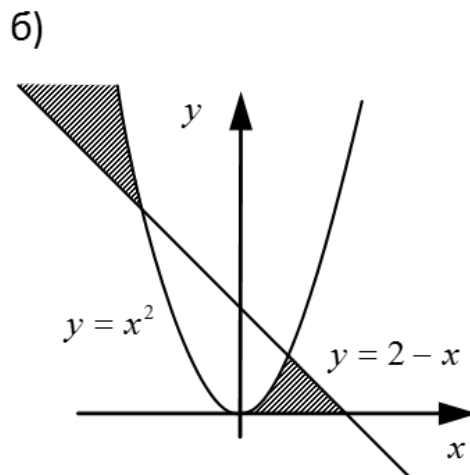
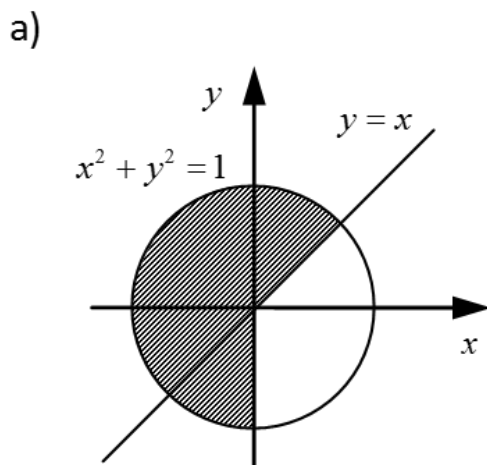
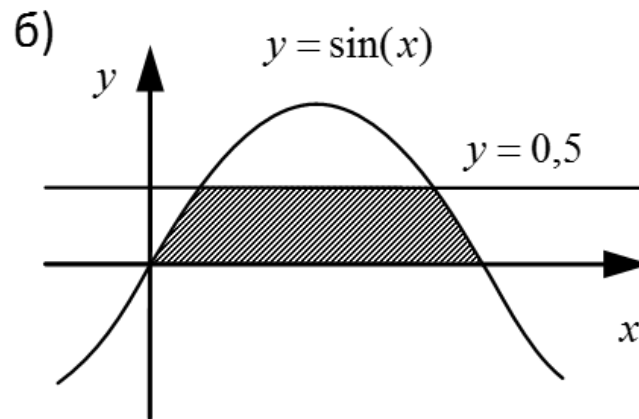
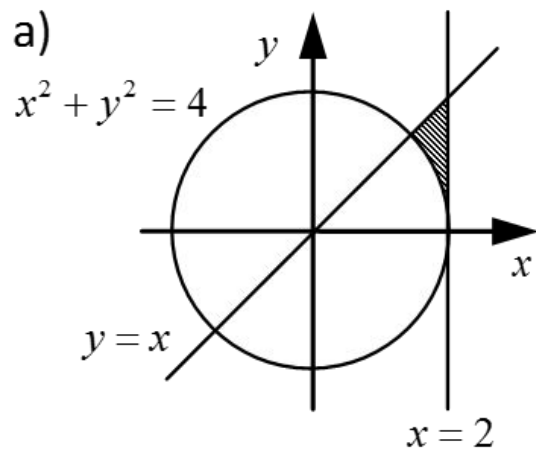
Введите три числа:

5 7 8

Нет одинаковых чисел.

Задача.

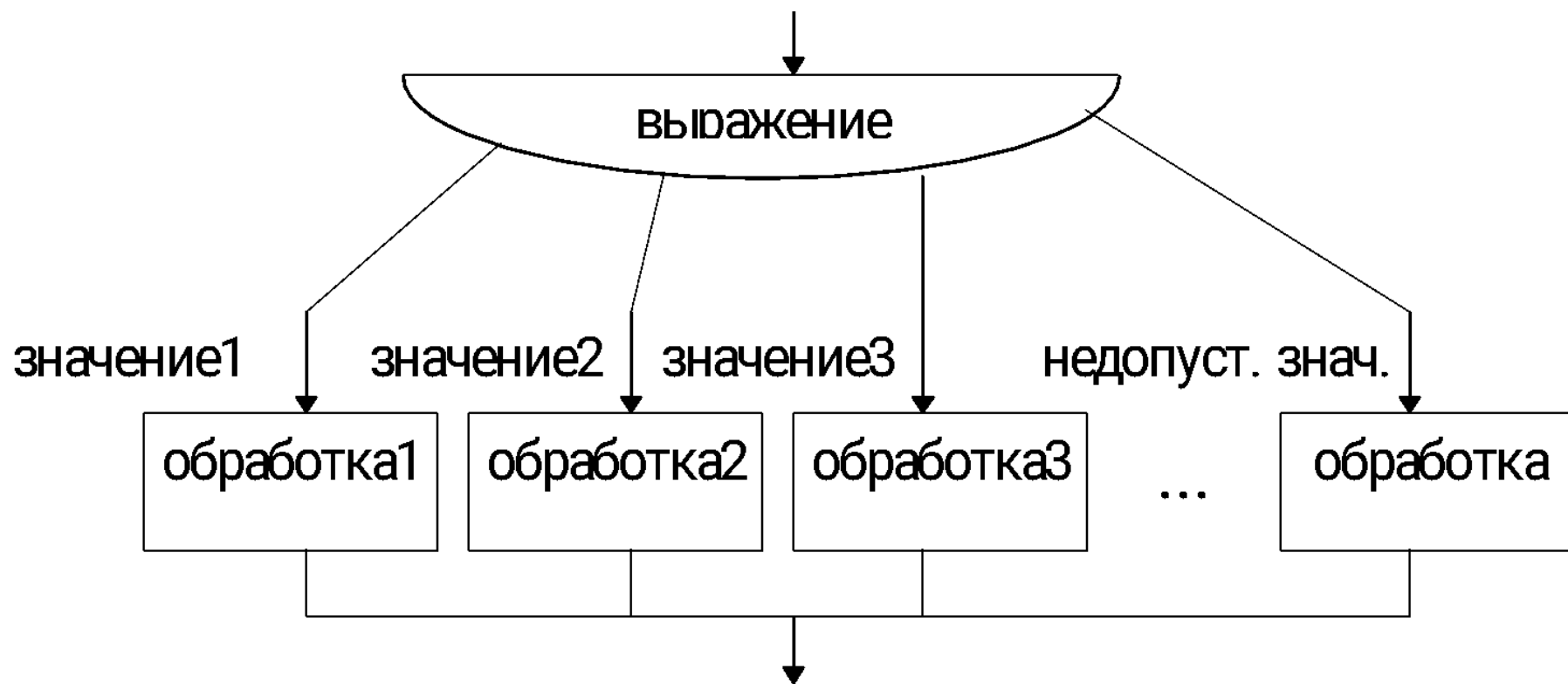
- Напишите условие, которое определяет заштрихованную область



Управляющие структуры

Альтернативные вычисления

(управляющая структура «**множественный выбор**»)



Множественный выбор (VBA)

```
Select Case Выражение
    Case Список_Выражений-n
        [действия – функции-n]]...
    Case Else
        [действия – функции_else]]
End Select
```

Выражение – обязательный элемент – любое числовое или строковое выражение.

Список_Выражений-n - обязательный элемент структуры при наличии предложения Case. Представляет собой список с разделителями. Структура такого списка может быть одной из следующих:

- *Выражение*,
- *Выражение_1 To Выражение_2*,
- Is Оператор_сравнения *Выражение*.

- Ключевое слово To задаёт диапазон значений. При использовании To перед ним должно находиться меньшее значение.
- Ключевое слово Is с операторами сравнения

Множественный выбор. Пример (C#)

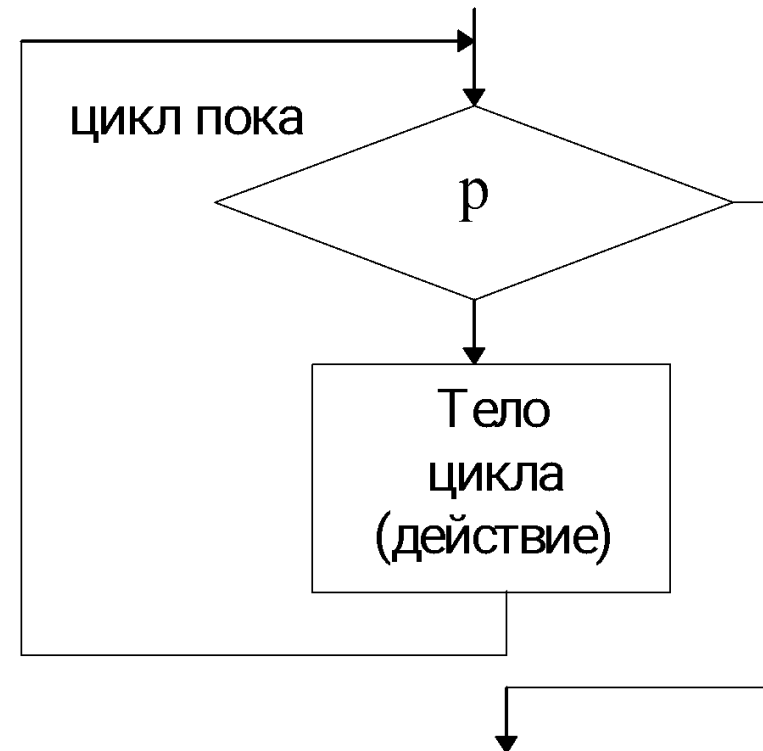
```
bool ok = true;
// ввод требуемых переменных (не приводится)
float res = 0;
switch (op)
{
    case '+':
        res = a + b; break;
    case '-':
        res = a - b; break;
    case '*':
        res = a * b; break;
    case '/':
    case ':':
        if (b != 0)
        {
            res = (float)a / b; break;
        }
        else
            ok = false; break;
}
if (ok) Console.WriteLine("{0} {1} {2} = {3}", a, op, b, res);
else Console.WriteLine("error");
```


Управляющие структуры

Повторяющиеся вычисления («цикл пока»)

Цикл – это многократное выполнение одинаковых действий

Цикл с **неизвестным** числом шагов



- Предписывает выполнять тело цикла до тех пор **ПОКА** истинно условие **p**.
- Тело "цикла пока" может не выполняться ни одного раза

Организация цикла

Пример while.py

Цикл с известным числом повторений

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

Задается в
условии

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

Задается в
переменной



Как увеличить счётчик на 1?

счётчик = счётчик + 1

счётчик += 1

Организация цикла

Цикл с **не известным** числом повторений

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

сколько_цифр = 0

пока **n** > 0:

отсечь последнюю цифру **n**

увеличить сколько_цифр на 1

значение **n** заранее
неизвестно



Как отсечь последнюю цифру?

n = **n** // 10

Решение. Определить количество цифр в десятичной записи целого положительного числа, записанного в переменную `n`

начальное
значение
переменной для
подсчета цифр

Счетчика цикла в этом
случае нет!

условие
продолжения

заголовок
цикла

тело цикла

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

изменение
значения
переменной для
подсчета цифр



Цикл с предусловием – проверка на входе в цикл!

Вопрос. Сколько раз выполняется цикл?

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

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

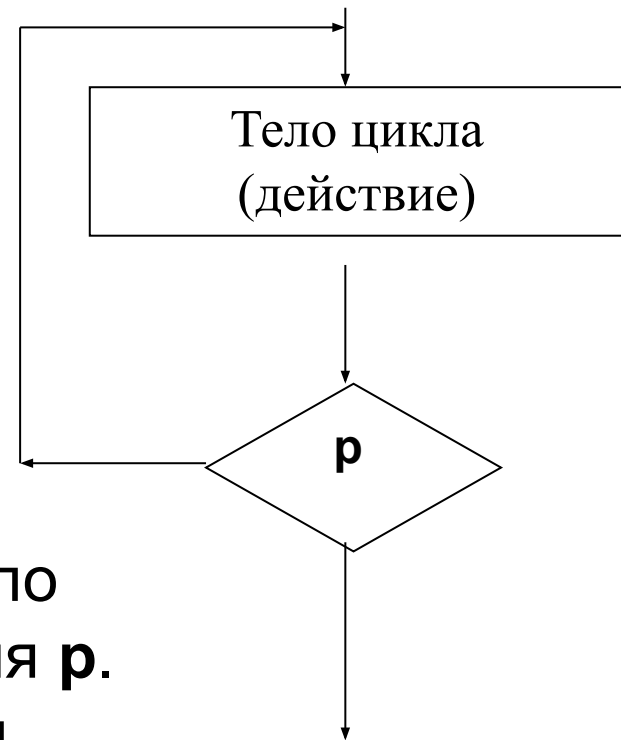
```
a = 4; b = 6  
while a > b:  
    a += 1
```

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

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

Управляющие структуры

Повторяющиеся вычисления («цикл до»)



- Предписывает выполнять тело цикла до выполнения условия **p**.
- Тело "цикла до" выполняется хотя бы один раз

Управляющие структуры

Пример Ugad_while.py

Повторяющиеся вычисления («цикл до»)

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

бесконечный
цикл

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

тело цикла

условие
выхода

прервать
цикл

- Предписывает выполнять тело цикла до выполнения условия **p**.
- Тело "цикла до" выполняется хотя бы один раз

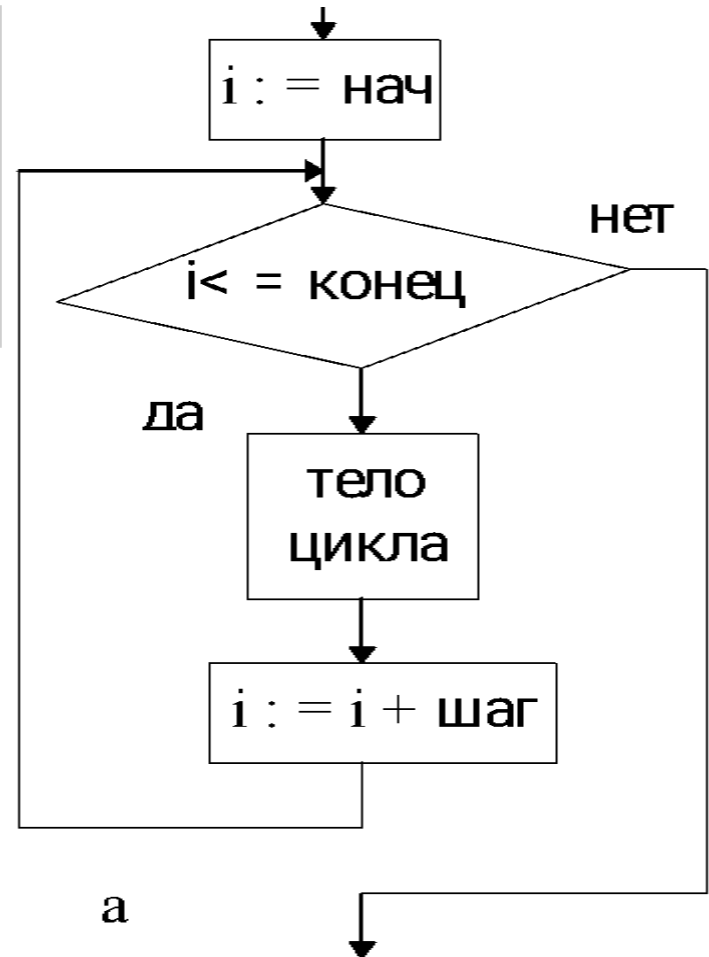
Управляющие структуры

Повторяющиеся вычисления («цикл для»)

```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine(i);  
}
```

0 1 2 3 4 5 6 7 8 9

- Приращение параметра цикла называется "шаг цикла"



Управляющие структуры

Повторяющиеся вычисления («цикл с переменной»)

```
foreach (int number in numbers)
{
    Console.WriteLine(number);
}
```

Выполняет инструкции для каждого элемента коллекции

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

Управляющие структуры

Повторяющиеся вычисления («цикл с переменной»)

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

1, 3, 5, 7, 9

шаг

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

1

9

25

49

81

В языке Python циклы предлагают дополнительную возможность: сразу после повторно выполняемого внутреннего блока цикла можно поместить блок **else**

```
for <элемент> in <последовательность>
    <тело цикла>
[else:
    <блок, который будет выполнен, если не использовался
оператор break>
]
```

```
while <логическое выражение>:
    <тело цикла>
[else:
    <блок, который будет выполнен, если не использовался
оператор break>
]
```

- ❑ Блоки **else** выполняются сразу же после того, как заканчивается выполнение цикла – почему используется именно слово "else"?

Особенности применения

```
for i in range (3):  
    print ('Цикл %d' % i)  
    if i == 1:  
        break  
else:  
    print ('Блок Else!')
```

Цикл 0
Цикл 1

```
for i in range (3):  
    print ('Цикл %d' % i)  
    if i == 1:  
        break  
else:  
    print ('Блок Else!')
```

Цикл 0
Блок Else!
Цикл 1

Особенности применения

```
for i in []:  
    print ('Важное действие')  
else:  
    print ('Тоже важно Else!')
```

Тоже важно Else!

```
for i in [1]:  
    print ('Важное действие')  
else:  
    print ('Тоже важно Else!')
```

Важное действие
Тоже важно Else!

Особенности применения

```
while False:  
    print ('Важное действие')  
else:  
    print ('Тоже важно Else!')
```

Тоже важно Else!

Когда использовать?

- Выполнение блоков **else** после циклов полезно в тех случаях, когда циклы используются для поиска каких-то объектов

Рекомендация. Избегайте использования блоков `else` после циклов, поскольку их поведение не является интуитивно понятным и может вносить путаницу



Практическое занятие

- 3. Реализация условных конструкций
- 4. Реализация циклов