



Урок 1

Введение в алгоритмизацию и простые алгоритмы на Python

Введение в алгоритмизацию. Решение практических задач.

[Введение](#)

[Основы алгоритмизации](#)

[Понятие алгоритма](#)

[Способы представления алгоритма](#)

[Блок-схема](#)

[Исполнители алгоритма](#)

[Виды алгоритмов](#)

[Линейный алгоритм](#)

[Разветвляющийся алгоритм](#)

[Циклический алгоритм](#)

[Закрепляем изученное](#)

[Задача 1. Сравниваем числа и находим максимум](#)

[Задача 2. Вычислить значение функции \$y=f\(x\)\$](#)

[Задача 3. Проверим делимость одного числа на другое](#)

[Задача 4. Перевести байты в килобайты или наоборот](#)

[Практическое задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Введение

В предыдущих курсах вы изучили основы программирования и синтаксис Python, научились решать практические задачи и разрабатывать графические многопользовательские приложения. Этот курс познакомит вас с фундаментальными алгоритмами программирования. Вы научитесь анализировать и выбирать оптимальные программные решения для своих задач.

Еще в начале 1970-х годов профессор Чарльз Хоар в статье «Аксиоматическая основа программирования для вычислительных машин» показал, что знание существующих алгоритмов помогает избежать ошибок в построении программ. Чтобы мыслить как профессионал, вам нужно освоить фундаментальные классические алгоритмы.

Можно сказать, что программирование — это вид конструирования. Мы собираем программы, классы, методы и функции из составных частей и постепенно дорабатываем конструкцию. Чтобы научиться строить алгоритмы под заданные требования, разберёмся в основах алгоритмизации и выясним, что же такое алгоритм.

Основы алгоритмизации

Когда вы думаете над задачей, вы строите в голове путь к её решению. Последовательность действий, которая ведёт к результату — это и есть алгоритм. С одним условием: решение должно быть применимо к разным исходным данным. Обобщим сказанное в одной формулировке.

Понятие алгоритма

Классический алгоритм — это совокупность действий (инструкций), которая приводит к достижению результата за конечное число шагов.

Почему мы говорим «совокупность», а не «последовательность»? Потому что действия в алгоритме не всегда выполняются последовательно — они могут повторяться, меняться местами или опускаться в зависимости от заданных условий.

Для более глубокого понимания алгоритма посмотрим, какими свойствами он должен обладать, чтобы с его помощью удобно было решать задачи.

Свойства алгоритмов:

1. **Дискретность (прерывность, раздельность).** Алгоритм должен состоять из элементарных действий, которые выполняются последовательно. Пока не закончится выполнение одного действия, следующее не начинается.
2. **Детерминированность.** Все действия в алгоритме должны быть строго определены: ни одно из них не может двояко трактоваться исполнителем.
3. **Конечность.** Алгоритм должен выполняться за определенное количество шагов.
4. **Массовость.** Алгоритм должен вести к результату при любых исходных данных.
5. **Результативность.** Работа алгоритма всегда должна приводить к ожидаемому результату.



Основная цель алгоритмизации — составление алгоритмов для определенного класса задач, решаемых исполнителем.

Несколько примеров алгоритма из повседневной жизни:

1. Инструкция по эксплуатации бытового прибора — это, по сути, алгоритм правильного пользования устройством.
2. Правила дорожного движения однозначно регламентируют поведение всех участников движения. Когда водитель соблюдает правила, он действует по определенному алгоритму.
3. Чтобы запустить массовое промышленное производство, нужно определить порядок (алгоритм) сборки продукции на конвейере. Алгоритм сборки — это набор действий, в результате выполнения которых мы получаем продукт.

Способы представления алгоритма

Разработку алгоритма можно условно разбить на три этапа:

- обдумывание общей идеи алгоритма,
- формализация идей,
- написание программного кода — реализация алгоритма для конкретного исполнителя.

На каждом этапе алгоритм принимает разную форму. Сначала мы описываем ход решения задачи словами. Это называется «словесное представление». Затем мы постепенно конкретизируем алгоритм (создаём псевдокод, блок-схему) и в итоге приходим к цельной конкретизированной реализации алгоритма — описываем его с помощью алгоритмического языка.

Итак, **формы представления алгоритма**:

- словесная — на естественном языке;
- псевдокод — описание алгоритма на искусственном (формальном) языке с использованием зарезервированных ключевых слов. В псевдокоде отсутствуют несущественные для понимания алгоритма человеком детали: описания переменных, системно-зависимый код, директивы компилятора и др.;
- блок-схема — графическое представление;
- код программы — на алгоритмическом языке программирования.

Чтобы понять, как это работает, решим задачу — найдем частное двух чисел.

Сначала опишем порядок обработки данных словами:

- Шаг 1. Получаем на вход два числа, одно из них делимое, второе — делитель.
- Шаг 2. Проверяем, не нарушается ли математическое правило деления на нуль.
- Шаг 3. Если правило деления на нуль не нарушено и делитель не равен нулю, рассчитываем частное и запоминаем ответ.
- Шаг 4. Если правило деления на нуль нарушено и делитель равен нулю, рассчитывать частное мы не можем и задача не имеет решения.

Исполнитель алгоритма в данном случае — человек, который выполняет наши инструкции.

Чем плохо словесное описание? Оно лишено строгой формализации: высказывания могут быть избыточными и даже противоречить друг другу. Мы не достигаем свойств алгоритма, о которых говорили выше.

Теперь посмотрим на псевдокод. Это симбиоз словесной и программной форм. Плюс в том, что мы пишем алгоритм на языке, близком к естественному, но при этом можем пользоваться формальными конструкциями и математическими символами.

У псевдокода нет строгих синтаксических правил, поэтому он удобен на этапе проектирования алгоритма. От такого проекта легко перейти к реализации алгоритма на конкретном формальном языке. Поэтому псевдокод — промежуточное звено между описанием алгоритма для человека и описанием для вычислительного исполнителя.

Система команд псевдокода в табл. 1.1.

Название структуры	Псевдокод
Присваивание	=
Ввод	Ввод (данные)
Вывод	Вывод (данные) Вывод ("строка")
Ветвление	Если условие то действие иначе действие
Повторение	Пока условие начало пока действие конец пока

Представим алгоритм решения нашей задачи в псевдокоде:

Начало

вывод ("введите два числа")

ввод (первого числа)

ввод (второго числа)

если (второе число) делитель \neq нуля

то частное = (первое число) делимое / (второе число) делитель

вывод ("Результат: =" частное)

иначе вывод ("нет решения")

конец алгоритма



Мы используем три переменные. Первая хранит делимое, вторая — делитель, третья — частное. Делимое и делитель вводятся с клавиатуры. Частное считаем только при условии, что не нарушается правило деления на ноль.





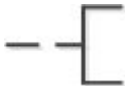
Чтобы выбросить из псевдокода всё лишнее, представим алгоритм в виде блок-схемы.

Блок-схема

Блок-схему строят из графических функциональных блоков. Каждое действие алгоритма представляют отдельной геометрической фигурой. Чтобы показать последовательность выполнения действий, блоки связывают соединительными линиями.

Это позволяет наглядно описать алгоритм человеку, даже незнакомому с предметной областью, в которой работает алгоритм. Элементы блок-схемы определены стандартом «ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».

Таблица 1.2. Базовые элементы блок-схемы		
Название	Обозначение	Функция
Терминатор		Определяет начало и конец алгоритма.
Процесс		Определяет процесс выполнения одного или нескольких действий (шагов).

Решение		Определяет ход алгоритма (решение) и отображает альтернативные ветви алгоритма.
Предопределенные процесс		Определяет функции (отдельные подпрограммы основного алгоритма).
Данные		Определяет действия ввода/вывода данных.
Соединитель		Определяет переход между частями блок-схемы, когда необходимо ее разбить на части.
Комментарий		Пояснение какого-либо действия алгоритма.

Исполнители алгоритма

Исполнитель алгоритма — это абстрактная или реальная (техническая, биологическая, биотехническая) система, способная выполнить предписанные в алгоритме действия. Исполнителями могут быть физические устройства или специальные программы — среды разработки.

Исполнителя характеризуют:

- среда разработки алгоритма (JetBrains PyCharm, JetBrains IDEA Community и т.д.);
- простые действия;
- система команд (язык программирования: Java, Python, C++ или др.);
- отказы.

Виды алгоритмов

По способу организации алгоритмы бывают трёх видов:

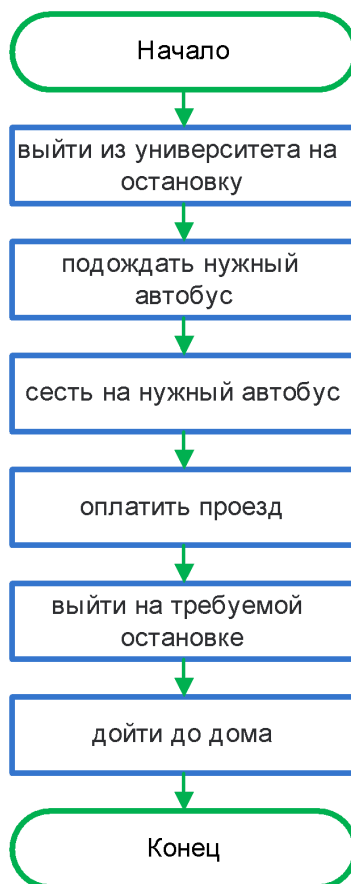
1. Линейный. Пример: движение по карте из одной точки в другую.
2. Разветвляющийся. Пример: «Чай или кофе?»
3. Циклический. Пример: работа часов.

Разберем каждый вид алгоритма подробнее.

Линейный алгоритм

В линейном алгоритме, который еще называют следованием, все действия выполняются одно за другим и только один раз.

Когда вы прокладываете путь из пункта А в пункт Б, вы создаёте линейный алгоритм. Давайте представим алгоритм, по которому студент возвращается домой из университета.



Как убедиться, что это линейный алгоритм? Все действия следуют одно за другим, выполняются однократно и без условий.

Разветвляющийся алгоритм

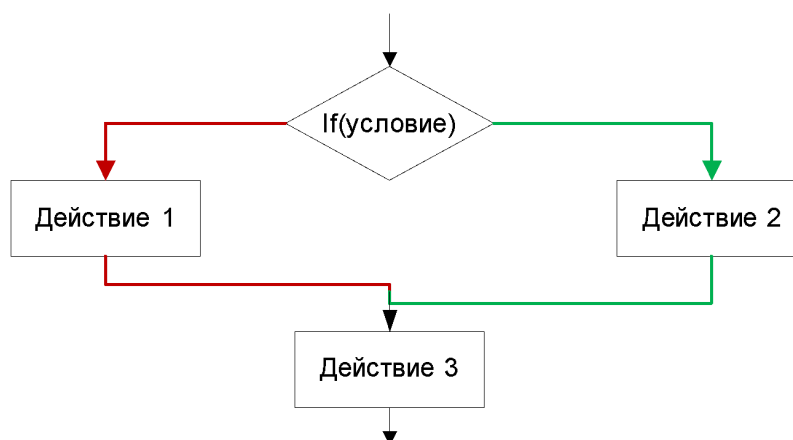
Что надеть, что съесть, на каком языке программировать — мы постоянно выбираем. И в каждом выборе, который мы делаем под влиянием условий, заложен алгоритм действий. Если на улице снегопад и большие пробки, мы оставляем автомобиль и едем общественным транспортом. В программировании ситуацию выбора описывают с помощью разветвляющегося алгоритма.

Разветвляющийся алгоритм предполагает выполнение одной или другой последовательности действий, в зависимости от условия.

Конструкция «Ветвление» обеспечивает выбор одного из альтернативных путей работы алгоритма по результатам проверки заданного условия.

Практически во всех языках программирования для реализации ветвления используются операторы **if** («если») и **else** («иначе»).

Классическая конструкция «Ветвление» в виде блок-схемы:



Простой пример разветвляющегося алгоритма: если на улице идет снег, надеть зимнюю шапку, иначе — надеть кепку.

Но условные конструкции могут выглядеть и по-другому. Бывает выбор без альтернативы, когда действие выполняется только в одном случае. Это называется одиночным условием. Если снег — надеть шапку. Соответственно, если нет, ничего не делаем.

А ещё условия можно вкладывать одно в другое. Проверяем внешнее и, если оно истинно, проверяем вложенное. Пример: Если идет снег, надеть шапку, а если ещё и сильный ветер — дополнительно надеть шарф. Можно связывать два условия в одно с помощью логических операторов (AND, OR). Если снег И ветер, надеть шапку и шарф.

Алгоритм деления из нашей задачи тоже разветвляется, когда проверяет, не равен ли делитель нулю.

Циклический алгоритм

Достижениям современной техники пока не удалось окончательно оторвать нас от природы и её процессов. Мы по-прежнему радуемся приходу весны и ясной погоде. Наш организм формируется в 24-часовом суточном ритме, и мы воспринимаем это как данность. В основе нашего естественного ритма жизни — повторяющиеся действия: ночью спим, днем — бодрствуем, вечером отдыхаем. Чтобы сделать жизнь интересной и разнообразной, в каждом новом цикле мы можем делать что-то новое.

Обратите внимание: сам процесс измерения времени основан на периодической повторяемости физических процессов, которые, по наблюдению человека, всегда длятся одинаково долго. Механические часы отсчитывают колебания маятника, а маятник имеет свойство совершать колебания за одно и то же время (при выполнении некоторых условий).

В программировании есть свой механизм, который моделирует периодическое повторение процессов. Это цикл.

Команды циклического алгоритма повторяются заданное количество раз подряд. В циклическом алгоритме важно прописать условие его завершения, иначе цикл будет выполняться бесконечно.

Давайте закрепим изученное на практике и разработаем несколько алгоритмов.

Закрепляем изученное

Задача 1. Сравниваем числа и находим максимум

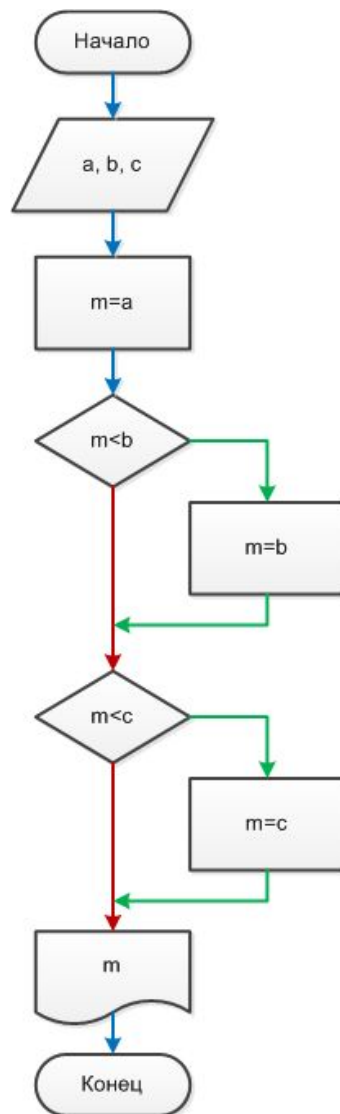
Из трёх целых чисел нужно выбрать наибольшее.

Обозначим исходные числа переменными A, B и C. Затем создадим переменную m, куда будет записан результат сравнения.

Теперь составим словесное описание алгоритма:

- Шаг 1. Сделаем первое предположение: переменная A содержит наибольшее значение. Тогда присвоим переменной m значение переменной A.
- Шаг 2. Сделаем второе предположение: если текущее значение $m < B$, присвоим m значение B.
- Шаг 3. Сделаем третье предположение: если текущее значение $m < C$, присвоим m значение C. Если нет, значение m не меняем.

Нарисуем блок-схему алгоритма:



Программный код по блок-схеме (на языке Python):

```
a = int(input('Первое число: '))
b = int(input('Второе число: '))
c = int(input('Третье число: '))

m = a
if m < b:
    m = b
if m < c:
    m = c

print(m)
```

Результат работы программы:

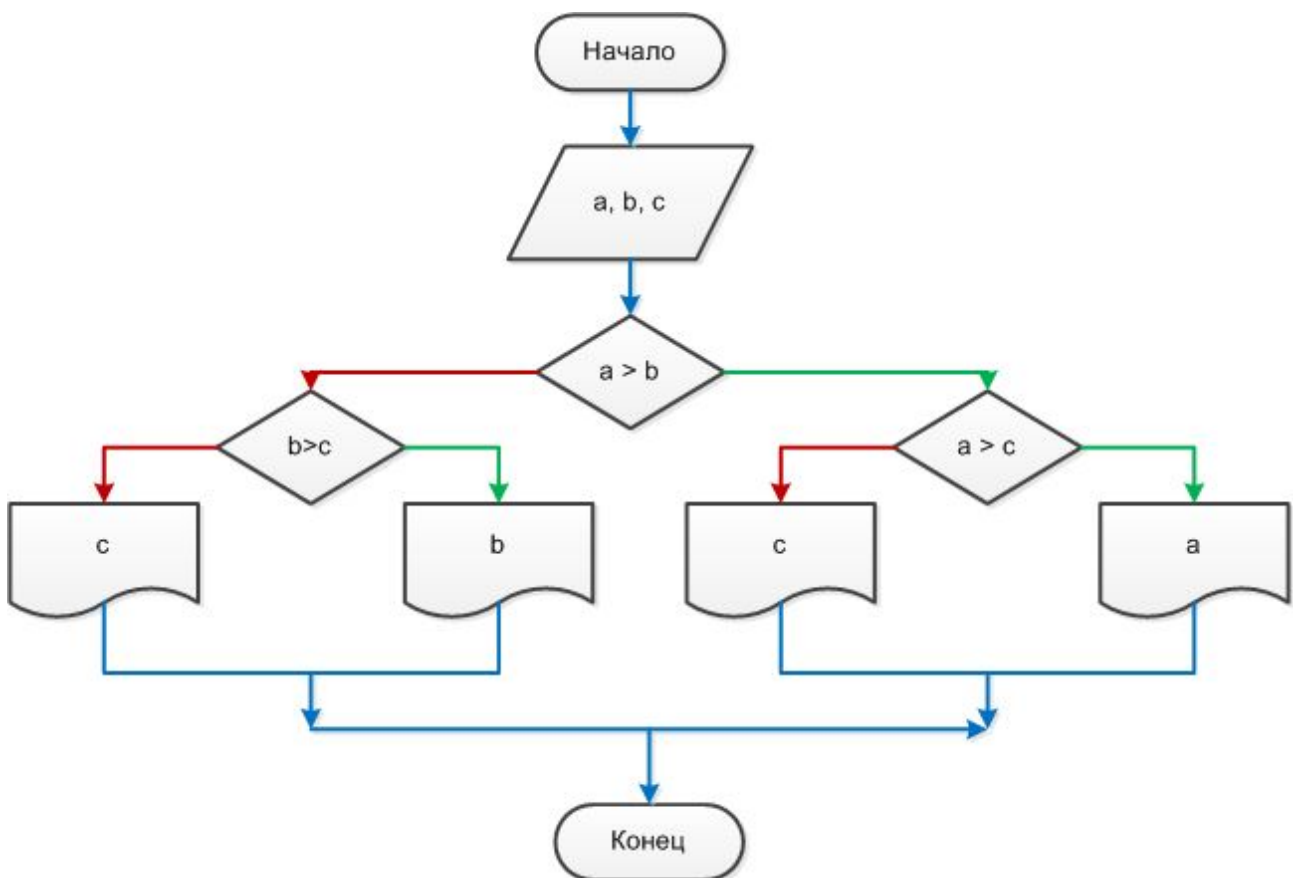
```
Run Exp1(Max1)
C:\Users\UAI-311\AppData\Local\Programs\Python\Python36\python.exe
34
67
3
67
Process finished with exit code 0
```

Задачу можно решить и без дополнительной переменной m , но программа будет логически сложнее. Алгоритм будет таким:

Шаг 1. Сделаем первое предложение: если $A > B$, то проверить $A > C$. Если это так, значение переменной A — максимум. Если $A > B$, но $A < C$, то максимальным является значение C .

Шаг 2. Иначе (когда $A < B$) сравним значения B и C . Большее из них и есть максимальное.

Блок-схема алгоритма:



Программный код по блок-схеме:

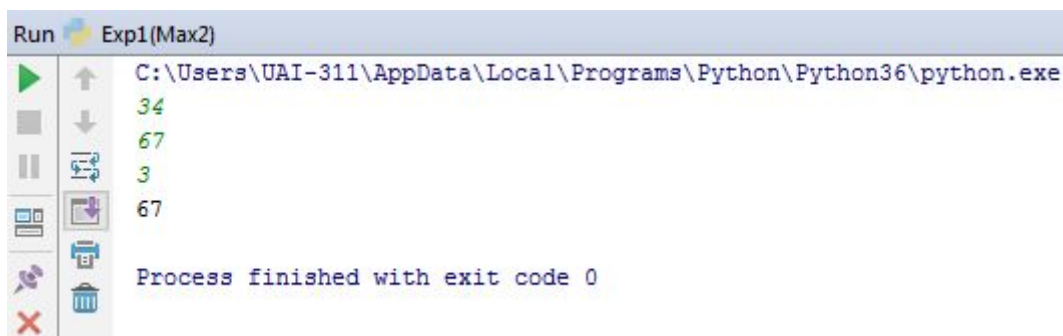
```

a = int(input('Первое число: '))
b = int(input('Второе число: '))
c = int(input('Третье число: '))

if a > b:
    if a > c:
        print(a)
    else:
        print(c)
else:
    if b > c:
        print(b)
    else:
        print(c)

```

Результат работы программы:



Задача 2. Вычислить значение функции $y=f(x)$

Дана функция $y=f(x)$:

$y = 2x - 10$, если $x > 0$

$y = 0$, если $x = 0$

$y = 2 * |x| - 1$, если $x < 0$

Требуется найти значение функции по введенному значению X .

Алгоритм решения задачи сводится к следующим шагам:

Шаг 1. Пользователь вводит значение X .

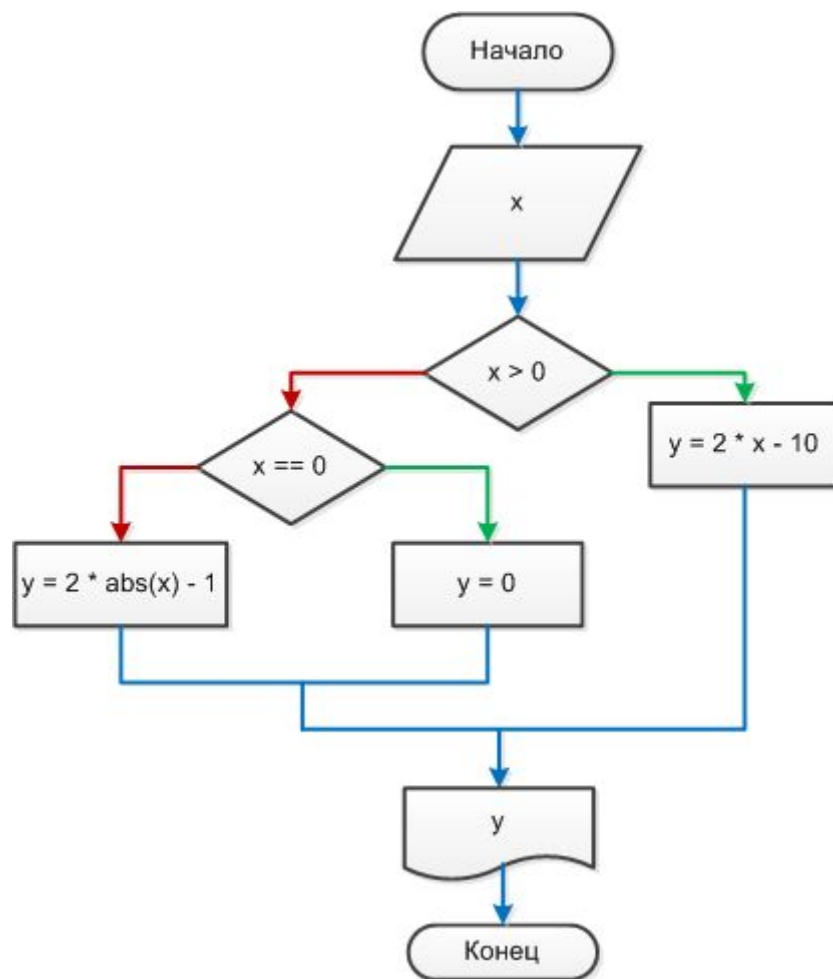
Шаг 2. Если $X > 0$, вычислить $y = 2*x-10$.

Шаг 2.1. Если $X = 0$, то $y = 0$.

Шаг 2.2. $y = 2*|x|-1$.

Шаг 3. Выводим значение y на экран.

Блок-схема алгоритма:



Программный код по блок-схеме:

```

x = input('x = ')
x = int(x)

if x > 0:
    y = 2 * x - 10
elif x == 0:
    y = 0
else:
    y = 2 * abs(x) - 1

print(y)

```

Результат работы программы:

```

Run Exp2
C:\Users\UAI-311\AppData\Local\Programs\Python\Python36\python.exe
90
170
Process finished with exit code 0

```

Задача 3. Проверим делимость одного числа на другое

Пользователь будет вводить с клавиатуры два числа, для хранения которых потребуются переменные. После ввода нужно проверить, делится ли первое введенное число на второе и, если да, вывести результат на экран. Учтите, что результат деления может быть с остатком.

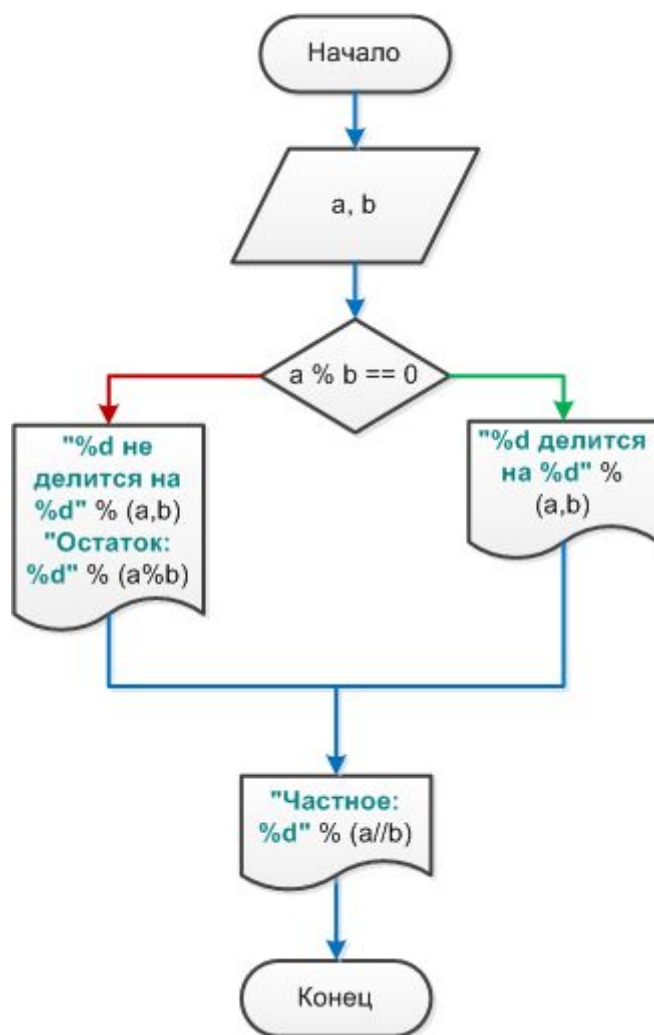
Алгоритм решения задачи в словесном представлении:

Шаг 1. Если первое число делится на второе без остатка, вывести сообщение об этом.

Шаг 2. Иначе вывести сообщение о том, что первое число не делится на второе нацело, затем найти остаток от деления и вывести его.

Шаг 3. В конце программы найти частное от деления чисел и вывести его.

Блок-схема алгоритма:

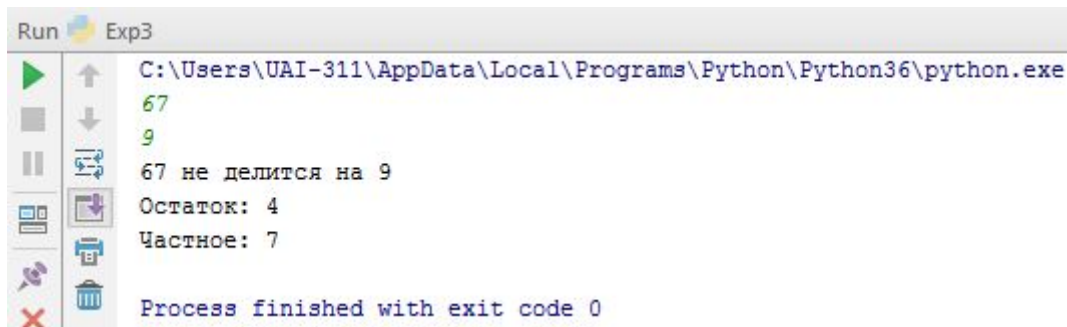


Программный код:

```
a = int(input('a = '))
b = int(input('b = '))
if a % b == 0:
    print(f"{a} делится на {b}")
else:
    print(f"{a} не делится на {b}")
```

```
print(f"Остаток: {a % b}")
print(f"Частное: {a // b}")
```

Результат работы программы:



```
Run Exp3
C:\Users\UAI-311\AppData\Local\Programs\Python\Python36\python.exe
67
9
67 не делится на 9
Остаток: 4
Частное: 7
Process finished with exit code 0
```

Задача 4. Перевести байты в килобайты или наоборот

Принять число с клавиатуры и перевести его в байты или килобайты — по выбору пользователя.

Определим изначально, что для перевода числа в байты пользователь вводит 'b', а в килобайты — 'k'.

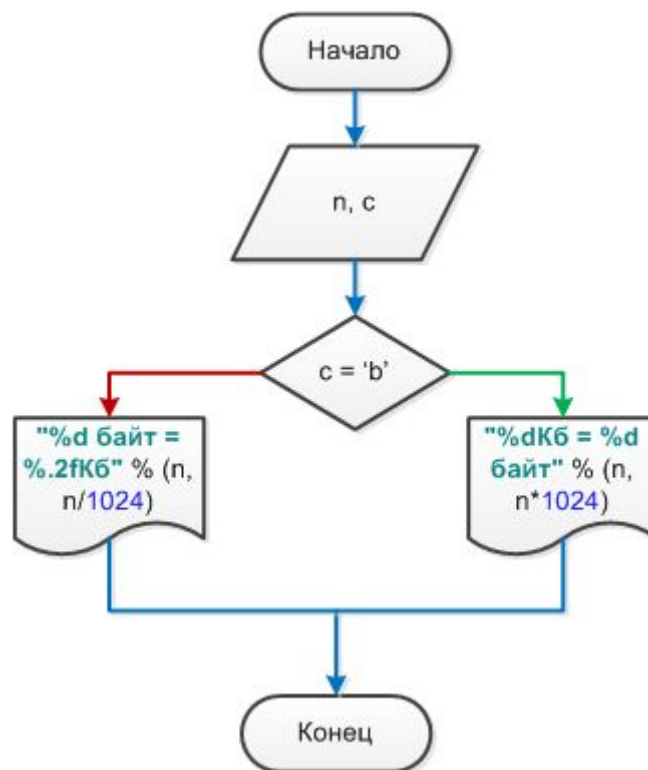
Алгоритм решения:

- Шаг 1. Пользователь вводит значение.
- Шаг 2. Пользователь выбирает единицы измерения.
- Шаг 3. Если выбран перевод в байты, умножить число на 1024.
- Шаг 4. Если выбран перевод в килобайты, разделить число на 1024.

В этой задаче пользователь может ввести что угодно, поэтому нельзя оставлять второй случай без проверки, т.е. использовать исключительно ветку `else`. Нужно использовать либо две отдельные ветки `if`, либо вложенное в `else` второе `if`-условие. Второй вариант предпочтителен с точки зрения оптимизации кода: если сработал первый `if`, второй проверяться не будет.

Ещё проще решить эту задачу, если вместо `if` использовать оператор выбора `case`.

Блок-схема алгоритма:



Программный код на Python:

```

n = int(input("Число: "))
change_type = input("Перевести в байты (b) или килобайты (k): ")
if change_type.lower() == 'b':
    print(f"{n} Кб = {n * 1024} байт")
elif change_type.lower() == 'k':
    print(f"{n} байт = {n / 1024} Кб")
  
```

Результат работы программы:

```

Run Exp4
C:\Users\UAI-311\AppData\Local\Programs\Python\Python36\python.exe
Число: 567
Перевести в байты (b) или килобайты (k): b
567Кб = 580608 байт
Process finished with exit code 0
  
```

Подведем итоги:

1. Программирование — это не просто решение практических задач, но также поиск и выбор оптимального способа решения.
2. Не изобретайте велосипед. Пользуйтесь тем, что уже разработано, и оптимизируйте алгоритм под свои задачи.
3. Когда задача решена, подумайте, как усовершенствовать код. Он может пригодиться вам и в других проектах.

Практическое задание

Для каждого упражнения составить графическое представление алгоритма в виде блок-схемы и написать программную реализацию.

1. Найти сумму и произведение цифр трехзначного числа, которое вводит пользователь.
2. Выполнить логические побитовые операции "И", "ИЛИ" и др. над числами 5 и 6. Выполнить над числом 5 побитовый сдвиг вправо и влево на два знака.
3. По введенным пользователем координатам двух точек вывести уравнение прямой вида $y = kx + b$, проходящей через эти точки.
4. Написать программу, которая генерирует в указанных пользователем границах
 - случайное целое число,
 - случайное вещественное число,
 - случайный символ.

Для каждого из трех случаев пользователь задает свои границы диапазона. Например, если надо получить случайный символ от 'a' до 'f', то вводятся эти символы. Программа должна вывести на экран любой символ алфавита от 'a' до 'f' включительно.

5. Пользователь вводит две буквы. Определить, на каких местах алфавита они стоят, и сколько между ними находится букв.
6. Пользователь вводит номер буквы в алфавите. Определить, какая это буква.
7. По длинам трех отрезков, введенных пользователем, определить возможность существования треугольника, составленного из этих отрезков. Если такой треугольник существует, то определить, является ли он разносторонним, равнобедренным или равносторонним.
8. Определить, является ли год, который ввел пользователем, високосным или не високосным.
9. Вводятся три разных числа. Найти, какое из них является средним (больше одного, но меньше другого).

Примечания ко всем задачам урока:

1. Решите задачу при помощи линейного алгоритма или алгоритма с ветвлением (цикл и рекурсии будут на уроке 2 и тут их не используем для решения).
2. Аналогично п. 1. массивы пройдем на уроке 3, поэтому постарайтесь решить задачи без них.
3. Если речь идет о символах, используйте только строчные буквы английского алфавита.

Дополнительные материалы

1. <http://www.intuit.ru/studies/courses/10/320/info>

Используемая литература

1. <https://www.python.org>
2. <http://www.intuit.ru/studies/courses/10/320/info>
3. Марк Лутц. Изучаем Python, 4-е издание