# Глава 5 Алгоритмы и программирование с помощью программ невизуального доступа к информации

## 5.1. Циклические алгоритмы

Вы уже хорошо знакомы с понятием алгоритма. Это одно из важнейших понятий информатики. Прежде, чем выполнить какую-либо операцию на компьютере или написать программу, необходимо разработать алгоритм для решения поставленной задачи.

Алгоритмы состоят из отдельных команд, которые исполнитель выполняет одну за другой в определённой последовательности. Разделение информационного процесса в алгоритме на отдельные команды является важным свойством алгоритма и называется дискретностью. Другими словами, прежде, чем приступать к следующему шагу алгоритма, необходимо завершить выполнение предыдущего шага.

В алгоритме должны быть определены начальное состояние объекта и его конечное состояние (цель преобразования). Алгоритм преобразует объект из начального состояния в конечное за конечное число шагов. Такое свойство алгоритма называется результативностью или конечностью.

Запись алгоритма должна быть такова, чтобы, выполнив очередную команду, исполнитель точно знал, какую команду необходимо исполнять следующей. Это свойство алгоритма называется детерминированностью.

Таким образом, основные свойства алгоритмов следующие:

1. Понятность для исполнителя — исполнитель алгоритма должен понимать, как его выполнять. Иными словами, имея алгоритм и произвольный вариант исходных данных, исполнитель должен знать, как надо действовать для выполнения этого алгоритма.

2. Дискретность (прерывность, раздельность) — алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определённых) шагов (этапов).

3. Детерминированность (Определённость) — каждое правило алгоритма должно быть чётким, однозначным и не оставлять места для произвола. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

4. Результативность (или конечность) состоит в том, что за конечное число шагов алгоритм либо должен приводить к решению задачи, либо после конечного числа шагов останавливаться из-за невозможности получить решение с выдачей соответствующего сообщения.

5. Массовость означает, что алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

Разрабатывая алгоритм выполнения определённой операции на компьютере вы , как правило, использовали линейную алгоритмическую структуру, а в более сложных случаях – структуру ветвления. С этими структурами вы хорошо знакомы. Для решения многих задач необходимо многократное повторение отдельных участков вычислений. В этом случае применяются циклические алгоритмы.

Цикл – это последовательность команд, которая повторяется до тех пор, пока не будет выполнено заданное условие. Циклическое описание многократно повторяемых процессов значительно снижает трудоёмкость написания программ.

Например, Алгоритм перевода десятичного числа в двоичное циклический.

Алгоритмическая структура цикл состоит из управляющей конструкции и многократно повторяющегося набора инструкций, называемого телом цикла.

В языке Python Циклические алгоритмические конструкции бывают двух типов:

* циклы с параметром (или со счётчиком), в которых тело цикла выполняется заранее заданное количество раз;
* циклы с условием, в которых тело цикла выполняется, пока это условие истинно.

В следующих параграфах этой главы будет подробно рассмотрена практическая реализация этих циклов на языке Python.

В качестве примера циклического алгоритма рассмотрим алгоритм Евклида. Этот алгоритм предназначен для нахождения наибольшего общего делителя (НОД) двух натуральных чисел.

Алгоритм может иметь следующую запись:

1. Ввести два натуральных числа.

2. Если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма.

3. Определить большее из чисел.

4. Заменить большее из чисел разностью большего и меньшего.

5. Повторить алгоритм с шага 2.

Описанный алгоритм применим к любым натуральным числам и должен приводить к решению поставленной задачи. Убедитесь в этом самостоятельно, определив с помощью этого алгоритма наибольший общий делитель чисел 125 и 75.

### Контрольные вопросы

1. В чем состоит различие между естественными языками и языками программирования?
2. Используя известные вам математические алгоритмы, приведите примеры:  
   А) линейного алгоритма;  
   Б) ветвящегося алгоритма;  
   В) циклического алгоритма.
3. Какие свойства алгоритмов вы знаете?
4. В чём состоит свойство дискретности?
5. В чём состоит свойство массовости?
6. Из чего состоит циклическая алгоритмическая структура?
7. Что является телом цикла в алгоритме Евклида? А управляющей конструкцией?

## 5.2. Цикл с параметром

В языке Python цикл с параметром обеспечен необходимыми возможностями для быстрого и эффективного программирования циклических алгоритмов. Для задания этой алгоритмической конструкции используется ключевое слово for. Это же ключевое слово используется и в большинстве других языков программирования, поэтому цикл с параметром часто называют «цикл for».

В первой (управляющей) строке цикла после ключевого слова for указывается имя переменной и последовательность или диапазон значений, которые эта переменная будет принимать. Заканчивается эта строка двоеточием.

Непосредственно после управляющей строки цикла с одинаковым отступом от начала строки записываются инструкции тела цикла. Инструкции, образующие тело цикла, объединяются созданием одинакового отступа в начале каждой из объединяемых строк. Так же, как и в условной инструкции if, здесь делают отступ из четырёх пробелов.

Напомним, что строки символов в языке Python относятся к категории последовательностей, в том смысле, что символы, которые они содержат, имеют определённый порядок следования слева направо. Строка - это представитель класса объектов, называемых последовательностями.

Цикл с параметром обладает возможностью «перебирать» значения некоторой последовательности. Рассмотрим эту возможность на примерах.

Пример 1. Программа выводит на экран символы некоторой строки по одному.

s ='Привет!'

for ch in s:

print(ch)

На первом шаге цикла переменная ch содержит первую букву строки «Привет!», т.е. букву «П». Именно эта буква и будет выведена на экран инструкцией print() тела цикла. На втором шаге цикла в переменной ch будет вторая буква «р» и в теле цикла она выводится на экран. Так перебор строки (последовательности) продолжается семь шагов, т.е. цикл срабатывает семь раз. На последнем шаге в переменную ch помещается символ «!» (восклицательный знак). После вывода его на экран цикл завершается и, если в программе есть ещё инструкции после цикла, выполняются они. В примере 1 после окончания цикла других инструкций нет и программа завершается.

Заметим, что в языке Python нет специального типа для хранения единственного символа и одиночные символы хранятся как строки длинны 1.

Обратите внимание, что при выполнении программы примера 1, буквы слова «Привет!» выводятся с новой строки каждая, поскольку инструкция print() вызывается для каждой из них заново.

Пример 2. Программа последовательно выводит на экран названия цветов радуги.

colors ='Красный', 'Оранжевый', 'Жёлтый', 'Зелёный', 'Голубой', 'Синий', 'Фиолетовый'

for c in colors:

print(c)

В этом примере переменная c будет содержать на каждом шаге цикла название одного из цветов радуги. Например, на третьем шаге в ней будет находится слово «Жёлтый». После того, как параметр цикла (переменная c) примет последовательно все семь значений, цикл завершит свою работу.

Цикл for (цикл с параметром) чаще всего используется для перебора и обработки элементов строки или другой последовательности, а также для выполнения группы инструкций (тела цикла) заданное число раз. Параметр цикла может принимать значения из некоторого диапазона целых чисел. При таком использовании параметр цикла изменяется от наименьшего до наибольшего значения с определённым шагом.

Для повторения цикла заданное количество раз следует использовать в его управляющей строке функцию range().

Пример 3. Программа выводит на экран таблицу квадратов целых чисел от 0 до 9.

for i in range(10):

print('Квадрат числа', i, 'равен', i\*\*2)

В теле цикла внутри скобок функции print() расположены четыре аргумента, разделённые запятой. Сперва идёт строка ‘Квадрат числа’, затем переменная i, затем строка ‘равен’ и, наконец, арифметическое выражение i\*\*2 (I в квадрате). Благодаря использованию функции range() в этом примере переменная i принимает последовательно значения от 0 до 9, т.е. на первом шаге цикла I =0, на втором шаге I =1 и т.д. А на последнем шаге I =9.

Обратите внимание, что указанное в качестве аргумента функции range() число 10 не попадает в переменную i. Данная конструкция работает так, что параметр цикла I принимает целые неотрицательные значения, меньшие числа 10. Если в качестве аргумента указан 0 или отрицательное число, то тело цикла не выполнится ни разу.

Функция range() может также принимать два аргумента. Вызов range(a, b) означает, что параметр цикла будет принимать значения от a до b -1, т.е. первый аргумент функции range(a, b) задаёт начальное значение параметра цикла, а второй аргумент задаёт первое по порядку целое значение, которое параметр цикла принимать не будет. Если a >=b, то цикл не будет выполнен ни разу.

Пример 4. Программа запрашивает натуральное число и выводит на экран сумму всех натуральных чисел от 1 до введённого значения.

n =int(input('Введите натуральное число '))

sum =0

for i in range(1, n +1):

sum =sum +i

print('Сумма равна', sum)

Поясним работу этой программы:

1. переменная i принимает значения от 1 до n, поскольку второй аргумент функции range() равен n +1.

2. В начале программы в переменную sum положено значение 0. Эта процедура называется инициализация переменной. Программист должен следить за тем, чтобы перед первым использованием каждой переменной было присвоено какое-либо значение. Переменная n здесь инициализируется при первом использовании.

3. При выводе приглашения «Введите натуральное число» перед закрывающем апострофом имеется пробел. Если его не написать, то при работе программы на экране введённое пользователем число не будет отделено от приглашения пробелом (проверьте это с помощью брайлевского дисплея).

4. Инструкцию тела цикла sum =sum +i можно записать в краткой форме sum +=i. При такой записи к значению переменной sum на каждом шаге цикла также будет добавляться значение переменной i.

5. Инструкция print('Сумма равна', sum) записана без отступа (без пробелов в начале строки), поэтому она выполняется один раз после окончания работы цикла.

Чтобы организовать цикл, в котором параметр будет увеличиваться с каким-либо другим шагом или уменьшаться, необходимо использовать функцию range() с тремя аргументами. Первый аргумент задаёт начальное значение параметра цикла, второй аргумент задаёт значение, до которого будет изменяться параметр цикла (не включая его), а третий аргумент — величину изменения параметра (шаг цикла). Например, сделать цикл по всем нечётным числам от 1 до 99 можно при помощи функции range(1, 100, 2), а сделать цикл по всем числам от 100 до 1 можно при помощи range(100, 0, -1).

Более формально, цикл

for i in range(a, b, d):

при d > 0 задаёт значения параметра цикла i =a, i =a +d, i =a +2\*d и так для всех значений, для которых i < b. Если же d < 0, то параметр цикла принимает все значения i > b.

Пример 5. Программа запрашивает натуральное число от 1 до 99 и выводит на экран сумму всех натуральных чисел, кратных введённому значению и непревосходящих 99.

d =int(input('Введите натуральное число от 1 до 99 '))

sum =0

for i in range(d, 100, d):

sum +=i

print('Сумма равна', sum)

Пример 6. Программа вычисляет факториал натурального числа и выводит результат на экран. Напомним, что факториалом называется произведение всех натуральных чисел от 1 до данного. Например, факториал числа 3 равен 1\*2\*3 =6.

n =int(input('Введите натуральное число '))

f =1

for i in range(2, n +1):

f \*=i

print('Факториал числа', n, 'равен', f)

Пример 7. Программа запрашивает натуральное число и выводит сообщение «Число простое», если введённое число было простым, и выводит сообщение «Число составное» в противном случае.

n =int(input('Введите натуральное число '))

p =False

for i in range(2, n):

if n%i ==0:

p =True

if p:

print('Число составное')

else:

print('Число простое')

Обратите внимание, что логические значения «True» и «False» пишутся с большой буквы. Язык программирования Python различает большие и малые буквы при написании текста программы.

### Контрольные вопросы

1. Как реализуется цикл с параметром на языке Python?
2. Что такое параметр цикла?
3. Что такое последовательность в Python? Приведите примеры.
4. Какие значения может принимать параметр цикла?
5. Как используется функция range()?
6. Сколько аргументов может иметь функция range()? Расскажите о каждом из них.
7. Расскажите как работает программа вычисления суммы чисел, кратных данному.
8. Расскажите как работает программа вычисления факториала натурального числа.
9. Почему в программе вычисления факториала цикл начинается с числа 2?
10. Почему в программе вычисления факториала переменная f проинициализирована числом 1?
11. Расскажите как работает программа, отличающая простое число от составного.

## 5.3. Цикл с проверкой условия

Второй вариант реализации циклической конструкции на языке программирования Python – это циклы с условием, в которых тело цикла выполняется, пока это условие истинно. В языке программирования Python (как и во многих других языках) этот цикл начинается с ключевого слова while.

Цикл while (пока) позволяет выполнять определённый блок инструкций (тело цикла), пока проверяемое условие истинно. Условие записывается в управляющей строке после ключевого слова while и проверяется до выполнения тела цикла. Как правило, цикл while используется, когда невозможно определить точное значение количества проходов цикла.

Цикл while в языке Python представляет собой более универсальный инструмент выполнения циклически повторяющихся инструкций и не имеет прямой связи с последовательностями. Рассмотрим его работу на примерах.

Пример 1. Программа спрашивает у пользователя: «Сколько дней в апреле?» и проверяет ответ. Если ответ не правильный программа сообщает об этом и снова задаёт тот же вопрос. Если ответ правильный программа выводит соответствующее сообщение и завершается.

answer =int(input('Сколько дней в апреле? '))

while answer !=30:

print('Это неправильный ответ!')

answer =int(input('Сколько дней в апреле? '))

print('Правильно!')

Пример 2. Программа выводит на экран треугольник из символов «\*» (звезда). В первой строке выводится 5 звёзд, во второй – 4 и т.д. В последней пятой строке выводится одна звезда.

n =5

while n > 0:

print('\*'\*n)

n -=1

Синтаксис цикла while в простейшем случае выглядит так:

while условие:

Тело цикла (блок инструкций с отступом)

При выполнении цикла while сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передаётся на следующую инструкцию после тела цикла. Если условие истинно, то выполняется тело цикла, после чего условие проверяется снова и опять выполняется тело цикла. Так продолжается до тех пор, пока условие истинно. Как только условие станет ложно, работа цикла завершится и управление перейдёт к инструкции, следующей за циклом.

Пример 3. Программа выводит на экран квадраты натуральных чисел от 1 до 10 (видно, что цикл while заменяет цикл for).

i =1

while i <=10:

print('Квадрат числа', i, 'равен', i\*\*2)

i +=1

В этом примере переменная i внутри цикла изменяется от 1 до 10. Такая переменная, значение которой меняется с каждым новым проходом цикла, называется счётчиком.

После выполнения этого цикла значение переменной i будет равно 11, поскольку именно при i равно 11 условие i <=10 впервые перестанет выполняться.

Пример 4. Программа определяет количество цифр в записи введённого натурального числа n.

n =int(input('Введите число '))

length =0

while n > 0:

n//=10

length +=1

print('Во введённом числе', length, 'цифр')

В этом примере программа отбрасывает по одной цифре введённого числа, начиная с младшего разряда, что эквивалентно целочисленному делению на 10 (n //= 10), при этом переменная length увеличивается на единицу при каждом проходе цикла.

В языке Python есть и другой способ решения этой задачи, использующий специальную функцию:

length =len(str(n))

Для цикла while существует возможность выполнить некоторый блок инструкций после окончания работы самого цикла. Для этого после тела цикла можно написать ключевое слово else: и после него блок инструкций, который будет выполнен один раз после окончания работы цикла, когда проверяемое условие станет ложным.

На первый взгляд, никакого смысла в этом нет, поскольку данный блок инструкций можно написать после окончания цикла. Смысл появляется только вместе с использованием инструкции break. Если во время выполнения цикла интерпретатор Python встречает инструкцию break, то он сразу же прекращает выполнение этого цикла и выходит из него. При этом ветка else: исполняться не будет.

Пример 5. Программа запрашивает целые положительные числа и вычисляет их сумму. Для завершения процесса следует ввести число 0. При вводе отрицательного числа создаётся состояние ошибки и программа аварийно завершается не выводя результат.

n =int(input('Введите целое положительное число. Введите число 0 для завершения '))

sum =0

while n !=0:

sum +=n

if n < 0:

print('Произошла ошибка! Введено отрицательное число', n)

break

n =int(input('Введите целое положительное число '))

else:

print('Сумма введённых чисел равна', sum)

Существует ещё одна инструкция управления циклами – это инструкция continue (продолжение). Если инструкция continue встречается в теле цикла, то пропускаются все оставшиеся инструкции до конца тела цикла, и исполнение цикла продолжается со следующего повторения.

Заметим, что цикл for также может иметь ветку else: и содержать инструкции break и continue внутри себя.

При программировании следует избегать излишнего употребления инструкций break и continue и использовать их только в тех случаях, когда это действительно необходимо.

Приведём два более сложных примера использования цикла while.

Пример 6. Программа запрашивает натуральное десятичное число и преобразует его в двоичную запись.

x =int(input('Введите натуральное число '))

answer ='B'

while x > 0:

answer =str(x%2) +answer

x //=2

print('Двоичная запись введённого числа', answer)

Поясним работу этой программы:

1. В теле цикла формируется строка, содержащая результат в виде символов, а не цифр. На экран она выводится после окончания работы цикла.

2. Перед циклом этой строке присваивается символ «B», символизирующий, что в ней содержится именно двоичная запись числа. При выводе на экран символ «B» окажется в конце строки с нулями и единицами. Синтаксически этот символ не обязателен.

3. При формировании строки с результатом в теле цикла в неё сперва помещается вычисленная цифра, а затем добавляется предыдущее содержимое. Так сделано потому, что используемый алгоритм перевода десятичного числа в двоичное вычисляет двоичные цифры с конца.

Обратите внимание, что результат сложения строк (конкатенация) зависит от порядка следования слагаемых.

Пример 7. Программа запрашивает натуральное число и выводит на экран его разложение на простые множители.

n =int(input('Введите натуральное число '))

k =2

answer =str(n) +' ='

while n > 1:

if n%k ==0:

answer =answer +str(k) +'\*'

n =n//k

else:

k +=1

print(answer[:-1])

В этом примере вывод строки с результатом осуществляется с помощью среза. Напомним, что такой срез отсекает последний символ в строке, поскольку, в данном примере этот символ является звёздочкой, а в конце она не нужна.

### Контрольные вопросы

1. Какой тип алгоритмической структуры следует использовать, если последовательность инструкций должна быть выполнена определённое количество раз?
2. Какой тип алгоритмической структуры следует использовать, если последовательность инструкций должна выполняться или не выполняться в зависимости от некоторого условия?
3. Объясните как работает цикл с проверкой условия.
4. Какое ключевое слово используется для организации цикла с проверкой условия?
5. Что называют счётчиком цикла?
6. Как работает ветвь else: в циклах?
7. Для чего служит инструкция break?
8. Для чего служит инструкция continue?
9. Объясните как работает программа перевода числа из десятичной системы в двоичную.
10. Объясните почему при сложении строк результат зависит от порядка следования слагаемых.
11. Объясните как работает программа разложения натурального числа на простые множители.

## 5.4. Обработка данных с помощью циклов

Циклы являются очень удобным инструментом обработки данных, содержащихся в массивах. Массив — это определённое количество элементов (переменных) одного типа, которые имеют общее имя, и у каждого элемента есть свой индекс (порядковый номер).

Заметим, что формально в языке Python отсутствует такая структура, как «массив». В качестве массивов используются списки, которые обладают не только всеми свойствами массива, но имеют и некоторые дополнительные. Однако, термин «массив» употребляется и в языке Python.

Массивы бывают одномерными и многомерными. Если у каждого элемента один индекс, то такой массив называется одномерным. Если индексов у элемента два, то массив двумерный и т.д. В этой книге рассматриваются только одномерные массивы.

Размер массива – это количество его элементов. Размер массива ограничивается только объёмом рабочей памяти компьютера.

Не путайте размерность массива (количество индексов у элемента) и размер массива (количество элементов в массиве).

Получить доступ к каждому элементу массива можно через его индекс. Например, если есть массив с именем x, то его первому элементу (с индексом 0) можно присвоить значение 5 следующим образом:

x[0] =5

Индексы элементов массива заключаются в квадратные скобки и начинаются с нуля.

Наряду с конкретным значением (константой) в качестве индекса может быть использована переменная, например, при обработке массива поэлементно с помощью цикла for с параметром i допустимо присваивание:  
x[i] =i\*i  
такое присваивание в теле цикла поместит в каждый элемент массива квадрат его индекса.

При создании массива и заполнении его элементов числами используют метод append(). Метод append() объекта массив можно представлять себе как функцию для создания элемента массива и помещения в него данного числа. Мы познакомимся с ещё одним методом массива (списка) при изучении способов сортировки элементов массива. Однако, понятие «метод» принадлежит объектно-ориентированному программированию и в этой книге обсуждаться не будет. Заметим, только, что имя метода отделяется от имени массива точкой.

Изучим приёмы создания и заполнения массивов на примерах.

Пример 1. Программа запрашивает размер (количество элементов) массива и заполняет его числами, вводимыми с клавиатуры, а затем выводит на экран элементы массива в обратном порядке.

n =int(input('Введите число элементов массива '))

x =[]

print('Введите', n, 'чисел')

for i in range(0, n):

x.append(int(input()))

print('Вот эти числа в обратном порядке')

for i in range(n -1, -1, -1):

print(x[i])

Поясним работу этой программы:

1. Вторая строка программы x =[] говорит, что в программе будет использоваться массив с именем x. Пустые квадратные скобки означают, что x – это массив (список).

2. Внутри круглых скобок метода append заключена вложенная конструкция функций для ввода чисел с клавиатуры.

3. В управляющей строке второго цикла функция range() использует три параметра, причём третий параметр (шаг) есть число отрицательное, что позволяет изменять счётчик цикла в обратном порядке от значения n -1 до 0.

Пример 2. Программа запрашивает размер массива, заполняет его случайными целыми числами от -100 до 100, отыскивает максимальный элемент массива и выводит его на экран.

import random

n =int(input('Введите число элементов массива '))

x =[]

print('Создан массив из', n, 'элементов')

for i in range(0, n):

x.append(random.randint(-100, 100))

print(x[i])

c =x[0]

for i in range(0, n):

if c < x[i]:

c =x[i]

print('Максимальный элемент', c)

В теле первого цикла второй инструкцией идёт функция print() для контрольного вывода на экран заполненного случайными числами массива.

Заметим, что во всех примерах блок программы обработки данных отделен от блока ввода (или случайного заполнения массива). Это сделано для большей наглядности программы.

Подобным образом обрабатывая массив можно отыскать, например, максимальное значение температуры при любом количестве измерений.

В заключении параграфа заметим, что элементами массивов могут быть данные и других типов, например, строки. Сама строка также может быть рассмотрена как массив символов, из которых она образована.

Вернёмся к примеру 2 из параграфа 5.2. Ту же задачу можно решить с помощью массива, элементами которого будут строки:

Mas\_colors =['Красный', 'Оранжевый', 'Жёлтый', 'Зелёный', 'Голубой', 'Синий', 'Фиолетовый']

for i in range(7):

print(mas\_colors[i])

### Контрольные вопросы

1. Что такое массивы?
2. Какая структура играет роль массивов в языке Python?
3. Что называют элементом массива?
4. Что называют индексом элемента массива?
5. Что значит «одномерный массив»?
6. Сколько индексов может быть у одного элемента массива?
7. Что такое размерность массива? А размер?
8. Сколько элементов может содержать массив?
9. Расскажите как можно заполнить числами массив.
10. Для чего служит инструкция import?
11. Для чего служит метод append?

## 5.5. Вложенные циклы

Тело цикла может содержать другие циклы. Такая конструкция называется вложенными циклами. Так же, как и в инструкции if, при вложении цикла внутрь тела другого, его отступ увеличивается на 4 пробела.

Рассмотрим работу вложенных циклов на примерах.

пример 1. Программа запрашивает натуральное число и выводит на экран все простые числа, не превосходящие введённого.

n =int(input('Введите натуральное число '))

print('Таблица простых чисел до ', n)

for i in range(2, n +1):

k =2

while i%k !=0:

k +=1

if i ==k:

print(i)

В этом примере внешним является цикл for, а внутренним цикл while. Внешний цикл перебирает все натуральные числа от 2 до n, а внутренний проверяет, является ли каждое из них простым и, если число простое, то оно выводится на экран инструкцией print() (последняя инструкция программы).

Поясним более подробно работу этой программы:

1. Внешний цикл for начинает работу со значения параметра I =2, поскольку это наименьшее простое число.

2. Первой инструкцией тела внешнего цикла является присваивание k =2, т.е. число 2 помещается в переменную k при каждом прохождении цикла for.

3. Во внутреннем цикле while переменная k содержит делители проверяемого числа, именно поэтому перед прохождением внутреннего цикла переменной k присваивается значение 2 (поскольку на 1 делится любое число).

4. Тело внутреннего цикла состоит из единственной инструкции увеличения значения k на единицу (k +=1). Следующая за циклом while инструкция if принадлежит уже телу внешнего цикла for.

5. Последняя инструкция тела внешнего цикла выводит на экран найденное простое число. Эта инструкция является частью конструкции ветвления и поэтому написана с двойным отступом (8 пробелов). После неё инструкций программы больше нет.

Перед следующим примером рассмотрим ещё одну весьма удобную особенность языка Python. В нём можно за одну инструкцию присваивания изменять значение сразу нескольких переменных. Делается это с помощью множественного присваивания:

a, b =0, 1

Слева от знака «=» (присвоить) в множественном присваивании должны стоять через запятую имена переменных, а справа должны стоять произвольные выражения, разделённые запятыми. Главное, чтобы слева и справа от знака присваивания было одинаковое число элементов. Присваивание будет осуществляться по порядку: первой переменной слева будет присвоено значение, стоящее первым справа, второй переменной – второе значение и т.д.

Множественное присваивание удобно использовать, когда нужно обменять значения двух переменных. Рассмотрим пример, использующий подобную конструкцию.

Пример 2. Программа запрашивает размер массива, заполняет его случайными целыми числами от -100 до 100 и упорядочивает его по возрастанию.

import random

n =int(input('Введите число элементов массива '))

mas =[]

print('Не сортированный массив')

for i in range(0, n):

mas.append(random.randint(-100, 100))

print(mas[i])

for i in range(0, n):

for j in range(1, n):

if mas[j -1] > mas[j]:

mas[j -1], mas[j] =mas[j], mas[j -1]

print('Упорядоченный массив')

for i in range(0,n):

print(mas[i])

Для удобства блок упорядочивания массива, состоящий из двух вложенных циклов, отделён пустыми строками от остальных частей программы.

В этом примере для упорядочивания (сортировки) массива используется алгоритм, обладающий собственным именем – «Пузырьковый алгоритм». Он состоит в том, что весь массив проходится столько раз, сколько в нём элементов, и на каждом проходе проверяются пары идущих подряд элементов. Если в проверяемой паре элементы стоят правильно, т.е. предыдущий элемент меньше или равен последующему, то алгоритм переходит к проверке следующей пары. Если же предыдущий элемент больше последующего, то они меняются местами. Таким образом, после многократного прохождения цикла все его элементы будут расположены по возрастанию.

Задачу упорядочивания массива можно решить и без вложенных циклов. Для этого в языке программирования Python предусмотрены соответствующие возможности.

Пример 3. Программа выполняет те же действия, что и в предыдущем примере, но не использует «Пузырьковый алгоритм».

import random

n =int(input('Введите число элементов массива '))

mas =[]

print('Не сортированный массив')

for i in range(0, n):

mas.append(random.randint(1, 100))

print(mas[i])

mas.sort()

print('Упорядоченный массив')

for i in range(n):

print(mas[i])

Обратите внимание, что основной блок программы, сортирующий элементы массива, здесь состоит всего из одной строки. В этой строке используется метод sort() для сортировки элементов.

### Контрольные вопросы

1. Что такое вложенные циклы?
2. Как организовать вложенный цикл?
3. Для чего нужны вложенные циклы?
4. Как вы думаете, можно ли в цикл вложить несколько других циклов? Какие варианты вложения возможны?
5. Расскажите, как работает программа, выводящая на экран все простые числа, не превосходящие введённого.
6. Расскажите, как работает программа, реализующая «Пузырьковый алгоритм».

### Практические задания

42. Запишите в редакторе Word алгоритм перевода десятичного числа в двоичное. После текста алгоритма запишите номера шагов, образующих тело цикла. Сохраните результат в файл «задание 42.docx» в свою рабочую папку.

43. Запишите в редакторе Word «Пузырьковый алгоритм». После текста алгоритма запишите номера шагов, образующих тело внутреннего цикла. Сохраните результат в файл «задание 43.docx» в свою рабочую папку.

44. Напишите программу, которая запрашивает натуральное число n и выводит на экран сумму дробей 1/1 +1/2 +1/3 + … +1/n. Выводимое значение округляется до тысячных. Текст программы сохраните в файл «задание 44.py» в свою рабочую папку.

45. Напишите программу, которая запрашивает слово и выводит на экран количество букв «а» в нём. Текст программы сохраните в файл «задание 45.py» в свою рабочую папку.

46. Напишите программу, запрашивающую натуральное число и выводящую на экран все его делители. Текст программы сохраните в файл «задание 46.py» в свою рабочую папку.

47. Напишите программу, которая запрашивает натуральное число n и вычисляет сумму факториалов всех чисел, не превосходящих n, т.е. вычисляет сумму 1! +2! +3! … +n!. Напомним, что восклицательный знак означает факториал числа. Текст программы сохраните в файл «задание 47.py» в свою рабочую папку.

48. Напишите программу, которая запрашивает натуральное число n и выводит на экран количество чисел, кратных 3 и не превосходящих n. Текст программы сохраните в файл «задание 48.py» в свою рабочую папку.

49. Напишите программу, которая запрашивает натуральное число и выводит на экран сумму цифр его десятичной записи. Текст программы сохраните в файл «задание 49.py» в свою рабочую папку.

50. Напишите программу, которая запрашивает два натуральных числа и определяет, являются ли они взаимно простыми. На экран выводится соответствующее сообщение. Текст программы сохраните в файл «задание 50.py» в свою рабочую папку.

51. Напишите программу, которая запрашивает слово и определяет, есть ли в нём удвоенная буква. На экран выводится соответствующее сообщение. Текст программы сохраните в файл «задание 51.py» в свою рабочую папку.

52. Напишите программу, которая по введённому номеру выводит на экран соответствующее число последовательности Фибоначчи. Напомним, что последовательность чисел Фибоначчи задаётся следующим образом: первое число равно1; второе число равно 1; каждое последующее число равно сумме двух предыдущих. Например, начало последовательности чисел Фибоначчи выглядит так: 1, 1, 2, 3, 5, 8, 13… Текст программы сохраните в файл «задание 52.py» в свою рабочую папку.

53. Напишите программу, которая запрашивает натуральное число и выводит на экран его минимальный делитель. Текст программы сохраните в файл «задание 53.py» в свою рабочую папку.

54. Напишите программу, которая запрашивает числа и выводит по окончании работы количество введённых чисел. Для окончания работы программы вводится число 0. Текст программы сохраните в файл «задание 54.py» в свою рабочую папку.

55. Напишите программу, которая запрашивает числа и после каждого ввода выводит на экран их произведение. Для окончания работы программы вводится число 0. Текст программы сохраните в файл «задание 55.py» в свою рабочую папку.

56. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран их среднее арифметическое. Для завершения работы программы вводится число 0. Текст программы сохраните в файл «задание 56.py» в свою рабочую папку.

57. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран максимальное из введённых чисел. Для завершения работы программы вводится число 0. Текст программы сохраните в файл «задание 57.py» в свою рабочую папку.

58. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран номер минимального из введённых чисел. Для завершения работы программы вводится число 0. Текст программы сохраните в файл «задание 58.py» в свою рабочую папку.

59. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран сумму чётных введённых чисел. Для завершения работы программы вводится число 0. Текст программы сохраните в файл «задание 59.py» в свою рабочую папку.

60. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран количество введённых чисел, больших предыдущего. Для завершения работы программы вводится число 0. Текст программы сохраните в файл «задание 60.py» в свою рабочую папку.

61. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран максимальное количество идущих подряд равных чисел. Для завершения работы программы вводится число 0. Текст программы сохраните в файл «задание 61.py» в свою рабочую папку.

62. Напишите программу, которая сначала запрашивает десять чисел в одномерный массив, а затем суммирует отдельно все положительные элементы этого массива, отдельно отрицательные элементы и выводит на экран результаты. Текст программы сохраните в файл «задание 62.py» в свою рабочую папку.

63. Напишите программу, которая заполняет двадцатиэлементный массив случайными целыми числами из диапазона от -100 до 100. Затем программа запрашивает контрольное число и определяет, присутствует ли в массиве равный ему элемент. На экран выводится соответствующее сообщение, а также весь массив для контроля результата. Текст программы сохраните в файл «задание 63.py» в свою рабочую папку.

64. Напишите программу, которая заполняет двадцатиэлементный массив случайными целыми числами из диапазона от -100 до 100, а затем определяет, каких элементов больше положительных или отрицательных. На экран выводится одно из трёх сообщений: «Положительных элементов больше», «Отрицательных элементов больше», «Положительных и отрицательных элементов одинаковое количество». Весь массив также выводится на экран для контроля результата. Текст программы сохраните в файл «задание 64.py» в свою рабочую папку.

65. Напишите программу, которая заполняет двадцатиэлементный массив случайными целыми числами из диапазона от -100 до 100, а затем определяет среднее арифметическое элементов с чётными индексами, и среднее арифметическое элементов с нечётными индексами. Результаты выводятся на экран. Текст программы сохраните в файл «задание 65.py» в свою рабочую папку.

66. Напишите программу, которая заполняет двадцатиэлементный массив случайными целыми числами из диапазона от -100 до 100, а затем определяет количество положительных групп в этом массиве. Группой называется последовательность состоящая из двух и более положительных чисел находящихся рядом. Результат выводится на экран. Текст программы сохраните в файл «задание 66.py» в свою рабочую папку.

67. Напишите программу, которая создаёт массив, содержащий 2015 неотрицательных целых чисел из диапазона от 0 до 10000. Ямой называется не крайний элемент массива, который меньше обоих своих соседей. Необходимо найти в массиве самую глубокую яму, т.е. яму, значение которой минимально. Если в массиве нет ни одной ямы, ответ считается равным 0. Например, в массиве из шести элементов, равных соответственно 4, 9, 2, 17, 3, 8, есть две ямы – 2 и 3, самая глубокая яма – 2. Текст программы сохраните в файл «задание 67.py» в свою рабочую папку.

68. Напишите программу, которая создаёт массив из 40 элементов. Элементы массива могут принимать целые значения от 0 до 10000 включительно. Затем программа находит максимальное значение среди двузначных элементов массива, не делящихся на 3. Результат выводится на экран. Если в исходном массиве нет элемента, значение которого является двузначным числом и при этом не кратно трём, то на экран выводится сообщение «Не найдено». Текст программы сохраните в файл «задание 68.py» в свою рабочую папку.

69. Напишите программу, которая создаёт массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 1000. Затем программа вычисляет среднее арифметическое элементов массива, имеющих нечётное значение. Результат выводится на экран. Если нечётных элементов в массиве нет, то выводится сообщение «Не найдено». Текст программы сохраните в файл «задание 69.py» в свою рабочую папку.

70. Напишите программу, которая создаёт массив из 40 элементов. Элементы массива могут принимать целые значения от 0 до 10000. Затем программа вычисляет сумму элементов наибольшей возрастающей последовательности подряд идущих элементов массива. Текст программы сохраните в файл «задание 70.py» в свою рабочую папку.