# **Общероссийская общественная организация инвалидов - Российский союз инвалидов Учебно-методическое пособие Программирование без визуального контроля**

## **Глава 1 Алгоритм и его свойства**

### **Параграф 1 Что такое алгоритм**

Учиться программировать мы начнем с освоения такого важного понятия как «алгоритм». С этим понятием вы уже встречались ранее, однако, здесь мы дадим более строгие формулировки и опишем свойства алгоритма. Давать определение алгоритма можно по-разному. Самым распространенным является следующее:

Алгоритм - это строго детерминированная (определенная) последовательность действий, описывающая процесс преобразования объекта из начального состояния в конечное, записанная с помощью понятных исполнителю команд.

Рассмотрим пример стиральной машины. Чтобы постирать белье вы задаете машине алгоритм: с помощью кнопок и ручек управления указываете температуру, время, скорость вращения барабана и др. Стиральная машина является исполнителем. Список команд исполнителя четко определен элементами управления машины. Никаких других команд она не поймет. В этом примере объект «грязное бельё» будет преобразован в Объект «чистое бельё».

Алгоритмы могут описывать процессы преобразования самых разных объектов. Широкое распространение получили вычислительные алгоритмы, по которым осуществляют преобразования чисел. Примерами таких алгоритмов могут служить алгоритмы выполнения арифметических действий. Например, алгоритм сложения целых чисел, алгоритм умножения дробей и т.д.

Слово «алгоритм» происходит от algorithmi — латинской формы написания имени выдающегося математика IX века аль-Хорезми, который сформулировал правила выполнения арифметических операций.

Можно дать и другое определение алгоритма:

Алгоритм – это точное предписание, однозначно определяющее вычислительный процесс, ведущий от начальных (входных) данных к результату (выходным данным).

Как видно из определений, для выполнения алгоритма нужен исполнитель. Исполнитель понимает команды алгоритма и выполняет их. При составлении алгоритма надо следить за тем, чтобы все его команды были понятны исполнителю.

Например, если вы складываете два числа в столбик, значит вы являетесь исполнителем алгоритма сложения. Если числа складывает калькулятор, то исполнитель он. Очевидно, что для человека и калькулятора алгоритмы должны быть разными.

Универсальным, очень часто используемым исполнителем является компьютер. Алгоритм для компьютера означает точное описание некоторого процесса, инструкцию по его выполнению. Разработка алгоритма является сложным и трудоемким процессом. Алгоритмизация – это техника разработки (составления) алгоритма для решения задач на ЭВМ.

Чтобы создать какую-либо компьютерную программу для решения определенной задачи, предварительно необходимо разработать алгоритм её решения, а затем уже по нему писать программу.

Еще одним важным понятием, с которым вы уже встречались ранее, является «объект». Мы не будем пытаться дать определение этому понятию, а попробуем осознать его из примеров. Объект – это то, с чем мы работаем. Например, при выполнении арифметических действий объектами являются числа. При работе на компьютере объектом может быть символ, текст, файл, папка и т.д.

В качестве иллюстрации понятий «алгоритм», «исполнитель» и «объект» рассмотрим процедуру редактирования текста. У объекта текст есть свойства, например, шрифт, которым он набран. Рассмотрим алгоритм изменения свойства «шрифт» объекта «текст».

Алгоритм изменения свойства текста необходимо разбить на операции, которые должны быть записаны в виде отдельных команд исполнителю.

Если алгоритм записан на бумаге и вы читая команды этого алгоритма выполняете их, то вы являетесь исполнителем. Когда вы используя клавиатуру даёте команды компьютеру, то исполнителем в этой ситуации уже является он. Точнее, здесь исполнителем является компьютер с запущенным текстовым редактором, в котором происходит обработка текста. Исполнитель алгоритма изменения шрифта должен быть способен выполнить образующие алгоритм команды.

Итак, пусть имеется компьютер с запущенным редактором Word и в окне редактора есть произвольный текст, набранный шрифтом Areal. Приведем алгоритм для человека (пользователя),для изменения шрифта на Times New Roman.

Название алгоритма: Изменения шрифта.

Описание входных данных: Текст в редакторе Word, набранный шрифтом Areal.

Описание выходных данных: Тот же текст в редакторе Word, набранный шрифтом Times New Roman.

1. начало.

2. Выделить весь текст введя команду Ctrl +A.

3. Вызвать диалог изменения шрифта введя команду Ctrl +D.

4. В списке шрифтов выбрать шрифт Tines New Roman.

5. Нажать клавишу Enter.

6. Конец.

Исполнитель этого алгоритма – это человек, который работает на компьютере. Обратите внимание, что не каждый человек может быть исполнителем этого алгоритма. Выполнить алгоритм может только тот, кто имеет определенные пользовательские навыки.

Команды, которые с клавиатуры подавал человек, исполнял компьютер с помощью программы Word. Компьютер понимал какие клавиши нажаты и выполнял соответствующее действие, т.е. был исполнителем. Очевидно, что компьютер исполнял команды только из списка клавиатурных команд.

При реализации вышеприведенного алгоритма каждую команду подаёт человек, а компьютер выполняет. Это происходит потому, что алгоритм составлен из команд, которые понимает человек, но не понимает компьютер. Исполнителем такого алгоритма является человек, а вводимые им клавиатурные команды исполняет уже компьютер. Т.е., компьютер понимает не сами команды алгоритма, а клавиатурные команды вводимые человеком.

Представление алгоритма с помощью особых, понятных компьютеру, команд позволяет поручить ему выполнение такого алгоритма. В этом случае компьютер будет работать автоматически, без участия человека. Говорят, что компьютер исполняет программу, реализующую алгоритм. Алгоритм, записанный на понятном компьютеру языке, называется программой, а такой язык - языком программирования. Таким образом, программа – это последовательность команд, понятных исполнителю «компьютер».

Информацию в компьютере обрабатывает процессор, следовательно, алгоритм должен быть записан на языке, понятном для процессора, т.е. на машинном языке, представляющем собой последовательности нулей и единиц.

На заре компьютерной эры, в 50-е годы XX века, программы писались на машинном языке и представляли собой очень длинные последовательности нулей и единиц. Составление и отладка таких программ было чрезвычайно трудоемким делом.

В шестидесятые - семидесятые годы для облегчения труда программистов начали создаваться языки программирования высокого уровня. С помощью таких языков можно записать алгоритм в понятном для человека виде. Такие языки программирования строились на основе использования определенного алфавита и строгих правил построения языковых конструкций (синтаксиса).

Наиболее широко распространенным типом языков программирования высокого уровня являются процедурные языки. В таких языках широко используются управляющие конструкции (операторы), которые позволяют закодировать различные алгоритмические структуры (линейную, ветвление, цикл).

Одним из первых процедурных языков программирования был Basic, созданный в 1964 году. В течение последующего времени Basic развивался, появлялись его различные версии (MSX-Basic, Бейсик-Агат, QBasic и др.). Другим широко распространенным языком программирования алгоритмического типа является Pascal. Рассмотрим подробнее основные алгоритмические конструкции: линейную, ветвления и цикл.

Существуют алгоритмы, в которых команды должны быть выполнены последовательно одна за другой. Такие алгоритмы называются линейными. Например, рассмотренный выше алгоритм изменения шрифта линейный.

Линейный алгоритм– это такой алгоритм, в котором все действия выполняются последовательно друг за другом и только один раз.

На практике часто встречаются задачи, в которых в зависимости от первоначальных условий или промежуточных результатов необходимо выполнить вычисления по одним или другим формулам. Такие задачи решаются с помощью разветвляющихся алгоритмов. Примером ветвления может служить алгоритм решения квадратного уравнения.

В таких алгоритмах выбор направления продолжения вычисления осуществляется по итогам проверки заданного условия. Ветвящиеся структуры в языках программирования задаются оператором IF (условие).

Для решения многих задач характерно многократное повторение отдельных участков вычислений. Для решения таких задач 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 трудоемкость написания программ.

Например, Алгоритм перевода десятичного числа в двоичное циклический.

### **Контрольные вопросы**

1. Какие из перечисленных ниже правил являются алгоритмами? Ответ обоснуйте:  
А) орфографические правила;  
Б) правила выполнения арифметических операций;  
В) правила техники безопасности;  
Г) правила перевода чисел из одной системы счисления в другую.

2. В чем состоит различие между естественными языками и языками программирования?

3. Используя известные вам математические алгоритмы, приведите примеры:  
А) линейного алгоритма;  
Б) ветвящегося алгоритма;  
В) циклического алгоритма.

4. Составьте алгоритм преобразования слова «информатика» в слово «форма».

### **Параграф 2 Свойства алгоритма**

Алгоритмы состоят из отдельных команд, которые исполнитель выполняет одну за другой в определенной 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.

Для реализации алгоритма должны быть определены начальное состояние объекта и его конечное состояние (цель преобразования). Алгоритм должен обеспечивать преобразование объекта 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 определенных) шагов (этапов).

3. Детерминированность (Определённость) — каждое правило алгоритма должно быть четким, 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.

### **Параграф 3 Способы записи алгоритма**

Для записи алгоритма применяются следующие изобразительные способы:

1. Словесно- формульное описание.

2. Блок-схема (схема из особых графических символов).

3. Алгоритмические языки.

4. Операторные схемы.

5. Псевдокод.

Формульно-словесный способ записи алгоритма характеризуется тем, что описание осуществляется с помощью слов и формул. Содержание последовательности этапов выполнения алгоритмов записывается на естественном профессиональном языке предметной области в произвольной форме.

Графический способ описания алгоритма (блок - схема) получил самое широкое распространение. Для графического описания алгоритмов используются схемы алгоритмов или блочные символы (блоки), которые соединяются между собой линиями связи. Каждый этап вычислительного процесса представляется геометрическими фигурами (блоками). Они делятся на арифметические или вычислительные (прямоугольник), логические (ромб) и блоки ввода-вывода данных (параллелограмм). Порядок выполнения этапов указывается стрелками, соединяющими блоки. Геометрические фигуры размещаются сверху вниз и слева на право. Нумерация блоков производится в порядке их размещения в схеме. Блок-схемы будут подробно рассмотрены в следующей главе.

Алгоритмические языки - это специальное средство, предназначенное для записи алгоритмов в аналитическом виде. Алгоритмические языки близки к математическим выражениям и к естественным языкам. Каждый алгоритмический язык имеет свой словарь. Алгоритм, записанный на алгоритмическом языке, выполняется по строгим правилам этого конкретного языка.

Операторные схемы алгоритмов. Суть этого способа описания алгоритма заключается в том, что каждый оператор обозначается буквой (например, А – арифметический оператор, Р – логический оператор и т.д.). Операторы записываются слева направо в последовательности их выполнения, причем, каждый оператор имеет индекс, указывающий порядковый номер оператора. Алгоритм записывается в одну строку в виде последовательности операторов.

Псевдокод – система команд абстрактной машины. Этот способ записи алгоритма с помощью операторов близких к операторам алгоритмических языков.

На практике наиболее распространены следующие формы представления алгоритмов:

1. словесная (запись на естественном языке);

2. Блок-схема (изображения из графических символов);

3. псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);

4. программная (тексты на языках программирования).

Для записи алгоритма приняты следующие общие правила:

1. Каждый алгоритм должен иметь имя, которое раскрывает его смысл.

2. Должны быть обозначены начало и конец алгоритма.

3. Необходимо описать входные и выходные данные.

4. Необходимо указать команды, которые позволяют выполнять определенные действия над данными.

Таким образом, можно описать следующий общий вид алгоритма:

название алгоритма;

описание данных;

начало;

команды;

конец.

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Например, Запишем алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел (алгоритм Эвклида).

Алгоритм может быть следующим:

1. Алгоритм Евклида;

2. задать два числа;

3. если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;

4. определить большее из чисел;

5. заменить большее из чисел разностью большего и меньшего из чисел;

6. повторить алгоритм с шага 3.

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.

Словесный способ не имеет широкого распространения, так как такие описания:

строго не формализуемы;

Имеют многословную запись;

допускают неоднозначность толкования отдельных команд.

Как уже говорилось, псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Псевдокод занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.

Однако в псевдокоде обычно имеются некоторые конструкции, присущие формальным языкам, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке. В частности, в псевдокоде, так же, как и в формальных языках, есть служебные слова, смысл которых однозначно определен.

Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций. Примером псевдокода является школьный алгоритмический язык в русской нотации (школьный АЯ), описанный в учебнике А.Г. Кушниренко и др. "Основы информатики и вычислительной техники" 1991. Этот язык в дальнейшем мы будем называть просто "алгоритмический язык".

Приведем список основных служебных слов алгоритмического языка:

алг - алгоритм;

нач – начало;

кон – конец;

дано – служит для описания входных данных;

сим – символьный;

лит – литерный;

лог – логический;

цел – целый;

вещ – вещественный;

таб – таблица;

арг – аргумент;

рез – результат;

если – служит для описания конструкции ветвления;

иначе – служит для описания конструкции ветвления;

пока – служит для описания циклических конструкций;

для – служит для описания циклических конструкций;

нц - начало цикла;

кц - конец цикла;

ввод – ввод данных;

вывод – вывод результата.

Запишем общий вид алгоритма с помощью этих служебных слов:

алг название алгоритма (аргументы и результаты)

дано условия применимости алгоритма

надо цель выполнения алгоритма

нач описание промежуточных величин

последовательность команд (тело алгоритма)

кон

Часть алгоритма от слова алг до слова нач называется заголовком, а часть, заключенная между словами нач и кон — телом алгоритма.

В элементе алг после названия алгоритма в круглых скобках указываются характеристики (арг, рез) и тип значения (цел, вещ, сим, лит или лог) всех входных (аргументы) и выходных (результаты) переменных. При описании массивов (таблиц) используется служебное слово таб, дополненное граничными парами по каждому индексу элементов массива.

Примеры элементов алг:

алг Объём и площадь цилиндра (арг вещ R, H, рез вещ V, S);

алг Корни Квадратного уравнения (арг вещ а, b, c, рез вещ x1, x2, рез лит t);

алг Исключить элемент (арг цел N, арг рез вещ таб А[1:N]);

алг Диагональ (арг цел N, арг цел таб A[1:N, 1:N], рез лит Otvet).

Элементы дано и надо не обязательны. В них рекомендуется записывать утверждения, описывающие состояние среды исполнителя алгоритма, например:

алг Замена (арг лит Str1, Str2, арг рез лит Text)

дано длины подстрок Str1 и Str2 совпадают

надо всюду в строке Text подстрока Str1 заменена на Str2

алг Число максимумов (арг цел N, арг вещ таб A[1:N], рез цел K)

дано N > 0

надо К — число максимальных элементов в таблице А

алг Сопротивление (арг вещ R1, R2, арг цел N, рез вещ R)

дано N > 5, R1 > 0, R2 > 0

надо R — сопротивление схемы

Здесь в элементах дано и надо записаны комментарии. Комментарии можно помещать в конце любой строки. Они не обрабатываются транслятором, но существенно облегчают понимание алгоритма.

Приведем список команд школьного алгоритмического языка:

Команда присваивания. Служит для вычисления выражений и присваивания их значений переменным. Общий вид: A :=B, где знак ":=" означает команду заменить прежнее значение переменной, стоящей в левой части, на вычисленное значение выражения, стоящего в правой части.

Например, a :=(b +c)\*sin(Pi/4);i :=I +1.

Команды ввода и вывода.

ввод имена переменных

вывод имена переменных, выражения, тексты.

Команды если и выбор. Применяют для организации ветвлений.

Команды для и пока. Применяют для организации циклов.

Пример записи алгоритма на школьном Ая:

алг Сумма квадратов (арг цел n, рез цел S)

дано n > 0

надо S =1\*1 +2\*2 +3\*3 +... +n\*n

нач цел i

ввод n;

S:=0

нц для i от 1 до n

S :=S +i\*i

кц

вывод "S =", S

кон

При записи алгоритма в словесной форме, в виде блок-схемы или на псевдокоде допускается определенный произвол при изображении команд. Вместе с тем такая запись точна настолько, что позволяет человеку понять суть дела и исполнить алгоритм.

Однако на практике в качестве исполнителей алгоритмов используются специальные автоматы — компьютеры. Поэтому алгоритм, предназначенный для исполнения на компьютере, должен быть записан на понятном ему языке. И здесь на первый план выдвигается необходимость точной записи команд, не оставляющей места для произвольного толкования их исполнителем.

Следовательно, язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке — программой для компьютера.

### **Контрольные вопросы**

1. Какие способы записи алгоритмов вы знаете?
2. В чём состоит Формульно-словесный способ записи алгоритмов?
3. Что такое алгоритмические языки?
4. Что такое операторные схемы?
5. Что такое Псевдокод?
6. Опишите основные правила записи алгоритмов.
7. Расскажите о школьном алгоритмическом языке.

### **Упражнения для самостоятельного выполнения**

1. У исполнителя УТРОИТЕЛЬ две команды, которым присвоены номера:    
1. вычти 1  
2. умножь на 3   
Первая из них уменьшает число на экране на 1, вторая – увеличивает его в три раза.  
Запишите порядок команд в программе получения из числа 3 числа 16, содержащей не более 5 команд, указывая лишь номера команд.  
(Например, программа 21211 это программа   
умножь на 3  
вычти 1  
умножь на 3  
вычти 1  
вычти 1   
которая преобразует число 1 в 4.)

2. У исполнителя ДваПять две команды, которым присвоены номера:   
1. отними 2  
2. раздели на 5   
Выполняя первую из них, ДваПять отнимает от числа на экране 2, а выполняя вторую, делит это число на 5 (если деление нацело невозможно, ДваПять отключается).   
Запишите порядок команд в программе, которая содержит не более 5 команд и переводит число 152 в число 2.   
В ответе указывайте лишь номера команд, пробелы между цифрами не ставьте. Так, для программы   
раздели на 5  
отними 2  
отними 2   
нужно написать 211. Эта программа преобразует, например, число 55 в число 7.

3. Автомат получает на вход трёхзначное число. По этому числу строится новое число по следующим правилам.  
1. Складываются первая и вторая, а также вторая и третья цифры исходного числа.  
2. Полученные два числа записываются друг за другом в порядке убывания (без разделителей).  
Пример. Исходное число: 348. Суммы: 3 + 4 = 7; 4 + 8 = 12. Результат: 127.  
Сколько существует чисел, в результате обработки которых автомат выдаст число 1715?

4. У исполнителя Аккорд две команды, которым присвоены номера:   
1. вычти x   
2. умножь на 3   
где x – неизвестное положительное число. Выполняя первую из них, Аккорд вычитает из числа на экране x, а выполняя вторую, умножает это число на 3.   
Программа для исполнителя Аккорд – это последовательность номеров команд.   
Известно, что программа 12211 переводит число 12 в число 53. Определите значение x.   
5. На вход алгоритма подаётся натуральное число N. Алгоритм строит по нему новое число R следующим образом.  
1. Строится запись числа N в троичной системе счисления.  
2. К этой записи справа дописывается остаток от деления на 3 суммы цифр полученной записи.  
3. Шаг 2 выполняется еще один раз.  
Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа N в троичной системе счисления) является троичной записью искомого числа R. Укажите минимальное число N, после обработки которого, автомат получает число, большее 123. В ответе это число запишите в десятичной системе.

6. У исполнителя Вычислитель две команды, которым присвоены номера:  
1. прибавь4  
2. умножь на 2  
Выполняя первую из них, Вычислитель прибавляет к числу на экране 4, выполняя вторую, - умножает это число на 2. Программа для исполнителя Вычислитель – это последовательность номеров команд. Сколько различных чисел может быть получено исполнителем Вычислитель из числа 1, если известно, что программа содержит 5 команд?

7. У исполнителя ТриПять две команды, которым присвоены номера:  
1. прибавь 3  
2. умножь на 5  
Выполняя первую из них, ТриПять прибавляет к числу на экране 3, а выполняя вторую, умножает это число на 5. Запишите порядок команд в программе, которая содержит не более 5 команд и переводит число 4 в число 530. В ответе указывайте лишь номера команд, пробелы между цифрами не ставьте.  
Так, для программы:  
умножь на 5  
прибавь 3  
прибавь 3  
нужно написать: 211. Эта программа преобразует, например, число 8 в число 46.

8. Исполнитель Чертежник имеет перо, которое можно поднимать, опускать и перемещать. При перемещении опущенного пера за ним остается след в виде прямой линии. У исполнителя существуют следующие команды:  
Сместиться на вектор (a, b) – исполнитель перемещается в точку, в которую можно попасть из данной, пройдя a единиц по горизонтали и b – по вертикали.  
Запись:  
Повторить 5[Команда 1 Команда 2]  
означает, что последовательность команд в квадратных скобках повторяется 5 раз.  
Чертежник находится в начале координат. Чертежнику дан для исполнения следующий алгоритм:  
Сместиться на вектор (5, 2)  
Сместиться на вектор (-3, 3)  
Повторить 3[Сместиться на вектор (1, 0)]  
Сместиться на вектор (3, 1)  
На каком расстоянии от начала координат будет находиться исполнитель Чертежник в результате выполнения данного алгоритма?

9. У исполнителя Калькулятор две команды, которым присвоены номера:  
1. прибавь 1  
2. умножь на 2  
Запишите порядок команд в программе, которая преобразует число 7 в число 130 и содержит не более 6 команд. Указывайте лишь номера команд.

10. У исполнителя Аккорд две команды, которым присвоены номера:  
1. прибавь 2  
2. умножь на x  
где x – неизвестное положительное число. Программа для исполнителя Аккорд – это последовательность номеров команд.  
Известно, что программа 12211 переводит число 1 в число 52. Определите значение x.

## **Глава 2 Блок-схемы и работа с ними без визуального контроля**

### **Параграф 1 Запись алгоритма с помощью блок-схемы**

Как уже говорилось в предыдущей главе, одним из способов записи алгоритма является графический. Графический способ представления алгоритма делает его компактным и наглядным для восприятия человеком по сравнению с другими. При графическом представлении алгоритм изображается в виде последовательности связанных между собой стрелками блоков, каждый из которых соответствует реализации какой-либо алгоритмической структуры или выполнению одного или нескольких действий (операторов).

Такой способ записи алгоритма представляет собой блок-схему, состоящую из заранее определенных элементов – блоков. Т.е. элементы алгоритма изображаются с помощью заданных геометрических фигур. На блок-схеме хорошо видна структура алгоритма, по которой исполнителю (человеку) удобно отслеживать процесс его выполнения.

В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий и окончанию алгоритма) соответствует геометрическая фигура. Геометрические фигуры (блоки) на блок схеме соединяются стрелками, определяющими очередность выполнения действий. Ниже приведено описание наиболее часто употребляемых блоков:

1. Овал – начало или конец алгоритма (рис. 1 А). Часто обозначается прямоугольником со скруглёнными углами. Ставится в самом верху и в самом низу блок-схемы для обозначения начала и конца алгоритма.

2. Параллелограмм – ввод начальных данных или вывод результата (рис. 1 Б). Внутри параллелограмма указываются имена переменных, в которые осуществляется ввод или в которых хранятся значения для вывода. Также в параллелограмме могут указываться фразы для вывода. Например, «Решений нет».

3. Прямоугольник – обозначает выполнение команд (рис. 1 В). В прямоугольнике записывается присваивание значений переменным и вычислительные формулы. В некоторых случаях в одном прямоугольнике записывают несколько действий. Например, формулы для вычисления обоих корней квадратного уравнения можно записать в один прямоугольник.

4. Ромб – проверка условия (рис. 1 Г). Ромб служит для обозначения алгоритмической конструкции «Ветвление». Внутри ромба записывается условие, в зависимости от истинности или ложности которого определяется направление дальнейшего выполнения алгоритма. Из ромба выходят две стрелки, одна имеет подпись «Да», а другая – «Нет». Если условие истинно, то дальнейшее выполнение алгоритма идет по стрелке, помеченной «Да», а если ложно, то по стрелке «Нет».

5. Шестиугольник – модификация (рис. 1 Д). Слово «модификация» означает видоизменение, преобразование. Этот блок обозначает алгоритмическую циклическую структуру. Иначе говоря, используется для организации циклических конструкций. Внутри блока записывается параметр цикла, для которого указываются его начальное значение, конечное значение и шаг изменения значения параметра для каждого прохождения цикла. На уровне школьной программы этот блок используется очень редко, только в случае решения задач повышенной сложности.

Пример 1. Простейшая блок-схема. На рис. 2 изображена блок-схема простого алгоритма, содержащего все основные блоки, кроме шестиугольника.

Вверху этой блок-схемы расположен овал со словом «Начало» внутри. Этот блок означает начало алгоритма.

Идущая из середины овала стрелка вниз указывает на параллелограмм с переменной x внутри. Этот блок означает ввод числового значения в переменную x.

От параллелограмма вниз идет стрелка, указывающая на ромб с условием x > 0 внутри.

От левой вершины ромба влево и вниз идет стрелка, подписанная словом «Нет», а вправо и вниз идет стрелка с обозначением «Да». Это значит, что если условие в ромбе истинно, то дальнейшее выполнение алгоритма идёт по правой стрелке со словом «Да», а если ложно, то по левой стрелке со словом «Нет». Например, если в x было введено значение 5, то условие x > 0 истинно и далее следует идти по правой стрелке со словом «Да».

Стрелка «Да» указывает на прямоугольник с присваиванием y =2x, т.е., если в переменной x было значение 5, то оно будет умножено на 2 и результат помещен в переменную y. В результате в переменной y окажется число 10.

От рассмотренного правого прямоугольника стрелка идёт вниз и влево, объединяется со стрелкой от левого прямоугольника и указывает на параллелограмм, означающий вывод результата. Внутри параллелограмма написано, что выводиться в качестве результата будет значение переменной y.

Далее стрелка идёт вниз и указывает на овал со словом «Конец» внутри, что означает окончание работы алгоритма.

Если же в переменную x было введено отрицательное значение, например, -5, то условие в ромбе не выполнено (ложно) и далее необходимо двигаться влево по стрелке с надписью «Нет».

Эта стрелка идёт влево и вниз, указывая на прямоугольник, означающий присваивание. Внутри прямоугольника написано y =-2x, т.е. значение x надо умножить на -2 и результат положить в переменную y. В этом случае в y также попадёт число 10.

От левого прямоугольника стрелка идёт вниз и вправо, объединяясь со стрелкой от правого прямоугольника. Далее объединённая стрелка указывает на параллелограмм вывода результата, который уже был рассмотрен ранее.

После вывода результата алгоритм заканчивается овалом со словом «Конец» внутри.

Таким образом, алгоритм распадается в ромбе на две ветви, которые опять объединяются перед параллелограммом. Вывод, указанный в параллелограмме, осуществляется и том и в другом случае.

Легко видеть, что независимо от знака вводимого числа, выводится будет неотрицательное число, равное удвоенному модулю введённого.

Описанный пример блок-схемы содержит элементы и линейной структуры, и структуры ветвления. Теперь рассмотрим способы организации циклических структур с помощью блока проверки условия (ромб).

Пример 2. Цикл с предусловием (цикл типа «пока»). Блок-схема полностью здесь составляться не будет, а будет рассмотрен лишь её циклический фрагмент, изображённый на рисунке 3.

Фрагмент блок-схемы начинается со стрелки, входящей в него сверху. Имеется ввиду, что до этого фрагмента были какие-то ещё алгоритмические структуры не изображённые на рисунке.

Итак, входящая сверху стрелка указывает на ромб с условием. Конкретный вид условия будет определяться от решаемой задачи. Если условие истинно, то выполнение алгоритма идёт по ветви, соответствующей стрелке с подписью «Да», выходящей из нижней вершины ромба. Эта стрелка указывает на прямоугольник со словами «Тело цикла». Тело цикла – это набор операторов, которые выполняются на каждом шаге работы цикла. Тело цикла может содержать любые другие алгоритмические конструкции: линейные, ветвление и другие циклы.

От тела цикла стрелка идёт вниз, затем поворачивает влево, вверх и вправо, объединяясь со стрелкой, входящей в ромб сверху. Таким образом, замыкается циклическая конструкция. Алгоритм опять приходит к началу цикла, где проверяется условие.

Если же условие ложно, то алгоритм идёт по ветви, соответствующей стрелке со словом «Нет», выходящей из правой вершины ромба. Переход по этой стрелке означает завершение цикла. Далее выполняются команды алгоритма, следующие за циклом. На рисунке блоки следующих команд не изображены.

Например, пусть необходимо увеличить значение переменной n от 1 до 5 с шагом 1. Тогда в ромбе должно быть написано условие n < 5, а в прямоугольнике присваивание n =n +1. Начальное значение n =1 было присвоено до цикла. В этом случае, попадая в ромб, мы сравниваем n с числом 5. Если n оказывается меньше, т.е. условие истинно, то следует переходить по стрелке «Да» и в прямоугольнике увеличивать значение n на 1 так, что в n окажется число 2. После этого мы опять попадаем в ромб и сравниваем переменную n, в которой находится число 2, с числом 5. Условие опять истинно и цикл вновь повторяется. Так будет повторяться пока n станет равной числу 5. Когда в переменой n окажется число 5, условие в ромбе станет ложным и следующий переход надо будет совершить по стрелке со словом «Нет». Так завершится цикл и алгоритм будет выполняться дальше.

Таким образом, работу этого цикла можно описать словами: Проверить условие и выполнять тело цикла «пока» условие истинно. Поэтому такой цикл и называют циклом типа «пока». Также его называют циклом с предусловием, поскольку условие проверяется перед выполнением тела цикла.

Пример 3. Цикл с постусловием (цикл типа «до»). Блок-схема полностью здесь составляться не будет, а будет рассмотрен лишь её циклический фрагмент, изображённый на рисунке 4.

Стрелка, приходящая сверху, указывает на прямоугольник с телом цикла. От прямоугольника стрелка идёт вниз и указывает на ромб с условием. Из нижней вершины ромба выходит стрелка со словом «Нет», описывает петлю и объединяется со стрелкой, указывающей на прямоугольник. Из правой вершины ромба выходит стрелка со словом «Да» и цикл завершается. Здесь завершение цикла происходит при истинности условия.

Таким образом, данная циклическая конструкция имеет два отличия от предыдущей:

Во-первых, здесь сперва выполняется тело цикла, а потом проверяется условие.

Во-вторых, цикл работает пока условие ложно, а когда оно станет истинным, цикл завершается.

Работу этого цикла можно описать словами: Выполнять тело цикла, затем проверить условие и выполнять тело цикла «до» тех пор, когда условие станет истинным. Поэтому такой цикл и называют циклом типа «до». Также его называют циклом с постусловием, поскольку условие проверяется после первого прохождения тела цикла.

Пример 4. Цикл с параметром (цикл типа «Для»). Блок-схема полностью здесь также составляться не будет, а будет рассмотрен лишь её циклический фрагмент, изображённый на рисунке 5.

Как уже говорилось, блок-схемы с циклом с параметром в стандартных школьных задачах не встречается. Однако, эта циклическая конструкция используется в программировании и мы начнем знакомство с ней с блок-схем.

Итак, стрелка, входящая сверху, указывает на шестиугольник с описанием управления циклом внутри. Напомним, что этот блок называется Модификация.

Из шестиугольника выходит стрелка, указывающая на прямоугольник с телом цикла.

Из прямоугольника стрелка выходит вниз, образует петлю и объединяется со стрелкой, входящей в шестиугольник сверху.

Для описания условий выполнения цикла с параметром необходимо задать следующие значения:

1. Начальное значение параметра цикла (обозначено a);

2. Конечное значение параметра цикла (обозначено b);

3. Шаг цикла (обозначено h).

Работу цикла с параметром можно описать следующим образом.

Параметру цикла i присваивается заданное заранее начальное значение. Это происходит только при первом прохождении цикла. Затем выполняется тело цикла и в соответствии со стрелкой опять выполняются действия в шестиугольнике. Однако, присваивание начального значения больше не происходит. В шестиугольнике сравнивается значение параметра I с конечным значением. Если они равны или значение параметра цикла I больше, то выполнение цикла прекращается и выполняются следующие за ним шаги алгоритма. Если же значение параметра цикла I меньше конечного значения, то оно увеличивается на величину шага, после чего выполняется тело цикла, а потом опять осуществляется переход по стрелке и проверяется значение параметра. Цикл с параметром обязательно завершает свою работу после выполнения заданного числа шагов.

### **Контрольные вопросы**

1. Как на блок-схеме обозначается начало и конец алгоритма?
2. Что на блок-схеме записывается внутри прямоугольника?
3. Для чего на блок-схеме служит параллелограмм?
4. Что на блок-схеме записывается внутри ромба? Приведите примеры.
5. Что на блок-схеме означают стрелки с надписями «Да» и «Нет», выходящие из ромба?
6. Как на блок-схеме описать цикл с предусловием?
7. Может ли цикл с предусловием ни разу не выполнится? Почему?
8. Как на блок-схеме описать цикл с постусловием?
9. Может ли цикл с постусловием ни разу не выполнится? Почему?
10. Что на блок-схеме обозначает шестиугольник?
11. Что записывается внутри шестиугольника?
12. Какую структуру имеет алгоритм, блок-схема которого состоит только из овалов, параллелограммов и прямоугольников?

### **Упражнения для самостоятельного выполнения**

1. Определите по блок-схеме, изображенной на рис. 2, какое число будет выведено при вводе:  
А) числа -12;  
Б) Числа 37;  
В) Числа 0?

2. Запишите алгоритм, изображенный в виде блок-схемы на рис. 2, на школьном алгоритмическом языке.

3. Разработайте блок-схему алгоритма отыскания площади прямоугольника по его длине и ширине. При невозможности изобразить блок-схему на рельефном чертеже, допускается разработать её словесное описание.

4. Разработайте блок-схему алгоритма отыскания площади треугольника по двум сторонам и углу между ними. При невозможности изобразить блок-схему на рельефном чертеже, допускается разработать её словесное описание.

5. Разработайте блок-схему алгоритма отыскания наибольшего общего делителя двух натуральных чисел. При невозможности изобразить блок-схему на рельефном чертеже, допускается разработать её словесное описание.

6. Разработайте блок-схему алгоритма отыскания модуля вводимого действительного числа. При невозможности изобразить блок-схему на рельефном чертеже, допускается разработать её словесное описание.

7. Разработайте блок-схему алгоритма вычисления факториала натурального числа n. Напомним, что факториалом называется произведение всех натуральных чисел от 1 до n. При невозможности изобразить блок-схему на рельефном чертеже, допускается разработать её словесное описание.

### **Параграф 2 Решение задач по блок-схеме**

В предыдущем параграфе была рассмотрена одна из самых простых блок-схем, содержащих конструкцию ветвления, а также примеры организации циклических структур разного типа. В этом параграфе мы рассмотрим более сложные примеры блок-схем и подробно разберем работу с ними.

Пример 1. Квадратное уравнение. Пусть необходимо составить блок-схему решения квадратного уравнения:

ax2 +bx +c =0.

Алгоритм решения квадратного уравнения хорошо известен из школьной программы по алгебре. Этот алгоритм следует вспомнить для составлении разрабатываемой блок-схемы.

В соответствии с описанием элементарных блоков всякая блок-схема должна начинаться с овала со словом «Начало» внутри. Это и будет первым блоком разрабатываемой схемы.

Далее, необходимо ввести исходные данные, которыми являются коэффициенты уравнения a, b и c. Т.е. от овала вниз идёт стрелка, указывающая на блок ввода данных – параллелограмм с именами переменных a,b и c внутри.

Теперь, в соответствии с алгоритмом решения квадратного уравнения, необходимо вычислить дискриминант D. Для этого используем блок прямоугольник в который идет стрелка от параллелограмма. Внутри прямоугольника записывается формула вычисления дискриминанта:

D =b2 -4ac.

Для определения того, по какой ветви пойдёт алгоритм, следует сравнить дискриминант D с нулём. Для этого используется блок ромб с записанным внутри условием D >=0. Из левой вершины ромба выходит стрелка, подписанная словом «Нет», соответствующая ложности условия. Из правой вершины выходит стрелка со словом «Да», соответствующая истинности условия в ромбе.

Первой рассмотрим левую ветвь «Нет». Выходя из левой вершины ромба она поворачивает вниз и указывает на параллелограмм, в котором написано ДКН, что означает «Действительных корней нет». Это сокращение используется для экономии места на рисунке. Действительно, если условие в ромбе ложно, это означает, что дискриминант отрицательный (D < 0) и корней уравнение не имеет.

Далее из параллелограмма стрелка выходит вниз, поворачивает на право и объединяется с ветвью алгоритма, соответствующей слову «Да». Общая стрелка, получившаяся после объединения, указывает в овал со словом «Конец» внутри. Это последний блок блок-схемы.

Ветвь алгоритма, соответствующая выполнению условия D >=0, на блок-схеме обозначается стрелкой со словом «Да». Эта стрелка выходит из правой вершины ромба и, поворачивая вниз, указывает на прямоугольник, в котором записана формула вычисления первого корня:

Ниже этого прямоугольника сразу идёт второй с формулой вычисления второго корня уравнения внутри:

Обратите внимание, что обе формулы могут быть записаны в одном прямоугольнике. Далее, выходя из нижнего прямоугольника стрелка указывает на параллелограмм с переменными x1 и x2 внутри. Напомним, что параллелограмм обозначает вывод данных.

Выходящая из параллелограмма стрелка объединяется со стрелкой левой ветви и объединённая стрелка указывает на последний блок – овал со словом «Конец» внутри. Блок-схема построена. Она изображена на рисунке 6.

Пример 2. Количество корней квадратного уравнения. Пусть теперь необходимо составить блок-схему алгоритма, определяющего количество корней квадратного уравнения:

ax2 +bx +c =0.

Начало разрабатываемой блок-схемы будет таким же, как в примере 1. Т.е. блок-схема начинается с овала со словом «Начало» внутри. Затем стрелка идет вниз и указывает на параллелограмм ввода данных с переменными a, b и c внутри. Затем стрелка указывает на прямоугольник, содержащий формулу вычисления дискриминанта:

D =b2 -4ac.

Далее, двигаясь вниз по стрелке мы попадаем в ромб с проверкой условия D >=0. Из левой вершины ромба выходит стрелка со словом «Нет», соответствующая ветви алгоритма, выполняемой в случае отрицательного дискриминанта. Из правой вершины ромба выходит стрелка со словом «Да», соответствующая ветви алгоритма для случая неотрицательного дискриминанта. Рассмотрим эту ветвь подробнее.

Итак, стрелка со словом «Да» выходит из правой вершины ромба и поворачивает вниз, указывая на ещё один ромб с проверкой условия D =0. Напомним, что в случае равенства дискриминанта нулю, уравнение имеет два одинаковых (совпавших) корня. А в случае строго положительного дискриминанта (D > 0) уравнение имеет два различных корня.

В этом втором ромбе алгоритм опять разделяется и возникает ещё две ветви. Таким образом, алгоритм распадается на три ветви:

1. Ветвь, соответствующая стрелке со словом «Нет» от первого ромба (с условием D >=0);

2. Ветвь, соответствующая стрелке со словом «Нет» от второго ромба (с условием D =0);

3. Ветвь, соответствующая стрелке со словом «Да» от второго ромба (с условием D =0).

В конце блок-схемы все три ветви объединятся в одну и соответствующая ей стрелка укажет на овал со словом «Конец».

Очевидно, что каждая из этих трёх стрелок укажет перед объединением на соответствующий параллелограмм с выводом результата. Первая стрелка, соответствующая отрицательному дискриминанту (D < 0), укажет на параллелограмм со словом ДКН внутри. Напомним, что ДКН означает «Действительных корней нет». Вторая стрелка, соответствующая положительному дискриминанту (D > 0), укажет на параллелограмм с фразой для вывода «2разных корня». А третья, соответствующая нулевому дискриминанту (D =0), укажет на параллелограмм с фразой для вывода «2 одинаковых корня».

Обратите внимание, что для более логичного и компактного расположение блоков на схеме последняя из стрелок должна выходить из нижней вершины соответствующего ромба.

Далее, как уже говорилось, все три стрелки объединяются в одну, которая указывает на завершающий овал со словом «Конец» внутри.

Разработанная таким образом блок-схема изображена на рисунке 7.

Обратите внимание, что знак «=» (равно) используется в двух различных случаях. Он служит для присваивания какой-либо переменной значения или для сравнения выражения слева от знака равенства с выражением справа от него. Спутать смысл знака равно нельзя, поскольку в качестве присваивания он используется только внутри прямоугольников, а для сравнения – внутри ромба. В языках программирования знаки для присваивания и для сравнения имеют различное написание. Часто в блок-схемах для обозначения присваивания используют знак «:=» (двоеточие равно). Такой же знак используется в языке программирования Pascal.

В более сложных алгоритмах используют массивы. Массив состоит из однотипных переменных с одним именем, отличающихся между собой индексами. Например, корни квадратного уравнения хранятся в переменных x1 и x2. Можно сказать, что корни лежат в двухэлементном массиве. Т.е. элементы массива – это переменные с индексами.

В записи алгоритмов и в обычной математической записи бывают отличия в обозначениях. Так, в математике для обозначения элементов массива используются сниженные цифры в качестве индексов, а в алгоритмическом языке и в языках программирования для обозначения индекса используют квадратные скобки. Например, корни квадратного уравнения будут обозначаться x[1] и x[2]. Массивы могут содержать не только два, но и сколь угодно много элементов.

Рассмотрим пример блок-схемы с массивом. Обратите внимание, что в этой блок-схеме обозначения соответствуют алгоритмическому языку, а не стандартным математическим. В дальнейшем мы будем использовать обозначения, принятые в программировании и теории алгоритмов.

Пример 3. Массивы. Пусть необходимо определить какой результат выдаст алгоритм, изображённый на рисунке 8 (а, Б), если на вход подать числа:

4, 1, -1, 0, 3.

Как и все блок-схемы, эта начинается с овала со словом начало внутри. Ввод данных здесь разбит на два параллелограмма. Это подчеркивает различное назначение вводимых данных. В первом параллелограмме находится переменная n, в которую вводится первое число из списка данных, т.е. число 4. Во втором параллелограмме задается ввод данных в массив a[1], a[2], a[3], a[4]. Длина массива зависит от значения переменной n. Поскольку в неё было введено число 4, массив используется четырёхэлементный. Если бы в n было введено другое число, то и массив имел бы другую длину.

Итак, в массив вводятся числа:

A[1] :=1

A[2] :=-1

A[3] :=0

A[4] :=3

Далее по стрелке идёт прямоугольник с тремя присваиваниями:

B :=a[1]

C :=b

I :=1

Поскольку в данном примере в a[1] было введено число 1, то в переменной b также окажется 1, а затем 1 попадёт и в переменную c. В переменной I после присваивания будет лежать число 1. С помощью переменной i в данном примере далее организуется цикл, и в нём эта переменная играет роль счётчика цикла, т.е. в ней всегда будет хранится номер шага, выполняемого циклом.

Из рассмотренного прямоугольника стрелка идёт вниз и указывает на ромб. Обратите внимание, что в эту стрелку слева вливается другая стрелка, приходящая снизу. Данный ромб организует цикл, в нём написано условие I <=n. Поскольку в переменной i лежит число 1, а в переменной n – число 4, то цикл совершит 4 шага. Цикл завершит работу при i =5, т.е. когда условие станет ложным. Таким образом, после окончания цикла алгоритм пойдёт по ветви, соответствующей стрелке со словом «Нет», выходящей из правой вершины ромба.

Эта стрелка указывает на прямоугольник с присваиванием s :=c +b. Далее по стрелке идёт параллелограмм вывода, в котором записана переменная s. Затем стрелка указывает на овал со словом «Конец» и алгоритм заканчивается. Таким образом, результат работы алгоритма зависит от переменных b и c, за которыми нам и следует следить в дальнейшем.

Рассмотрим теперь тело цикла. Оно состоит из двух ромбов с проверкой условия и трёх прямоугольников с присваиванием. На первый взгляд эта структура производит впечатление очень сложной. Разберём её последовательно.

Итак, от уже рассмотренного ромба с условием окончания цикла (I <=n) идёт стрелка вниз со словом «Да». Она указывает на первый ромб тела цикла с условием a[i] > b. На первом шаге цикла в переменной i лежит число 1, т.е. в этом ромбе осуществляется сравнение

a[1] > b.

Поскольку в переменной b лежит то же число 1, что и в элементе массива a[1], то условие ложно и дальнейший переход следует осуществить по стрелке, выходящей из нижней вершины ромба, со словом «Нет».

Эта стрелка указывает на следующий ромб с условием

A[i] < b.

По аналогичным причинам это условие также ложно и далее следует переходить по стрелке со словом «Нет», выходящей из нижнего угла ромба.

Эта стрелка указывает на прямоугольник с присваиванием

I :=I +1.

Это присваивание позволяет перейти к следующему шагу цикла. После его выполнения к старому значению переменной i будет прибавлена единица, и значение переменной i станет равно 2. Стрелка выходит из прямоугольника влево и, поднявшись вверх, объединяется со стрелкой, входящей в ромб с условием окончания цикла. Таким образом, начинается выполнение второго шага цикла.

Итак, мы видим, что если в ромбах в теле цикла условия ложны, то никаких изменений в интересующих нас переменных b и c не происходит. Рассмотрим случаи ложности этих условий.

Если в первом ромбе тела цикла условие

A[i] > b

Окажется истинным, то осуществив переход по стрелке со словом «Да», мы окажемся в прямоугольнике с присваиванием

B :=a[1],

и далее по стрелке вернемся в уже рассмотренный второй ромб тела цикла. Однако, присваивание

b :=a[1]

уже было осуществлено в самом первом прямоугольнике блок-схемы.

Истинность условия

A[i] < b,

Записанного во втором ромбе тела цикла, Означает переход по стрелке со словом «Да», выходящей из правой вершины этого ромба. Стрелка указывает на прямоугольник с присваиванием

C :=a[1].

Но это присваивание также уже было осуществлено в первом прямоугольнике блок-схемы.

Таким образом, значения переменных b и c в теле цикла не изменяются при движении по любой стрелке. Значения b и c остаются те же, что были им присвоены в первом прямоугольнике блок-схемы. Вспоминая эти значения, можно сделать вывод, что в переменную s попадёт число 2 и, соответственно введённым в этом примере данным, результат работы алгоритма будет равен 2.

Обратите внимание, что такая сложная на первый взгляд блок-схема, при подробном разборе оказалась весьма простой. Алгоритм, заданный этой блок-схемой, не решает какую-либо задачу, а разработан в учебных целях для развития умения анализировать блок-схемы.

### **Контрольные вопросы**

1. Расскажите по блок-схеме о работе алгоритма решения квадратного уравнения.
2. Может ли на одной блок-схеме присутствовать пять ромбов?
3. Как на блок-схеме отличить знак присваивания от знака сравнения?
4. Что такое массивы?
5. Как на блок-схеме записываются индексы элементов массива?
6. Какой тип цикла реализован на блок-схеме из примера 3 (рис. 8)?

### **Упражнения для самостоятельного выполнения**

1. По блок-схеме, описывая каждый шаг алгоритма, решите квадратное уравнение:  
А) x2 -5x +6 =0;  
Б) x2 +2x +5 =0;  
В) x2 -6x +9 =0.

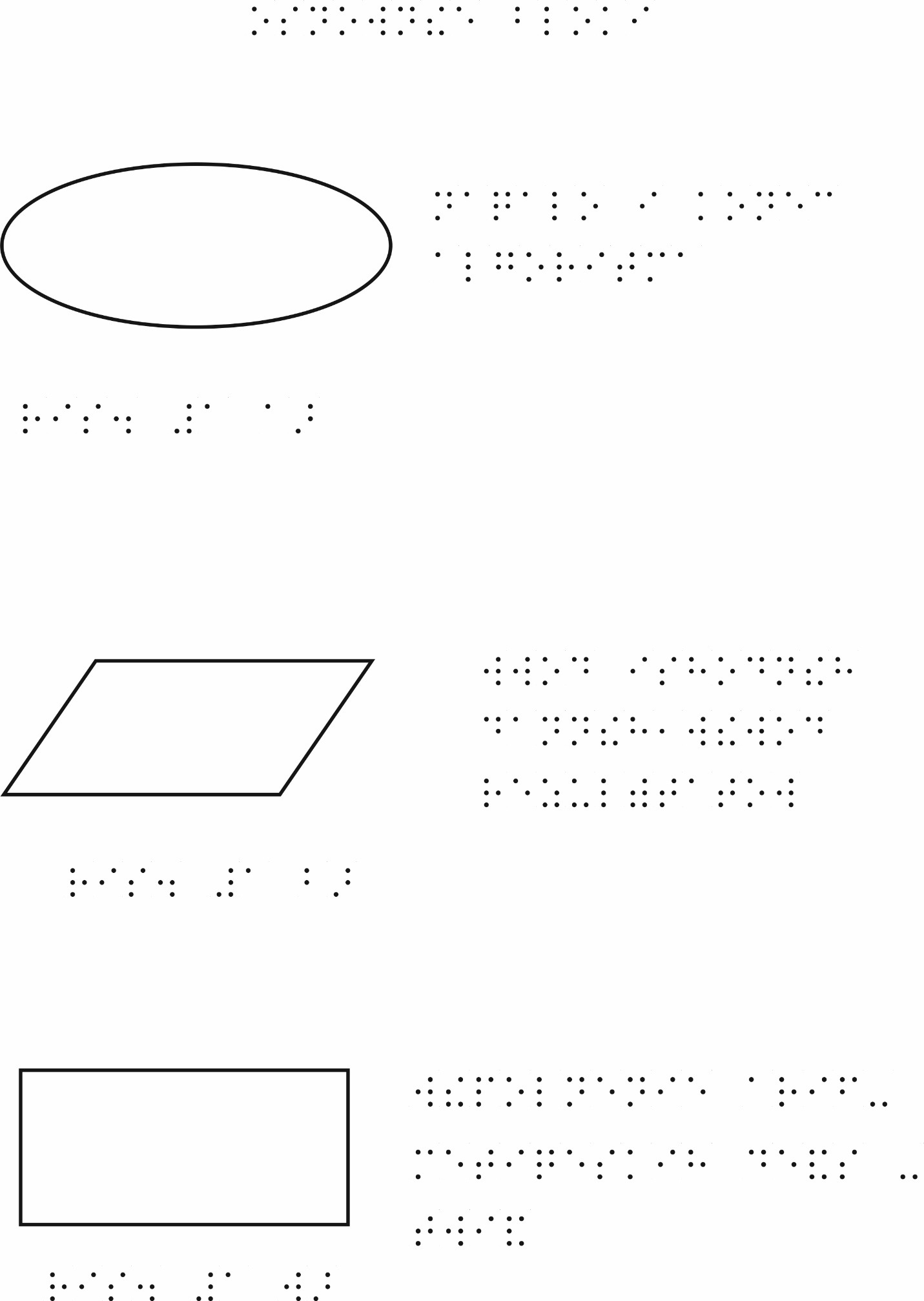
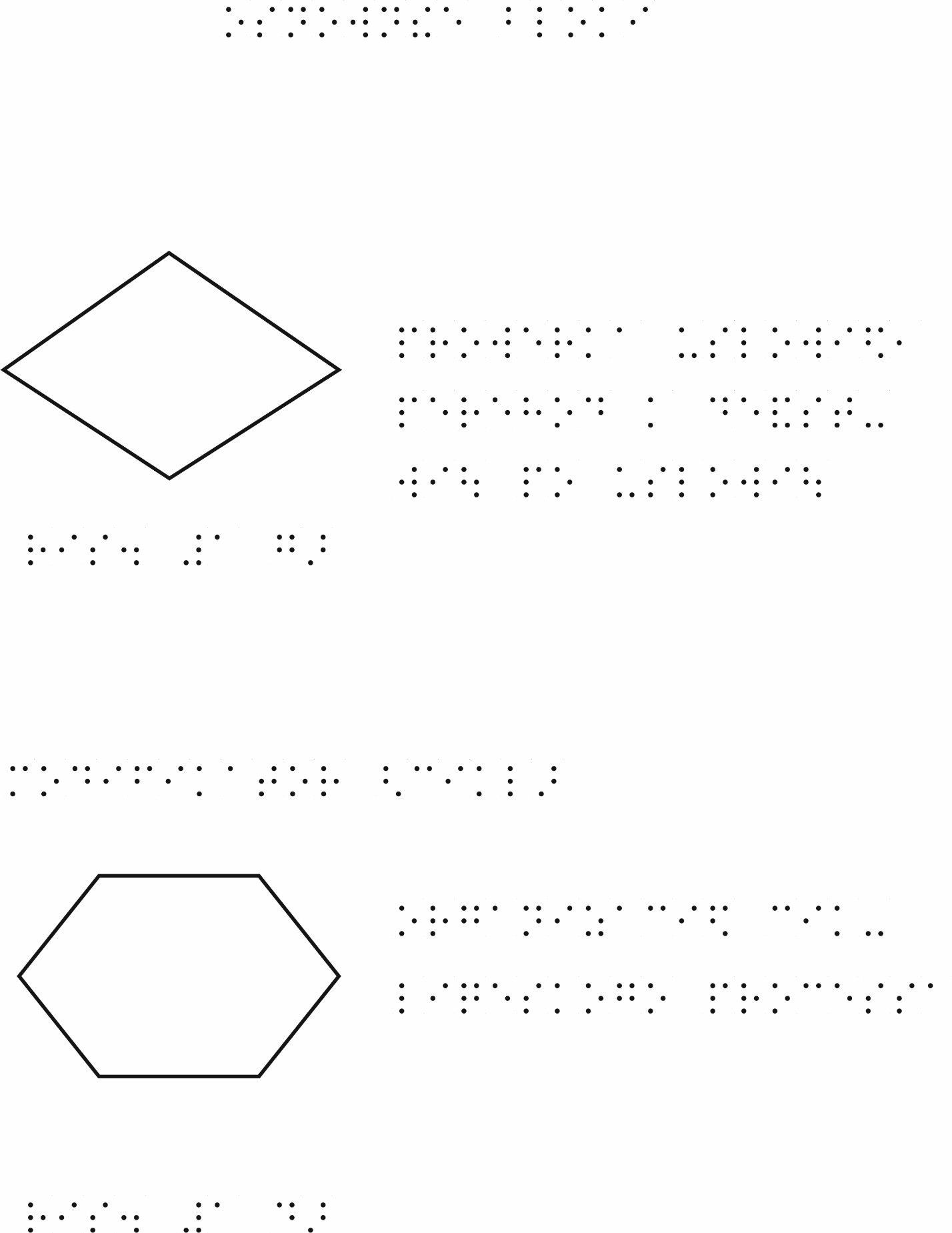
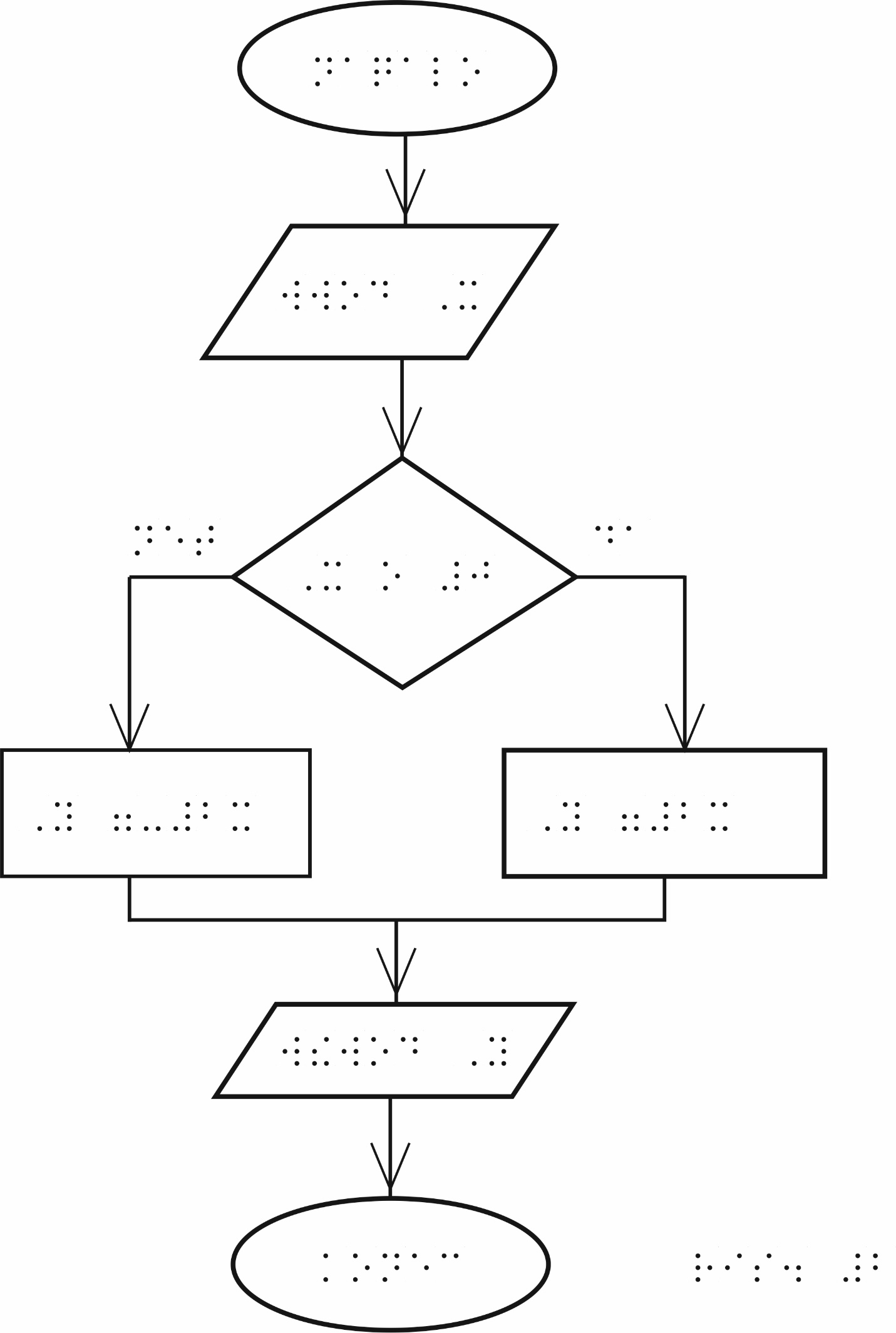
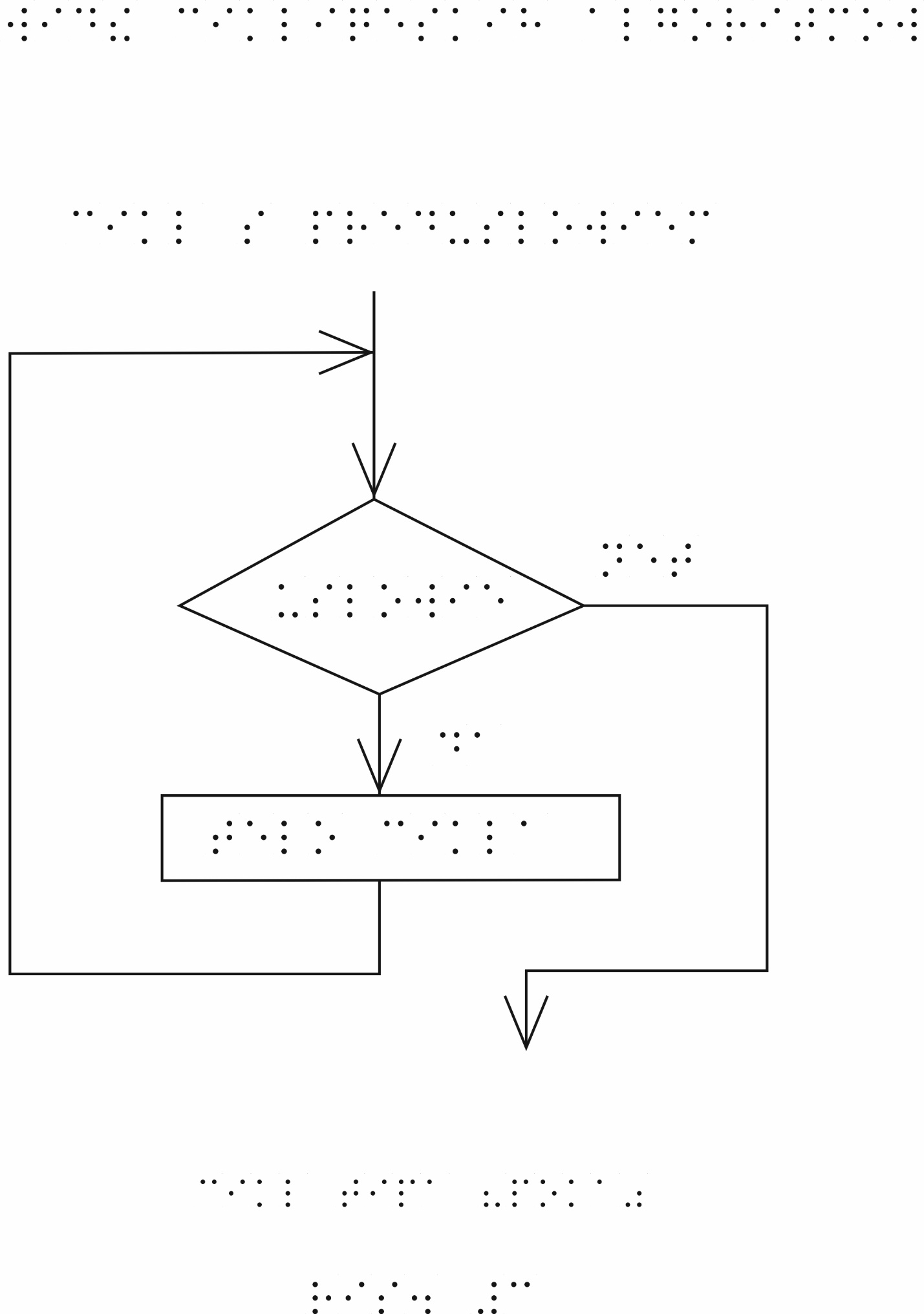
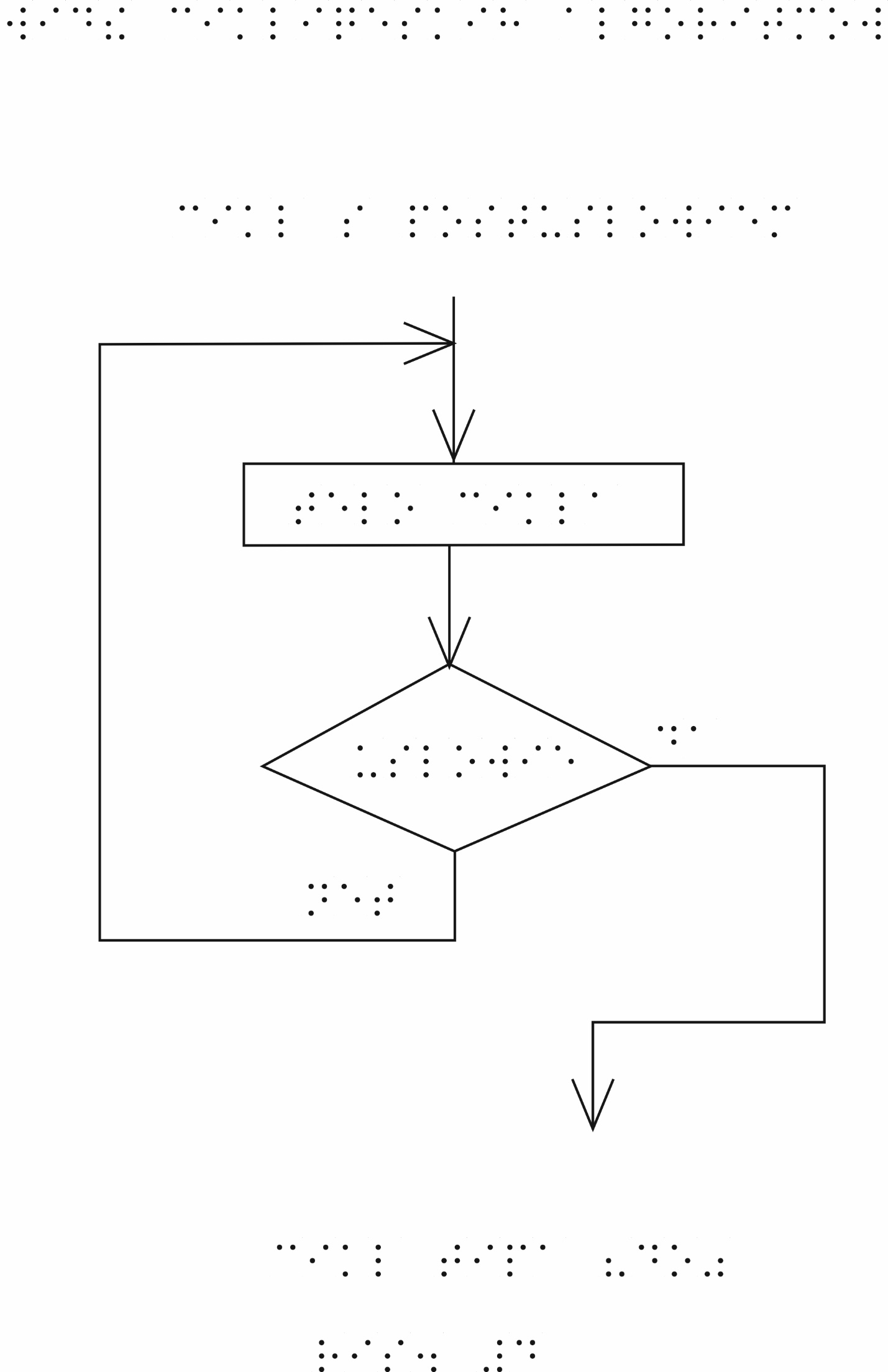
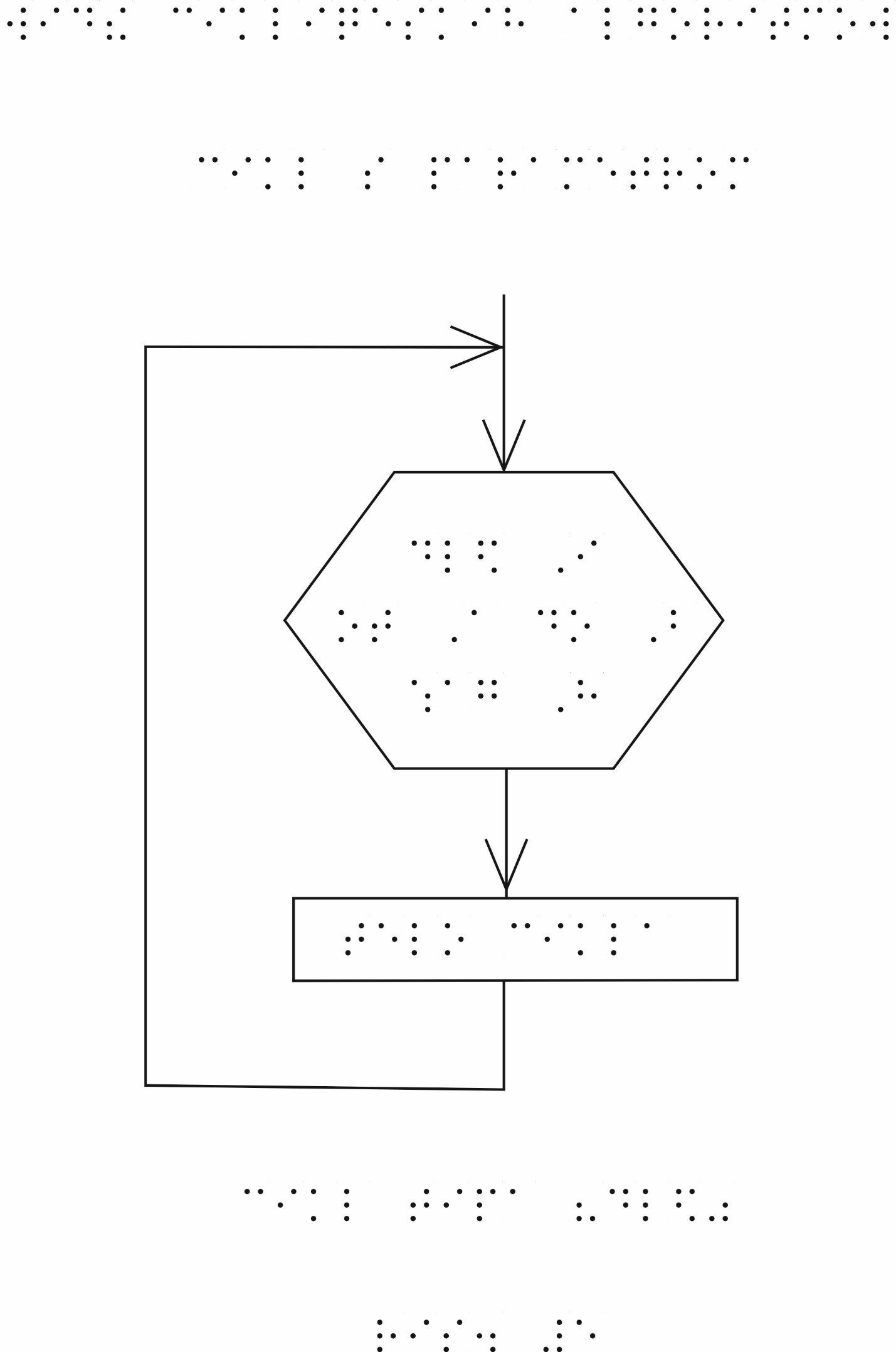
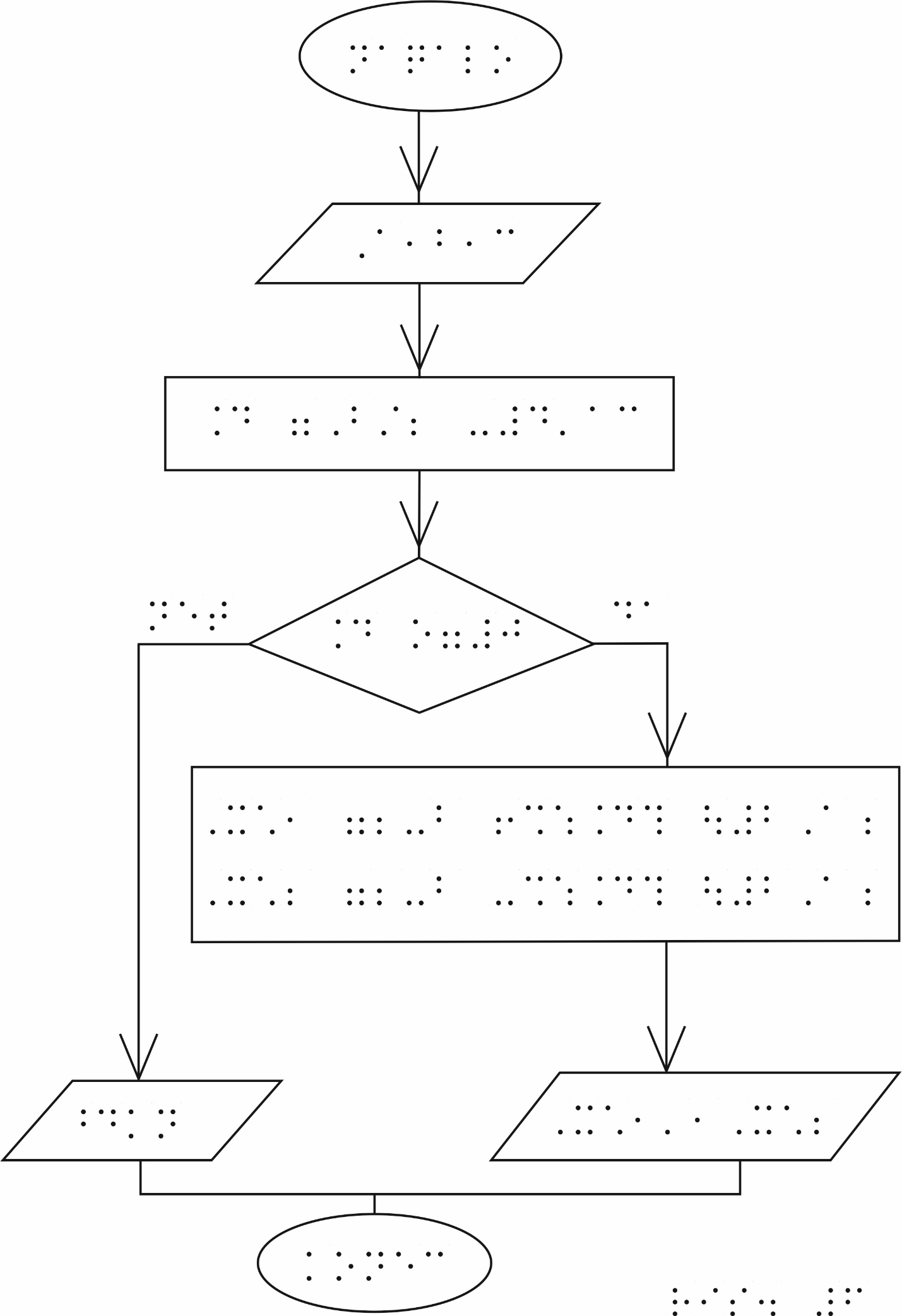
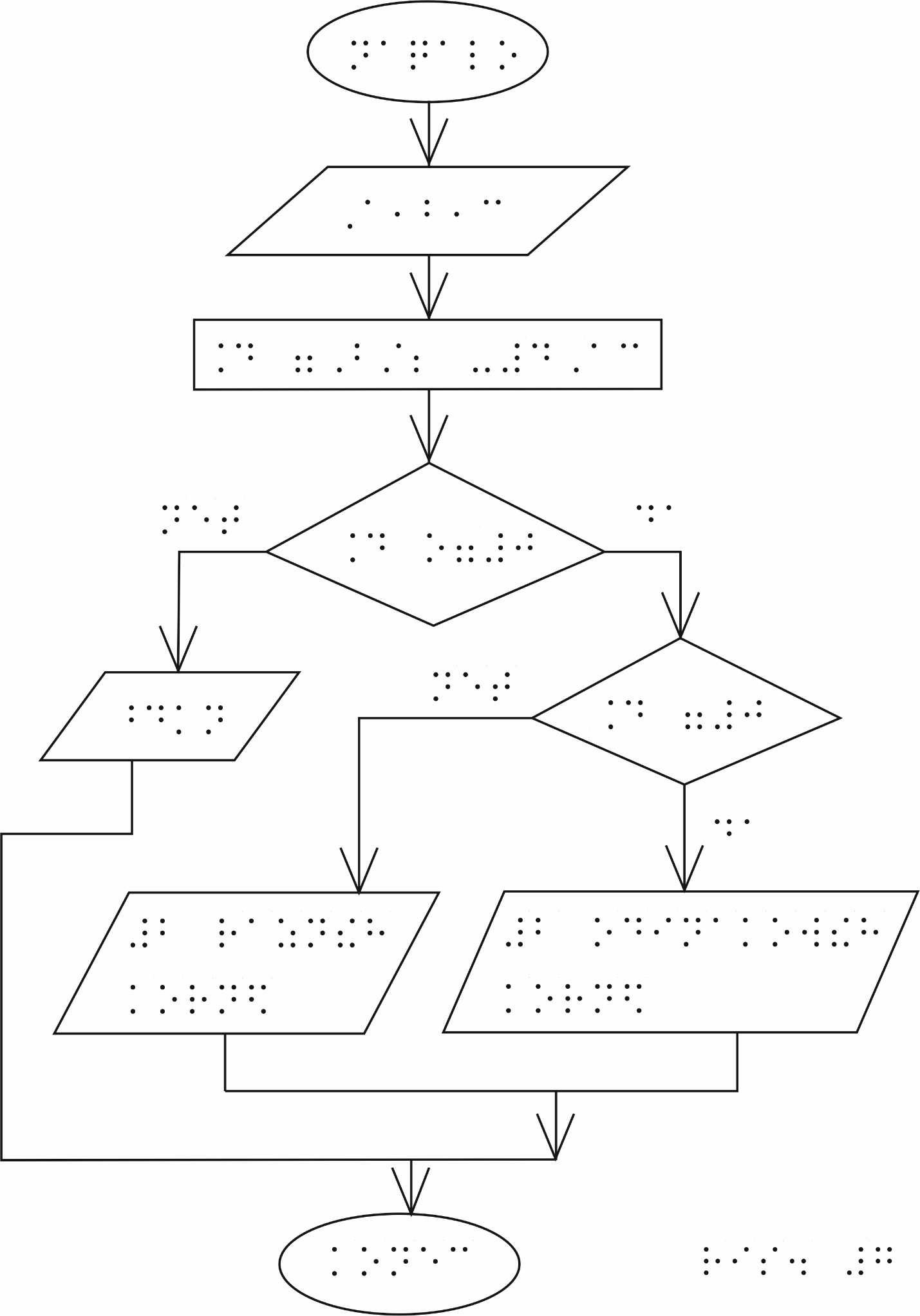
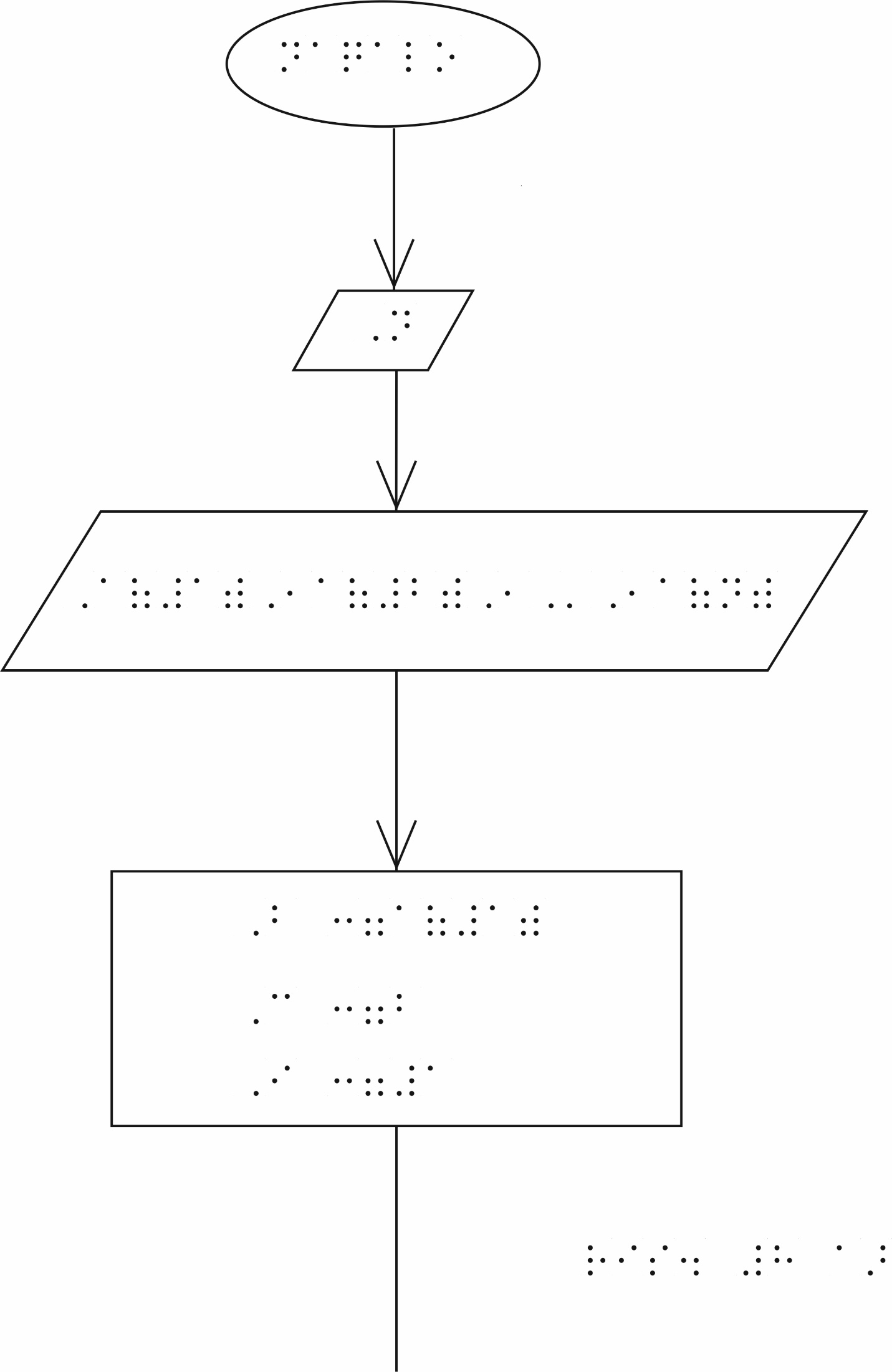
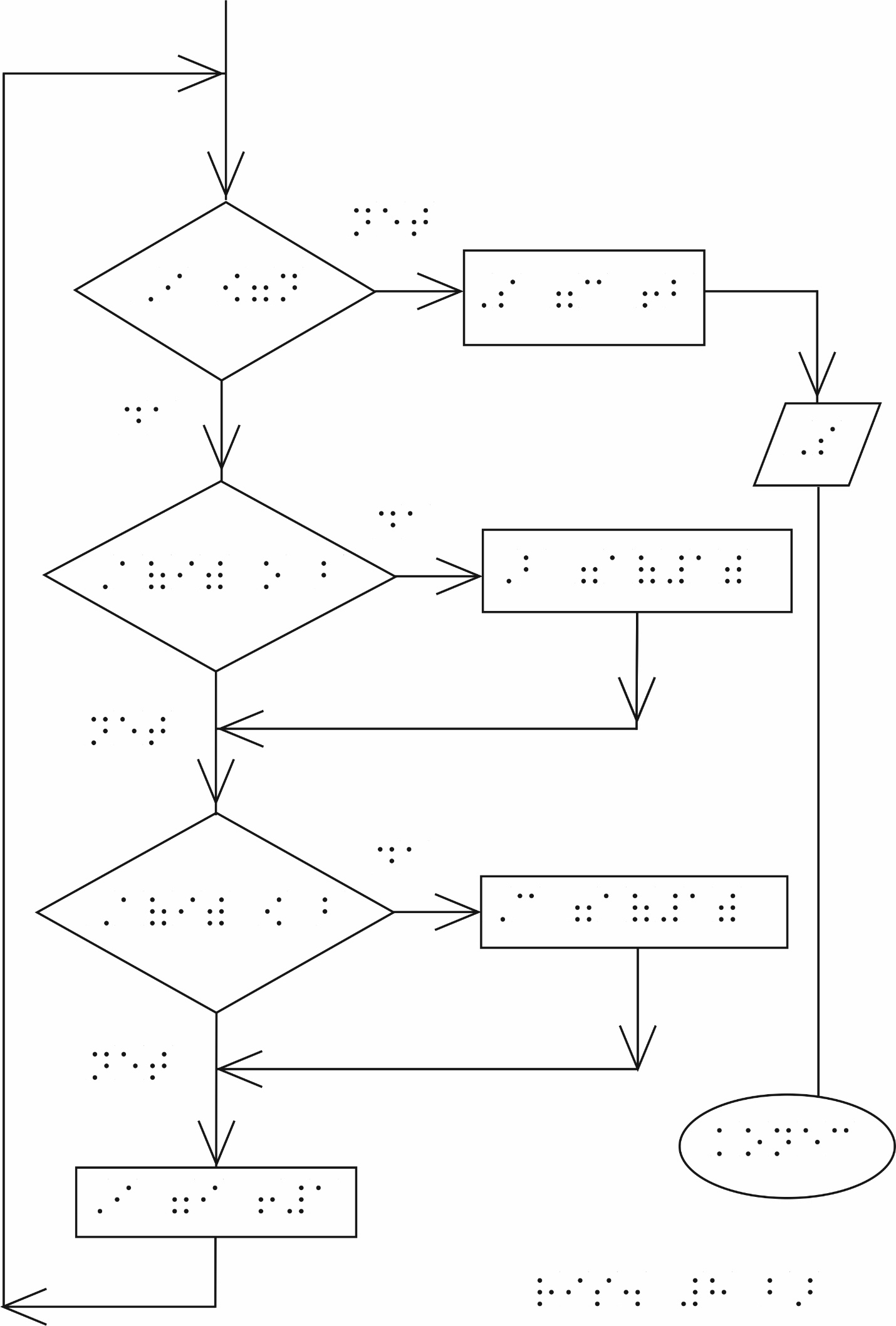
2. По блок-схеме, описывая каждый шаг алгоритма, определите количество корней квадратного уравнения:  
А) 3x2 -4x +4 =0;  
Б) x2 -25x +150 =0;  
В) 16x2 -256x +1024 =0.

3. Разработайте блок-схему алгоритма, определяющего количество корней квадратного уравнения больших числа 5. При невозможности изобразить блок-схему на рельефном чертеже, допускается разработать её словесное описание.

4. Разработайте блок-схему алгоритма, определяющего является ли данное число корнем данного квадратного уравнения. При невозможности изобразить блок-схему на рельефном чертеже, допускается разработать её словесное описание.

5. Разработайте блок-схему алгоритма определения простого числа. Алгоритм получает на входе число, а на выходе выдаёт сообщение простое это число или составное. При невозможности изобразить блок-схему на рельефном чертеже, допускается разработать её словесное описание.

6. По блок-схеме на рисунке 8 определите какое число будет выдано в качестве результата при входных данных:  
 А) 3, 1, 2, 0;  
Б) 5, -2, -1, 2, 0, 10;  
В) 8, 3, -1, 5, 2, 1, 2, 3, -3.



## **Глава 3 Языки программирования**

### **Параграф 1 История развития языков программирования**

Само программирование появилось задолго до первых электронно-вычислительных машин. Идеи, положенные в основу программирования, высказал ещё Чарльз Бэббидж (1791 - 1871). Этого человека по праву считают основателем современной теории программирования и вычислительных машин. В те годы еще не существовало электронных ламп, транзисторов или микросхем, но Бэббидж достаточно точно описал основные принципы, на которых теперь строятся все вычислительные машины.

Бэббидж является автором идеи создания вычислительной машины, которая в наши дни называется компьютером. Впервые Бэббидж задумался о создании механизма, который позволил бы автоматически производить сложные вычисления с большой точностью, в 1812 году. На эти мысли его натолкнуло изучение логарифмических таблиц, при пересчёте которых были выявлены многочисленные ошибки в вычислениях, обусловленные человеческим фактором. Ещё тогда он начал осмысливать возможность проведения сложных математических расчётов при помощи механических аппаратов.

Однако Бэббидж не сразу начал заниматься развитием идеи построения вычислительного механизма. Лишь в 1819 году, когда заинтересовался астрономией, он более точно определил свои идеи и сформулировал принципы вычисления таблиц разностным методом при помощи машины, которая впоследствии была названа разностной. Эта машина должна была производить комплекс вычислений, используя только операцию сложения. В 1819 году Чарльз Бэббидж приступил к созданию малой разностной машины, а в 1822 году он закончил её строительство и выступил перед Королевским Астрономическим обществом с докладом о применении машинного механизма для вычисления астрономических и математических таблиц. Он продемонстрировал работу машины на примере вычисления членов последовательности. Работа разностной машины была основана на методе конечных разностей. Малая машина была полностью механической и состояла из множества шестерёнок и рычагов. В ней использовалась десятичная система счисления. Она оперировала 18-разрядными числами с точностью до восьмого знака после запятой и обеспечивала скорость вычислений 12 членов последовательности в 1 минуту. Малая разностная машина могла считать значения многочленов седьмой степени.

За создание разностной машины Бэббидж был награждён первой золотой медалью Астрономического общества. Однако малая разностная машина была экспериментальной, так как имела небольшую память и не могла быть использована для больших вычислений.

В 1822 году Бэббидж задумался о создании большой разностной машины, которая позволила бы заменить огромное количество людей, занимающихся вычислением различных астрономических, навигационных и математических таблиц. Это позволило бы сэкономить затраты на оплату труда, а также избавиться от ошибок, связанных с человеческим фактором. Однако, большая разностная машина так и не была достроена.

Труды Бэббиджа по созданию разностной машины не пропали даром. В 1854 году шведский изобретатель Шойц по работам Бэббиджа построил несколько разностных машин. А ещё через некоторое время Мартин Виберг усовершенствовал машину Шойца и использовал её для расчётов и публикации логарифмических таблиц.

Несмотря на неудачу с большой разностной машиной, Бэббидж в 1833 году начал разработку программируемой вычислительной машины, которую он назвал аналитической (прообраз современного компьютера). В отличие от разностной машины, аналитическая машина позволяла решать более широкий ряд задач. Именно эта машина стала делом его жизни и принесла посмертную славу.

Архитектура современного компьютера во многом схожа с архитектурой аналитической машины Бэббиджа. В аналитической машине он предусмотрел следующие части:

склад (store);

фабрика или мельница (mill);

управляющий элемент (control);

устройства ввода-вывода информации.

Склад предназначался для хранения как значений переменных, с которыми производятся операции, так и результатов операций. В современной терминологии это называется памятью.

Мельница (арифметико-логическое устройство, часть современного процессора) должна была производить операции над переменными, а также хранить в регистрах значение переменных, с которыми в данный момент осуществляет операцию.

Третье устройство, которому Бэббидж не дал названия, осуществляло управление последовательностью операций, помещением переменных в склад и извлечением их из склада, а также выводом результатов. Оно считывало последовательность операций и переменные с перфокарт. Перфокарты были двух видов: операционные карты и карты переменных. Из операционных карт можно было составить библиотеку функций. Кроме того, по замыслу Бэббиджа, Аналитическая машина должна была содержать устройство печати и устройство вывода результатов на перфокарты для последующего использования.

Для создания компьютера в современном понимании оставалось лишь придумать схему с хранимой программой, что было сделано 100 лет спустя Эккертом, Мочли и Фон Нейманом.

Бэббидж разрабатывал конструкцию аналитической машины в одиночку. Он часто посещал промышленные выставки, где были представлены различные новинки науки и техники. Именно там состоялось его знакомство с Адой Августой Лавлейс дочерью Джорджа Байрона (1815 - 1852), которая стала его очень близким другом, помощником и единственным единомышленником.

В 1840 году Бэббидж ездил по приглашению итальянских математиков в Турин, где читал лекции о своей машине. Луиджи Менабреа, преподаватель туринской артиллерийской академии, создал и опубликовал конспект лекций на французском языке. Позже Ада Лавлейс перевела эти лекции на английский язык, дополнив их комментариями по объёму превосходящими исходный текст. В комментариях Ада сделала описание Цифровой Вычислительной Машины (ЦВМ) и разработала инструкции по программированию к ней. Это были первые в мире программы. Именно поэтому Аду Лавлейс справедливо называют первым в истории программистом. В своих работах Она показала, что эффективное использование вычислительных машин должно опираться на алгоритмы, записанные в форме программ.

Однако, аналитическая машина так и не была закончена. Вот, что писал Бэббидж в 1851 году: «Все разработки, связанные с Аналитической машиной, выполнены за мой счёт. Я провёл целый ряд экспериментов и дошёл до черты, за которой моих возможностей не хватает. В связи с этим я вынужден отказаться от дальнейшей работы». Несмотря на то, что Бэббидж подробно описал конструкцию аналитической машины и принципы её работы, она так и не была построена при его жизни. Причин этому было много, но основными стали полное отсутствие финансирования проекта по созданию аналитической машины и низкий уровень технологий того времени.

В 1864 году Чарлз Бэббидж написал: «Пройдёт, вероятно, полстолетия, прежде чем люди убедятся, что без тех средств, которые я оставляю после себя, нельзя будет обойтись». В своём предположении он ошибся на 30 лет. Только через 80 лет после этого высказывания была построена машина МАРК-I, которую назвали «осуществлённой мечтой Бэббиджа». Архитектура МАРК-I была очень схожа с архитектурой аналитической машины. Говард Эйкен на самом деле серьёзно изучал публикации Бэббиджа и Ады Лавлейс перед созданием своей машины, причём его машина идеологически незначительно ушла вперёд по сравнению с недостроенной аналитической машиной. Производительность МАРК-I оказалась всего в десять раз выше, чем расчётная скорость работы аналитической машины.

Бэббидж оставил огромный след в истории XIX века и сделал переворот не только в математике и вычислительной технике, но и в науке в целом.

Программирование не могло развиваться без развития вычислительной техники. Сколько-нибудь совершенных механических вычислительных машин больше построено не было, следующий импульс к развитию программирование дало изобретение электронно-вычислительных машин. (Более подробно о развитии электронно-вычислительных машин можно узнать из Учебно-методического пособия по информатике и информационно-коммуникационным технологиям для инвалидов по зрению, разработанного общероссийской общественной организацией инвалидов – Российский Союз Инвалидов в 2018 году).

До пятидесятых годов прошлого века языки программирования представляли собой набор машинных инструкций, трудно понимаемых человеком. Сегодня для написания программы нет необходимости знать команды процессора и другие особенности архитектуры компьютера. Существуют языки программирования, позволяющие писать достаточно сложные программы, способные работать на любом компьютере.

На заре компьютерной эры разработкой языков программирования занимались люди, непосредственно связанные с созданием вычислительных машин, т.е. инженеры. Они представляли язык программирования в виде последовательности номеров операций и ячеек памяти. Приведем пример фрагмента подобной программы:

01 x y (прибавление содержимого ячейки памяти y к ячейке x);

02 x y (вычитание содержимого ячейки памяти y из содержимого ячейки памяти x).

Таким образом, код программы выглядел как набор цифр:

01 10 15

02 11 29

01 10 11

Программистам приходилось долго изучать машинные команды конкретного устройства, а затем писать код программы, представляющий собой последовательность непонятных для человека цифр. Это послужило причиной того, что квалифицированных программистов, способных писать подобные программы стало недостаточно.

Было предложено перевести цифровые обозначения операций в буквенные, т.е. вместо «01 10 15» использовать «ADD 10 15». Это требовало организации автоматического перевода символьных обозначений в машинную команду. Решение оказалось настолько очевидным, что точно неизвестно, кто предложил его первым.

Так был изобретен язык Ассемблера. Вероятнее всего, он появился одновременно сразу в нескольких местах. Предложили название и популяризировали Язык Ассемблера авторы книги «The preparation of programs for a digital computer» Уилкс, Уилер и Гилл. Позднее символами стали обозначать не только простейшие операции, но и адреса памяти, что значительно упростило читаемость программного кода.

Название Ассемблер происходит от английского слова assemble – собирать, монтировать.

Введение в практику языка Ассемблера было сложным процессом, в результате которого появилось такое важное понятие программирования как переменная (символьная переменная).

Дальнейшее развитие программирования привело к созданию компилятора. Компилятор – это программа, преобразующая буквенные обозначения команд и ячеек памяти в машинные коды. Входной информацией для компилятора (исходный код программы) является описание алгоритма или программы на предметно-ориентированном языке, а на выходе компилятор выдает эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код). На основе компилятора стали разрабатываться близкие к естественному человеческому языки программирования.

Одним из первых языков программирования в современном смысле был созданный Джоном Бэкусом язык Fortran. Автор начал работать над созданием своего языка в 1954 году и завершил работу почти через 5 лет.

Полное название этого языка – «The IBM Formula Translating System», или «FORmula TRANslator». Именно в Fortran впервые были реализованы многие атрибуты языка программирования, среди которых:

арифметические и логические выражения;

цикл DO (ранняя был цикл FOR);

условный оператор IF;

подпрограммы;

массивы.

Ещё одним важным наследием языка Fortran является использование ограничений для целочисленных переменных. Имена таких переменных должны были начинаться с одного из 6 символов: I, J, K, L, M, N. Именно отсюда возникла традиция для целых чисел брать имена переменных i, j и т.д.

Fortran является предметно-ориентированным инженерным и научным языком, в его основе лежат операции с комплексными числами и с числами с плавающей запятой. Он не умеет обрабатывать текст. Для работы с текстом на языке Fortran его приходится преобразовывать в специальные коды.

Fortran оказался непригоден для бизнеса, для обслуживания нужд которого был создан язык Cobol. Синтаксис у этого языка иной, максимально приближенный к естественному английскому языку. Cobol стал олицетворением максимального удаления от прежнего машинно-ориентированного мышления к общечеловеческому. Одной из основных особенностей этого языка была возможность работать с текстом и записями.

Исторически следующим языком программирования стал Algol (ALGOrithmic Language), предназначенный для научных отчётов и публикаций. В нём впервые появились естественные для современных программистов особенности:

отличия между присваиванием (:=) и логическим равенством (=);

использование цикла for с тремя аргументами - начальное значение, конечное значение, шаг;

блочная структура программ, заключённых между ключевыми словами begin и end.

Именно из языка Algol произошли C, C++, C#, Java и многие другие популярные сегодня языки программирования.

Четвёртым известным языком программирования пятидесятых годов прошлого века на основе компилятора был Lisp разработанный Джоном Маккарти специально для обслуживания потребностей создания искусственного интеллекта. Особенностями языка стали динамическая типизация и автоматическое распределение памяти. Именно Лисп стал прародителем таких современных языков, как Python и RUBY, а сам до сих пор активно применяется в сфере искусственного интеллекта.

Итак, в пятидесятые годы были созданы четыре языка программирования, приближенные к естественному человеческому языку, это Fortran, COBOL, Algol и Lisp. В последующие годы эти языки совершенствовались и приобретали новые черты. Так были созданы ALGOL 60 и 68, Fortran IV, 66 и 77, COBOL 61, 65 и 68 и др.

Когда эти языки прочно заняли место в арсенале средств разработки, программирование стало значительно доступнее. Программированию стали обучать в институтах и университетах, студентам больше не требовалось быть инженерами, чтобы понимать, как функционирует вычислительная машина с написанным ими текстом. В это время были созданы ориентированные на обучение языки программирования BASIC и Logo.

BASIC объединил в себе FORTRAN и ALGOL 60. Несмотря на большое количество ограничений, он стал первым языком, позволившим обучать программированию на профессиональном уровне.

В истории значение языка BASIC трудно переоценить. Такие известные компании как Microsoft и Apple во многом своим развитием обязаны именно этому языку и его ответвлениям. VB6 и Visual BASIC по-прежнему востребованы, причём не только в системе образования, но и в сфере создания полноценных приложений.

BASIC был создан в шестидесятых годах прошлого века Джоном Кемени и Томасом Курцем в Дартмутском колледже в США, и должен был выполнить две основные функции:

Язык должен быть простым в освоении и использовании;

Язык должен сделать программирование общедоступным.

Известная компания Microsoft продолжала разрабатывать BASIC вплоть до девяностых годов , а окончательной версией стал Quick BASIC – полностью структурированный язык, который сохранил свою простоту и понятность.

Так как BASIC достаточно долго шёл к мировой известности, многие учёные продолжали создавать образовательные языки. Никлаус Вирт руководствовался теми же идеями, что и Кемени с Курцем. Он хотел создать язык, пригодный для обучения информатике в школах и вузах, при этом язык должен был работать на мини-компьютере и быть похожим на один из профессиональных языков. Взяв за основу Algol, в 1972 году Вирт создал язык PASCAL, названный в честь французского математика XVII века Блеза Паскаля. Развитие языков Pascal и BASIC привело к созданию их диалектов qBasic и Turbo Pascal.

Одним из недостатков существующих на тот момент популярных языков программирования была их ограниченность. Например, если необходимо было написать операционную систему, сначала приходилось иметь дело с функциями аппаратного обеспечения (машинные коды), и лишь заложив фундамент, можно было перейти к программированию на языке высокого уровня.

Эту проблему решили в компании Bell Labs – именно здесь увидели свет операционная система Unix и язык программирования C. создал Новый язык C Деннис Ритчи. Язык сочетал в себе функции низкого уровня со структурой Algol. Таким образом, вначале C получил известность как структурированный язык ассемблера.

Самым ярким доказательством качества языка являлось то, что основная часть операционной системы Unix, кроме небольшого ядра, была полностью переписана на C. Слава языка C распространялась, и в конечном итоге C начал использоваться в других операционных системах, и как итог – получил славу языка общего назначения.

Очередной революцией в истории развития языков программирования стало появление в восьмидесятые годы объектно-ориентированных языков. Однако первым таким языком принято считать язык SIMULA, разработанный в 1967 году в Норвежском Вычислительном Центре тремя учёными – Оле-Йоханом Далем, Бьорном Мирхаугом и Кристен Нигаард. Язык был развитием Algol 60 и на первый взгляд не содержал ничего нового. Реальная же важность SIMULA 67 заключалась в философии разделения понятия объекта и его экземпляра. С этой идеи стало развиваться направление объектно-ориентированного программирования. Сегодня для программистов являются уже привычными понятия класса, объекта, экземпляра и наследования, но зародились эти понятия ещё в языке SIMULA. Потребовалось время для совершенствования этих идей и массового внедрения их в практику программирования.

Следующим важным шагом в развитии объектно-ориентированных языков стал Smalltalk, заимствовавший идеи SIMULA. Его создал Алан Кэй в восьмидесятые годы прошлого века. Smalltalk стал первым полноценным объектно-ориентированным языком и задал направление развития таких языков на долгие годы. На этом языке была написана первая графическая рабочая среда. Такие привычные сейчас объекты как окна, иконки, мышь были впервые реализованы на этом языке.

Объектно-ориентированный подход стал настолько популярен в конце восьмидесятых годов, что многие традиционные языки перерабатывались и становились объектно-ориентированными. Бьёрн Страуструп расширил возможности языка C и создал C++. Главная особенность C++ состоит в том, что для этого языка даже не нужен специальный компилятор. Работа с C++ была реализована в виде препроцессора, который преобразовывал код на языке C++ в код C. На каждом компьютере, поддерживающим язык C, мог работать и новый объектно-ориентированный C++.

Pascal, Fortran и Lisp тоже приобрели объектно-ориентированные расширения, но в сравнении с C++ это были незначительные доработки. Появились и новые языки, такие как Ada и Modula 2, которые включали некоторые элементы объектно-ориентированного метода.

В те годы возникали языки и другого типа – логические. Самым удачным из логических языков был Prolog. Он основан на идее создания программ как набора логических операторов и использования перебора для автоматического определения результатов. В программу на этом языке включаются только факты и их анализ. Основой его работы являются результаты математической логики.

В конце XX века стало приобретать популярность визуальное программирование. Визуальное программирование- это способ создания программы путём манипулирования некоторыми, заранее созданными, объектами вместо написания её текста. Такое программирование часто представляют как следующий этап развития текстовых языков. Системы визуального программирования называют также системами быстрой разработки приложений (RAD - Rapid Application Development). Примерами таких систем могут служить Borland Delphi или Microsoft Visual Studio, где редактируются графические объекты и одновременно отображается автоматически сгенерированный соответствующий текст программы.

В последнее время визуальному программированию стали уделять больше внимания в связи с развитием мобильных сенсорных устройств (КПК, планшеты , смартфоны и др.). Визуальное программирование в основном используется для создания программ с графическим интерфейсом. Однако, среда визуального программирования позволяет написать Веб-приложение для браузеров или создать консольное приложение (программа без графического интерфейса с выводом сообщений в консоль). В этих системах также можно программировать микроконтроллеры, микросхемы и т.д.

В отличие от текстовых языков, где принцип программирование и сам язык носят существенно линейный, последовательный характер, в визуальных средах программист работает с разветвленными структурами типа блок-схем. При этом элементами этих блок-схем являются заранее разработанные подпрограммы и функции с унифицированным автоматическим интерфейсом. Такие подпрограммы реализуют, например, кнопку, поле редактирования или любой другой элемент интерфейса. При соединении таких элементов их взаимодействие настраивается без участия программиста. Таким образом, визуальное программирование позволяет программировать на уровне алгоритмов, а не программного кода. Программный код на языках визуального программирования генерируется на основе составленной программистом "блок-схемы" в автоматическом режиме. Программист лишь в нестандартных случаях корректирует программный код, либо создает дополнительные объекты ( модули) для последующего использования в визуальном программировании.

Книга , которую вы читаете, посвящена программированнию на языке Python. Он был задуман в в восьмидесятые годы прошлого века, его создание началось в декабре 1989 года Гвидо ван Россумом в составе центра математики и информатики в Нидерландах. Ван Россум является основным автором Python и по сей день продолжает выполнять центральную роль в принятии решений относительно развития языка.

Версия Python 2.0 была выпущена 16 октября 2000 года и включала в себя много новых функций. Первая версии Python 3.0 была выпущена 3 декабря 2008 года после длительного периода тестирования. Многие функции в этой новой версии были совместимы с Python 2.6 и Python 2.7. В настоящее время это популярный и достаточно простой в изучении язык. Он преподаётся в школах и высших учебных заведениях многих стран. Именно ему будут посвящены следующие главы этой книги.

### **Контрольные вопросы**

1. Кто и когда предложил конструкцию первой вычислительной машины?
2. Кто был первым в истории программистом?
3. Что такое язык программирования?
4. Как выглядили первые языки программирования?
5. Что такое компилятор?
6. В чем особенность объектно-ориентированных языков программирования?
7. Какие языки программирования использовались в образовании?
8. Что такое визуальное программирование?

### **Упражнения для самостоятельной работы**

1. Подготовьте сообщение о Бэббидже. Автором каких изобретений он является?
2. Подготовьте сообщение об аналитической машине Бэббиджа.
3. Подготовьте сообщение опервой электронно-вычислительной машине.
4. Подготовьте сообщение об истории создания и развития языка Python.

### **Параграф 2 Классификация языков программирования**

Классифицировать языки программирования можно по различным признакам. Рассмотрим некоторые из таких классификаций.

Языки программирования можно разделять на поколения по историческому признаку. При этом, их принято делить на пять поколений.

Первое поколение соответствует Началу пятидесятых годов прошлого века , когда только появились электронно-вычислительные машины. Как уже говорилось в предыдущем параграфе, в те годы программирование осуществлялось непосредственно на машинных инструкциях.

Второе поколение появилось в конце пятидесятых - начале шестидесятых годов . Это поколение связано с символическим ассемблером, в котором появилось понятие переменной, а инструкции приобрели буквенные (символические) обозначения.

Языки третьего поколения родились в шестидесятые годы. Это универсальные языки высокого уровня, с их помощью удается решать задачи из любых областей. Такие качества новых языков, как относительная простота, независимость от конкретного компьютера и возможность использования мощных синтаксических конструкций, позволили резко повысить производительность труда программистов. Подавляющее большинство языков этого поколения успешно применяется и сегодня.

Четвертое поколение языков программирования зародилось в семидесятые годы и эти языки продолжают развиваться по настоящее время. Языки этого поколения предназначены для реализации крупных проектов, повышают их надежность и скорость создания, ориентированы на специализированные области применения. В эти языки встраиваются мощные операторы, позволяющие одной строкой описать такую функциональность, для реализации которой на языках младших поколений потребовались бы тысячи строк исходного кода.

Пятое поколение ассоциируется со временем с конца девяностых годов по настоящее время. К этому поколению относятся системы автоматического создания прикладных программ с помощью визуальных средств разработки, без знания программирования. Главная идея, которая закладывается в основу этих языков, состоит возможности автоматического формирования программного текста на универсальных языках программирования. Инструкции же вводятся в компьютер в максимально наглядном виде с помощью методов, наиболее удобных для человека, не знакомого с программированием.

Визуальное программирование перешло к иной, событийной парадигме, то есть программа в них выполняется не последовательно от начального ввода данных до выдачи результата, как в языках предыдущих поколений, а отдельными короткими звеньями кода (алгоритма), начинаясь с некоторого инициирующего события и заканчиваясь либо новым внешним событием, меняющим ход программы, либо генерируя свое событие для другого звена. Таким образом программа, как правило, не имеет характера однозначной, заранее предопределенной цепи последовательных блоков, как в текстовых языках, но некоторой разветвленной сети программных звеньев, последовательность выполнения которых диктуется внешними событиями. Такая парадигма является адекватным ответом на новый характер работы современного программного обеспечения, которое функционирует в режиме интерактивного взаимодействия с другими программами, устройствами и человеком.

Также языки программирования можно классифицировать на языки низкого и высокого уровня. Поясним, что это значит. Разные типы процессоров имеют разные наборы команд. Если язык программирования ориентирован на конкретный тип процессора и учитывает его особенности, т.е. является машинно-зависимым, он называется языком программирования низкого уровня. В данном случае «низкий уровень» не значит «плохой». Имеется в виду, что операторы языка близки к машинному коду и ориентированы на конкретные команды процессора.

Языками низкого уровня являются все языки ассемблера, которые представляет каждую команду машинного кода, не в виде чисел, а с помощью условных символьных обозначений. Так как наборы инструкций для каждой модели процессора отличаются, конкретной компьютерной архитектуре соответствует свой язык ассемблера, и написанная на нём программа может быть использована только в этой среде.

С помощью языков низкого уровня создаются очень эффективные и компактные программы, поскольку разработчик получает доступ ко всем возможностям процессора, но при этом требуется очень хорошо понимать устройство компьютера, затрачивать большие усилия на отладку и, при этом, программа не может быть перенесена на компьютер с другим типом процессора.

Подобные языки обычно применяют для написания небольших системных приложений, драйверов устройств, модулей стыковки с нестандартным оборудованием, когда важнейшими требованиями становятся компактность, быстродействие и возможность прямого доступа к аппаратным ресурсам.

Языки программирования высокого уровня значительно ближе и понятнее человеку, чем компьютеру. Особенности конкретных компьютерных архитектур в них не учитываются, поэтому создаваемые программы на уровне исходных текстов легко переносимы на другие платформы, для которых создан компилятор этого языка. Разрабатывать программы на языках высокого уровня с помощью понятных и мощных команд проще, а ошибок при создании программ допускается значительно меньше.

Ещё языки программирования можно классифицировать по принципу их работы на две следующие группы: операторные и логические. К логическим языкам относятся Lisp, Prolog и т.д. К операторным относятся такие известные языки, как Basic, Pascal, C и др.

### **Контрольные вопросы**

1. Какие классификации языков программирования вы знаете?
2. Какие языки программирования называются языками низкого уровня?Приведите примеры.
3. Какие языки программирования называются языками высокого уровня? Приведите примеры.
4. В чем особенность логических языков программирования (для ответа используйте материал предыдущего параграфа)?
5. К какому поколению языков программирования относится Python?

### **Упражнения для самостоятельного выполнения**

1. Подберите примеры языков программирования к каждому из пяти поколений.
2. Подготовте сообщение об интерпретаторах и компиляторах. Можно ли классифицировать языки программирования по этому признаку? Используйте дополнительные источники информации.

## **Глава 4 Программирование на языке Python без визуального контроля**

### **Параграф 1 Установка интерпретатора**

Названия языка программирования Python правильно произносить как Пайтон, но чаще используется непосредственный перевод этого слова на русский язык — Питон.

Python – это высокоуровневый скриптовый язык программирования интерпритирующего типа. Это достаточно простой для изучения язык и используя данное пособие вы сможете самостоятельно освоить программирование на нём. Python входит в школьную программу, он используется в контрольно-измерительных материалах единого государственного экзамена по информатике и ИКТ. Python хорошо подойдёт в качестве первого языка программирования для изучения.

Python обычно используется как интерпретируемый, но может использоваться и как компилирующий. По производительности интерпретируемый Python похож на все остальные подобные языки, но возможность компиляции в байт-код позволяет добиться большей производительности.

Основные факторы в пользу языка Python следующие:

Качество программного обеспечения. Для многих основное преимущество языка Python заключается в удобочитаемости, ясности и более высоком качестве, отличающими его от других инструментов в мире языков скриптов. Программный код на языке Python читается легче, а значит, многократное его использование и обслуживание выполняется гораздо проще, чем использование программного кода на других языках.

Высокая скорость разработки. По сравнению с компилирующими или строго типизированными языками, такими как C, C++ и Java, Python во много раз повышает производительность труда разработчика. Объем программного кода на языке Python обычно составляет треть или даже пятую часть эквивалентного программного кода на языке C++ или Java. Это означает меньший объем ввода с клавиатуры, меньшее количество времени на отладку и меньший объем трудозатрат на сопровождение.

Переносимость программ. Большая часть программ на языке Python выполняется без изменений на всех основных платформах. Перенос программного кода из операционной системы Linux в Windows обычно заключается в простом копировании файлов программ с одной системы на другую. Более того, Python предоставляет массу возможностей по созданию переносимых графических интерфейсов, программ доступа к базам данных, веб-приложений и многих других типов программ.

Библиотеки поддержки. В составе Python поставляется большое число собранных и переносимых функциональных возможностей, известных как стандартная библиотека. Эта библиотека предоставляет массу возможностей, востребованных в прикладных программах, начиная от поиска текста по шаблону и заканчивая сетевыми функциями.

Интеграция компонентов. Скрипты Python легко могут взаимодействовать с другими приложениями благодаря разнообразным механизмам интеграции. Эта интеграция позволяет использовать Python для настройки и расширения функциональных возможностей программных продуктов.

Из всех перечисленных факторов наиболее существенными для большинства программистов являются первые два (качество и производительность).

Для обозначения файлов с программным кодом на языке Python программисты часто используют слово «скрипт» вместо слова «программа». В этом пособии термины «скрипт» и «программа» рассматриваются как взаимозаменяемые с некоторым предпочтением термина «программа».

В современной реализации Python компилирует (транслирует) инструкции исходного программного кода в промежуточное представление, известное как байт-код, и затем интерпретирует этот байт-код. Байт-код обеспечивает переносимость программ, поскольку этот формат не зависит от платформы. Однако из-за того что Python не создает двоичный машинный код, некоторые программы могут работать медленнее своих аналогов, написанных на компилирующих языках, таких как C или C++. Для увеличения скорости работы программы на языке Python, достаточно выделить из приложения компоненты, требующие максимальной скорости работы, в виде компилированных расширений и связать их системой скриптов на языке Python.

Python используется не только отдельными пользователями, но также применяется и крупными коммерческими компаниями для создания продуктов, приносящих прибыль. Например:

Компания Google широко использует Python в своей поисковой системе и оплачивает труд создателя Python;

Служба коллективного использования видеоматериалов YouTube в значительной степени реализована на языке Python;

Популярная программа BitTorrent для обмена файлами в пиринговых сетях (peer-to-peer) написана на языке Python;

Такие компании, как Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm и IBM, используют Python для тестирования аппаратного обеспечения;

NASA, Los Alamos, Fermilab, JPL и другие используют Python для научных вычислений;

iRobot использует Python в разработке коммерческих роботизированных устройств;

Хорошо известная вам программа невизуального доступа NVDA написана на Python.

Python может использоваться и распространяться совершенно бесплатно. Как и в случае с другими открытыми программными продуктами, такими как Tcl, Perl, Linux и Apache, можно получить в Интернете полные исходные тексты реализации Python. Нет никаких ограничений на его копирование, встраивание в свои системы или распространение в составе ваших продуктов.

Прежде чем установить интерпретатор Python на компьютер, необходимо скачать установочный файл из сети Интернет. Сделать это лучше всего с основного ресурса по этому языку http://www.python.org. На нём находится и документация по языку, и релизы последних версий, и много другой полезной информации. Обратите внимание, что это англоязычный ресурс.

Итак, для установки Python можно воспользоваться следующим алгоритмом:

1. С помощью обозревателя Internet Explorer (или другого) загрузите сайт   
www.python.org.

2. На загрузившейся странице используя команду «h» перейдите к заголовку «Download» .

3. Двигаясь курсорной стрелкой вниз или нажав клавишу Tab найдите ссылку на последний релиз, например, «Python 3.7.2».

4. На открывшейся странице в таблице выберите нужный дистрибутив, например, «Windows x86-64 executable installer Windows for AMD64», и нажав клавишу Enter, скачайте его на свой компьютер.

Теперь необходимо установить интерпретатор Python на компьютер. Для этого следует поступать по алгоритму:

1. Используя какой-либо файловый менеджер, например, Total Commander, запустите скаченный файл для начала установки.

2. В открывшемся диалоге мастера установки установите флажок «Add Python 3.7 to PATH» (номер версии может быть другим).

3. В том же диалоге, перемещаясь по нему клавишей Tab, нажмите с помощью клавиши «Пробел» кнопку «Installation Now».

4. Далее отвечайте согласием на все предложения мастера установки, нажимая кнопку «Next».

Обратите внимание, что для некоторых релизов интерпретатора Python файл с дистрибутивом может иметь расширение «msi». Последние версии дистрибутивов имеют расширение «exe».

После установки в подменю «Все программы» в главном меню операционной системы появляется дополнительное меню Python, содержащее пять пунктов, обеспечивающих быстрый доступ к наиболее типичным задачам:

запуск IDLE;

чтение документации;

запуск интерактивного сеанса;

чтение стандартных руководств по языку Python в веб-обозревателе;

удаление Python.

После установки интерпретатор Python автоматически регистрирует себя в качестве программы по умолчанию для открытия файлов с расширением «py» (файлы с программами на языке Python).

Желаем вам успехов в освоении и дальнейшем использовании этого замечательного языка!

### **Контрольные вопросы**

1. Как произносится названия языка Python?
2. Назовите несколько основных причин, по которым программисты выбирают язык Python.
3. Назовите несколько известных компаний или организаций, использующих Python.
4. Какие задачи можно решать с помощью языка Python?
5. Чем отличается Python от остальных популярных языков?
6. Каковы основные преимущества языка Python?
7. В чём Python уступает языку C? Как можно преодолеть этот недостаток?
8. Где и как можно получить дистрибутив Python?
9. Расскажите о процессе установки интерпретатора Python на компьютер.

### **Параграф 2 Организация комфортной среды программирования без визуального контроля**

Интерпретатор Python позволяет работать двумя способами: Используя поставляемый в комплекте IDLE или командную строку. IDLE (Integrated Development and Learning Environment) — это интегрированная среда разработки и обучения на языке Python, созданная на этом же языке с помощью библиотеки Tkinter. С некоторыми возможностями этой библиотеки знакомит глава 9 этой книги. Работа в IDLE ориентирована на визуальный интерфейс и при использовании программ невизуального доступа вызывает определенный трудности. Работать в этой среде используя JAWS for Windows неудобно и сопряжено с неоправданной необходимостью преодоления некоторых трудностей.

Гораздо удобнее для программирования без визуального контроля на языке Python использовать командную строку. При дальнейшем изложении материала мы будем ориентироваться именно на этот способ работы. Желающие могут самостоятельно попробовать писать программы в IDLE.

Обратите внимание, что при описании процедуры установки интерпретатора Python в предыдущем параграфе было рекомендовано установить флажок «Add Python 3.7 to path» именно для обеспечения работы с использованием командной строки.

Для дальнейшей работы очень удобно использовать файловый менеджер Total Commander. При необходимости можно использовать другой файловый менеджер, но ниже будут приводится рекомендации по работе именно с Total Commander. Итак, будем считать, что на вашем компьютере установлен Total Commander, а в качестве текстового редактора в нём используется AkelPad.

Первое, что следует сделать – это создать папку, в которой будут хранится разрабатываемые программы. Для создания папки следует перейти в то место, где будет создаваться папка и нажать клавишу F7. Затем в открывшемся диалоге в поле редактирования ввести имя создаваемой папки и нажать клавишу Enter для подтверждения. Обратите внимание, что здесь в поле редактирования тоже будет присутствовать предлагаемое имя. В отличии от «Проводника» это будет не «Новая папка», а имя той папки или того файла, на котором стоял фокус в момент нажатия клавиши F7.

Далее необходимо создать файл, в котором будет располагаться текст вашей программы. Для этого можно поступать по следующему алгоритму:

1. Перейти в ранее созданную рабочую папку.

2. Ввести команду Shift +F4 для создания текстового файла.

3. В раскрывшемся диалоге создания файла в поле редактирования ввести имя создаваемого файла латинскими буквами без пробелов с расширением «py» и, нажав Enter, создать файл будущей программы. Откроется окно редактора AkelPad и в нём будет загружен созданный (пока пустой) файл.

4. В меню «Файл» редактора AkelPad выбрать команду "Сохранить как…».

5. В раскрывшемся диалоге сохранения файла используя клавишу Tab найти поле «Кодировка:».

6. Используя курсорные стрелки выбрать кодировку «65001 (UTF-8)» и нажать Enter.

После выполнения указанных операций курсор окажется в поле редактирования редактора AkelPad, а файл будет сохранен в нужной кодировке. Заметим, что кодировка UTF-8 необходима для того, чтобы текстовые сообщения, генерируемые программой, без ошибок выводились на экран компьютера.

Для удобства работы приведём краткую информацию по использованию файлового менеджера Total Commander.

Total Commander – это программа, по своему назначению аналогичная «Проводнику» Windows. Такие программы называют файловыми менеджерами. В отличии от «Проводника» Total Commander не включен в число приложений, входящих в дистрибутив операционной системы Windows, т.е. его надо устанавливать отдельно.

интерфейс программы разделен по вертикали на две независимые части (панели), на которых отображается список папок и файлов какого-либо носителя. В самом верху окна Total Commander расположено строка меню, как и у большинства приложений Windows. Внизу окна программы отображается Командная Строка.

Способ отображения папок и файлов на левой и правой панелях Total Commander можно настраивать независимо. Вариантов настройки здесь значительно больше, чем в «Проводнике» Windows. Рассмотрим два наиболее популярных режима отображения информации.

1. «Подробный»: имена объектов расположены столбцом. После каждого имени объекта указан его тип, дата создания, размер и т.д. При перемещении вертикальными стрелками JAWS будет озвучивать всю эту информацию. Установить режим «Подробный» можно с помощью клавиатурной команды Ctrl +F2.

2. «Краткий»: в этом случае всё содержимое панели выводиться в несколько столбцов, в которых отображены только имена объектов без какой-либо дополнительной информации. Поскольку имена объектов представлены в виде списка, то при перемещении фокуса вниз, JAWS будет последовательно озвучивать все объекты автоматически переходя из столбца в следующий столбец. С точки зрения пользователя программ невизуального доступа имена объектов расположены в один столбец. Установить этот вид можно клавиатурной командой Ctrl +F1.

Выбрать любой из имеющихся режимов отображения файлов и папок можно через меню «Вид» строки меню программы Total Commander. В этом меню представлены и другие режимы.

В меню «Вид» также можно выбрать режим сортировки объектов. Например, введя клавиатурную команду Ctrl +F4 можно установить режим сортировки «по типу». В этом режиме в самом верху будут располагаться имена папок, а затем имена файлов, упорядоченные в алфавитном порядке по расширению. Т.е. сперва будут идти файлы с расширением com, затем с расширением docx, а затем exe. Если файлы имеют одинаковое расширение, то они будут упорядочены в алфавитном порядке относительно имени.

Можно упорядочить файлы по имени. Для этого следует ввести клавиатурную команду Ctrl +F3. В этом случае приоритет в алфавитном порядке будет отдан имени файла. Если же имена будут одинаковые, то файлы упорядочатся относительно расширения.

Заметим, что устанавливаться режимы будут для той панели, на которой находился фокус. Если повторить команду сортировки вторично, то порядок объектов изменится на обратный. Перемещать фокус с панели на панель можно клавишей Tab.

На каждой панели можно независимо выбрать содержимое какого-либо носителя. Для выбора носителя левой панели используйте клавиатурную команду Alt +F1, а для правой панели – Alt +F2. Раскроется список подключенных к компьютеру носителей информации. Сперва в нем будут идти логические диски «C:», «D:» и т.д. За логическими дисками пойдут имена подключенных к компьютеру флешек. Перемещаясь горизонтальными курсорными стрелками можно выбрать желаемый носитель информации и нажать клавишу Enter. На соответствующей панели будет отображено его содержимое.

Скопировать или переместить объект можно действуя по следующему алгоритму:

1. Откройте на любой панели тот носитель, на который надо скопировать или переместить объект. Используйте команды Alt +F1 или Alt +F2.

2. Откройте папку, в которую следует скопировать или переместить объект. Для входа в папку используйте клавишу Enter.

3. Откройте на противоположной панели тот носитель, на котором находится подлежащий копированию или перемещению объект. Используйте команды Alt +F1 или Alt +F2. Обратите внимание, что при открытии какого-либо носителя на данной панели фокус переходит на эту панель автоматически, т.е. нажимать клавишу Tab нет необходимости.

4. Установите фокус на папку или файл, подлежащий копированию или перемещению. Используйте для этого вертикальные курсорные стрелки и клавишу Enter для входа в подкаталоги.

5. Нажмите клавишу F5 для копирования объекта или F6 для его перемещения. Подтвердите свое желание выполнить операцию нажатием клавиши Enter.

Установив фокус на файл и нажав клавишу F6 можно не только переместить его, но и переименовать. Для этого после нажатия клавиши F6 в появившемся поле редактирования удаляем отображенный там путь перемещения и вводим новое имя файла. Подтверждаем операцию нажатием клавиши Enter. В этом случае файл никуда не переместиться, а будет переименован и останется в том же месте. Точно также можно переименовывать и папки.

С помощью клавиши F8 в Total Commander удаляются папки и файлы. Чтобы удалить какой-либо объект следует установить на него фокус, нажать клавишу F8 и Enter для подтверждения. Для окончательного удаления папок и файлов без помещения их в «Корзину» надо нажимать сочетание клавиш Shift +F8. Если вы случайно нажали F8, закрыть диалог удаления можно клавишей Escape и файлы удалены не будут.

Просмотреть текстовый файл можно с помощью клавиши F3. В этом случае файл можно будет только читать, возможности его редактирования не будет. Для редактирования файла следует пользоваться клавишей F4. При этом файл откроется для редактирования в редакторе AkelPad. Редактор по умолчанию можно заменить на любой другой с помощью настроек Total Commander.

Для выделения группы объектов в Total Commander используется клавиша «Пробел». Перемещаясь вертикальными стрелками по списку папок и файлов, нажимайте «Пробел» на тех объектах, которые нужно выделить. При перемещении фокуса выделение не снимается, т.е. таким образом можно выделять объекты как идущие подряд, так и в разброс. Для выделения всех объектов в папке можно также, как в «Проводнике» использовать комбинацию клавиш Ctrl +A. После выделения группы объектов как и в «Проводнике» Windows, с ней можно проделать любую из перечисленных выше операций.

Приведём список некоторых клавиатурных команд Total Commander:

F1 - Помощь;

F2 - Обновить содержимое панели (то же, что и Ctrl +R);

F3 - Просмотреть файлы (способ и объекты просмотра задаются в настройках);

F4 - Редактировать файлы;

F5 - Копировать;

F6 - Переименовать/переместить;

F7 - Создать папку;

F8 – Удалить (то же, что и DEL);

F9 – открыть меню над текущей панелью;

F10 – Открыть крайнее левое меню / Выйти из меню;

Shift +F1 - Выбрать вид файловой панели / Настроить наборы колонок;

Shift +F2 - Сравнить файловые списки в панелях;

Shift +F4 - Создать новый текстовый файл и загрузить его в редактор;

Shift +F5 - Копировать файлы с переименованием в ту же папку;

Shift +F6 - Переименовать файлы в той же папке;

Alt +F1 - Открыть список дисков для левой панели;

Alt +F2 - Открыть список дисков для правой панели;

Alt +F5 - Упаковать выбранные файлы в архив;

Alt +Shift +F5 - Упаковать с последующим удалением исходных файлов;

Alt +F6 - Распаковать файлы из архива;

Alt +F7 - Поиск файлов;

Alt +F8 - Открыть историю командной строки;

Ctrl +F1 – «Краткий» режим представления файлов;

Ctrl +F2 – «Подробный» режим представления файлов;

Ctrl +F3 - Сортировать по имени;

Ctrl +F4 - Сортировать по расширению;

Ctrl +F5 - Сортировать по дате/времени;

Ctrl +F6 - Сортировать по размеру;

Ctrl +F7 - Без сортировки;

Ctrl +F10 - Показывать в панели все файлы;

Ctrl +F11 - Показывать в панели только программы;

Ctrl +A - Выделить всё;

Ctrl +C - Копировать файлы в буфер обмена;

Ctrl +J - Копировать имя файла в командную строку (то же, что и Ctrl +Enter);

Ctrl +Shift +J - Копировать имя файла с полным путём в командную строку (то же, что и Ctrl +Shift +Enter);

Ctrl +L - Подсчитать занимаемое место (для выбранных файлов/каталогов);

Ctrl +V - Вставить файлы из буфера обмена в текущий каталог;

Ctrl +X - Вырезать файлы в буфер обмена;

Ctrl +Y - Очистить командную строку (то же, что и ESC);

Alt +Enter - Показать диалог свойств файла/каталога;

Ctrl +Shift +Enter - Копировать имя файла с полным путём в командную строку;

ESC - Очистить командную строку;

Shift +Esc - Свернуть окно Total Commander.

Если вы в работе используете брайлевский дисплей, то для знакомства с его командами полезно обратиться к главе 10 этой книги.

### **Контрольные вопросы**

1. Что такое IDLE?
2. Как рекомендуется работать с Python без визуального контроля?
3. Что такое «Командная строка»?
4. Для чего служит программа Total Commander?
5. Как создать папку в Total Commander?
6. Расскажите, как создать файл для программы на языке Python в Total Commander?
7. Зачем нужно устанавливать кодировку UTF-8?
8. Какие возможности Total Commander вы знаете?

### **Параграф 3 Обзор Интернет-ресурсов по Python и рекомендации по их использованию**

В этом параграфе приведена информация для тех, кто хочет более глубоко изучить возможности языка Python и продолжить освоение программирования на нём. Ниже указаны несколько Интернет-ресурсов, на которых можно найти необходимую для этого информацию, причём её доступность отвечает требованиям комфортной работы с программой JAWS for Windows.

1. https://pythonworld.ru/samouchitel-python/ На этом ресурсе находится достаточно понятный и полный самоучитель языка Python. Самоучитель ориентирован на неподготовленного читателя. В нём достаточно подробно рассматриваются основные возможности языка. Однако, следует отметить, что стиль изложения не всегда отвечает требованиям учебной литературы.

2. https://python.ru/ На этом ресурсе можно найти обзоры обновлений языка Python и научные статьи по вопросам программирования, а также , выступления специалистов по Python в прессе, на различных конференциях, симпозиумах и других мероприятиях.

3. https://ru.wikiversity.org/wiki/Курс\_по\_библиотеке\_Tkinter\_языка\_Python/ Здесь находится курс по библиотеке TkInter. Этой библиотеке посвящено достаточно много Интернет-ресурсов, однако предлагаемый выделяется своей структурированностью, простотой и невизуальной доступностью.

Помимо указанных ресурсов в сети Интернет много информации различного характера, посвященной языку Python. Используя поисковую систему можно найти любую дополнительную информацию по вкусу.

Приведём список клавиатурных команд Internet Explorer и программы JAWS for Windows, а также некоторые советы, способные помочь при поиски необходимой информации в сети.

Ctrl +O - открыть диалог «Открыть…» для указания адреса сайта;

Tab - переход к следующей ссылке или элементу управления;

Shift +Tab - переход к предыдущей ссылке или элементу управления;

Enter – переход по ссылке;

Alt +стрелка влево - возврат на предыдущую страницу;

Alt +стрелка вправо - переход на следующую страницу;

Alt +D - перевести фокус в адресную строку;

F5 - обновить экран;

F1 – справка;

Ctrl +F4 - закрыть текущую вкладку;

Alt +F4 - закрыть приложение;

Alt +N – переместить фокус на панель уведомлений;

Alt +Q – закрыть панель уведомлений;

Ins +A - читать адресную строку;

Ins +F7 - отобразить список ссылок в отдельном окне;

Ins +F5 - отобразить список полей форм;

Ins +F6 - отобразить список заголовков;

Ins +F9 - отобразить список фреймов;

Ctrl +Shift +k - управление ориентирами;

A - следующая радиокнопка;

B - следующая кнопка;

C - следующий комбинированный список;

D - следующий отличающийся элемент;

E - следующее поле редактирования;

F - следующее поле формы;

G - следующий графический элемент;

H - следующий заголовок;

I - следующий элемент списка;

J - перейти на заданную строку;

Shift +J – вернуться на строку до перехода;

K - следующий ориентир;

L - следующий список;

M - следующий фрейм;

N - пропустить ссылки;

O - следующий объект;

P - следующий абзац;

Q - следующая основная область;

R - следующая область;

S - следующий подобный элемент;

T - следующая таблица;

U - следующая не посещенная ссылка;

V - следующая посещённая ссылка;

X - следующий флажок;

Y - следующий элемент Span;

Z - следующий элемент Div;

; (Точка с запятой) - следующий элемент с атрибутом «По наведению мыши»;

/ (Косая черта) - следующий элемент с атрибутом «По щелчку»;

\ (Обратная косая черта) - следующая почтовая ссылка;

- (Тире) - следующий разделитель;

= (Равно) - следующий элемент с отношением FlowTo;

Для перехода к предыдущему элементу добавляйте к этим командам клавишу Shift.

Для вывода списка элементов добавляйте к команде сочетание Ctrl +Ins, например, Ctrl +ins +Z выводит список всех элементов Div на странице.

Команды для работы с таблицами

Win +Alt +Стрелка вниз - следующая строка;

Win +Alt +Стрелка вверх - предыдущая строка;

Win +Alt +, (Запятая) - Читать строку;

Win +Alt +Стрелка вправо - следующий столбец;

Win +Alt +Стрелка влево - предыдущий столбец;

Win +Alt +. (Точка) - читать столбец;

Alt +Ctrl +Стрелка вправо - следующая ячейка в строке;

Alt +Ctrl +Стрелка влево - предыдущая ячейка в строке;

Alt +Ctrl +Стрелка вниз - следующая ячейка в столбце;

Alt +Ctrl +Стрелка вверх - предыдущая ячейка в столбце;

Ctrl +Win +J - перейти в ячейку;

Команды для получения справки

Нажмите и удерживайте клавишу Insert, одновременно быстро дважды нажмите F1, откроется окно справки. Чтобы открыть раздел справки для Internet Explorer выберите его в списке и Нажмите клавишу F6.

Для получения справки по элементу, на который указывает фокус, используйте команду Ins +F1 (здесь нажатие однократное).

Для получения справки по клавиатурным командам Internet Explorer используйте команду Ins +W.

Для получения справки по командам JAWS используйте команду Ins +H.

Если вы в работе используете брайлевский дисплей, то для знакомства с его командами полезно обратиться к главе 10 этой книги.

Полезные советы

При загрузке WEB-страницы JAWS сообщает некоторую информацию о ней: количество областей, количество заголовков и количество ссылок. Внимательно слушайте сообщения JAWS при загрузки страницы и получите некоторое первичное представление о ее структуре. Затем, если на странице ест заголовки, можно прочитать их все используя клавишу H или команду Ins +F6.

Если изучаемый интернет-ресурс предполагает диалог с пользователем, то на нем должны быть формы. Форму образуют элементы, на которые пользователь может воздействовать, например, поля ввода, кнопки, флажки и т.д. Проверить наличие форм на странице можно используя клавиатурную команду Ins +F5. Она выведет в отдельное окно все элементы управления формы. Можно также посетить все элементы формы нажимая клавишу F.

После этого вы будете иметь некоторое представление о структуре и содержимом страницы и используя команды навигации сможете приступить к поиску необходимой информации.

Если первичное знакомство показывает, что на странице нет необходимой информации, можно не тратя время на полное ее чтение, переходить к следующей.

Обратите внимание, что при навигации по WEB-документу клавиатура не позволяет вводить символы. Нажатие буквенных клавиш приводит к переходу на соответствующий объект. Но при попадании в поле редактирования клавиатура переключается в режим ввода символов. При этом JAWS воспроизводит характерный звуковой сигнал. При покидании поля ввода JAWS также оповещает об этом воспроизводя соответствующий звуковой сигнал.

### **Контрольные вопросы**

1. Какие Интернет-ресурсы по языку Python вам известны?
2. Какие приёмы навигации по WEB-страницам вам известны?
3. Как получить список команд JAWS для навигации по WEB-страницам?
4. О чём сообщает JAWS при загрузке WEB-страницы?
5. Что такое «Режим форм»?
6. Как пользоваться ориентирами?

## **Глава 5 Линейные алгоритмы**

### **Параграф 1 Простейшие программы на языке Питон**

Программа на языке Python представляет собой обычный текстовый файл, содержащий инструкции Python. Например, следующий файл, с именем script1.py можно назвать программой на языке Python:

print(“Привет! Мир!”)

print(2\*\*50)

Этот файл содержит две инструкции print(), первая из которых выводят строку (текст в кавычках), а вторая выводит результат числового выражения (2 в степени 50) на экран компьютера. Как пользоваться арифметическими выражениями будет рассказано далее. Сейчас мы научимся создавать и запускать простейшую программу.

Итак, начать следует с создания рабочей папки и файла для текста разрабатываемой программы. Как это сделать было подробно описано в предыдущей главе. Напомним, что файлы с программами на языке Python должны иметь расширение «py».

Теперь, указанные выше две инструкции программы следует ввести в окно редактора. После создания файла курсор будет находиться в окне редактирования, где необходимо ввести следующие три строки:

print("Привет! Мир!")

print(2\*\*50)

input()

Зачем нужна третья строка будет объяснено ниже.

Поскольку файл на диске уже создан и его имя и кодировка указаны, можно дать команду сохранения файла с помощью клавиши F2.

Для закрытия редактора можно использовать клавишу Escape. После закрытия окна редактора фокус окажется на имени созданного файла script0.py.

Теперь можно выполнить созданную программу, т.е. выполнить все инструкции в файле от начала до конца. Программы запускаются несколькими способами. Самое простое – это, находясь в окне Total Commander, нажать клавишу Enter на имени файла с текстом программы. Раскроется окно командной строки, в котором ниже приглашения «C:\WINDOWS\py.exe» будут выведены две строки с результатами выполнения двух инструкций print():

Привет! Мир!

1125899906842624

Результат работы программы будет виден только в том случае, если в ней не было допущено ошибок. Когда интерпретатор языка Python находит ошибку, он прекращает работу и выводит на экран сообщение о найденной ошибке. Однако, особенности работы операционной системы Windows таковы, что после прекращения работы интерпретатора окно командной строки с сообщением об ошибке закроется и увидеть это сообщение будет нельзя.

Если же программа не содержала ошибок, то интерпретатор выполнит две её строки с инструкцией Print() и остановится ожидая ввода с клавиатуры. Это ожидание ввода есть результат выполнения инструкции Input(). В этой ситуации можно изучить окно командной строки, где и будет выведен результат работы программы. Когда результаты работы программы будут проверены, нажмите Enter и тогда окно командной строки закроется.

Приведём алгоритм, используя который можно изучать окно с результатами работы программы:

1. Запустите программу нажатием клавиши Enter на её имени в Total Commander.

2. Виртуализируйте окно командной строки введя Alt +Ins +W или дайте команду «JAWS к Pc» введя Ins +- (минус на дополнительной клавиатуре).

3. Используя курсорные стрелки изучите результаты работы программы.

4. Нажмите клавишу Escape для отключения виртуализации окна или включите pc-курсор командой + (плюс на дополнительной клавиатуре).

5. Нажав клавишу Enter (этим будет осуществлён пустой ввод) закройте окно командной строки.

Рассмотрим второй способ запуска программ на языке Python. При использовании этого способа вы сможете увидеть сообщения об ошибках при работе интерпретатора. Для этого можно использовать следующий алгоритм:

1. В программе Total Commander Перейдите в свою рабочую папку, например, «программирование».

2. В этой папке создайте файл start.cmd введя команду Shift +F4.

3. В раскрывшемся окне редактора введите в первой строке имя файла с текстом программы на языке Python, например, script0.py, а на следующей строке введите команду pause.

4. Сохраните файл с помощью команды F2 и закройте редактор клавишей Escape.

Теперь для запуска программы следует нажимать Enter на файле start.cmd. При этом окно командной строки в открытом состоянии будет держать команда pause этого файла. Обратите внимание, что последняя строка input() в файле script0.py уже не нужна, задержит окно на экране команда pause, которая выполнится уже после того, как интерпретатор Python выдаст сообщение об ошибке или результаты работы программы и прекратит работу.

Изучать окно с результатами работы программы можно также опираясь на приведённый выше алгоритм. В дальнейшем вы самостоятельно будете выбирать наиболее удобный способ запуска программ на языке Python.

Рассмотрим несколько примеров программ (скриптов) на языке Python.

Пример 1. Пусть необходимо написать программу, которая выводит на экран имя каталога, в котором установлен Python и название платформы, на которой работает ваш компьютер.

Решение: Для того, чтобы получить доступ к инструментам для выполнения этой задачи, следует подключить к программе модуль «sys». Делается это с помощью инструкции import. Для вывода информации на экран используем функцию print(). Итак, программа должна содержать следующие строки:

import sys

print(sys.exec\_prefix)

print(sys.platform)

Если вы запускали созданную программу вторым способом с использованием файла start.cmd, то на экран будет выведен следующий текст:

C:\Python37

win32

d:\программирование>pause

Для продолжения нажмите любую клавишу...

Обратите внимание, что задержка результатов работы программы на экране осуществляется функцией Pause, она же выводит сообщение «Для продолжения нажмите любую клавишу...».

Модуль Python – это библиотека дополнительных инструментов, здесь с её помощью программа получает каталог установки Python и название платформы. Имена sys.exec\_prefix и sys.platform - это строковые переменные, содержимое которых выводится на экран компьютера. Эти переменные находятся в стандартном модуле sys, который необходимо загрузить с помощью инструкции import.

Пример 2. Пусть необходимо вывести на экран компьютера 10 раз слово «Привет!» и результат возведения числа 2 в степень 10.

Решение: Для вывода строки и результата возведения в степень на экран используем всё ту же функцию print(). Для десятикратного повторения слова умножим строку «Привет!» на десять, а для возведения в степень воспользуемся уже встречавшейся командой \*\* (две звезды).

# Программа выводит 10 раз слово «Привет!» и возводит 2 в степень 10.

Print(“Привет!”\*10)

Print(2\*\*10)

input()

Эта программа содержит в конце строку input(), поэтому её можно запускать нажимая Enter на имени файла с текстом программы.

Символ # (решётка) означает комментарий. Комментарий – это текст, следующий за символом #. Комментарии могут занимать отдельную строку или добавляться в строку с программным кодом, правее его. Текст, следующий за символом #, интерпретатором игнорируется, как комментарий, добавленный для человека. В качестве комментария в текст программы добавляются пояснения для программиста, помогающие ему понять смысл программы.

Обратите внимание, что в этом случае непосредственного запуска программы (без командного файла start.cmd) при наличии ошибок сообщения о них прочитать не удастся.

### **Контрольные вопросы**

1. Что такое интерпретатор Python?
2. Что такое исходный программный код?
3. Что такое байт-код?
4. Что такое PVM?
5. Расскажите, как создать папку в Total Commander.
6. Расскажите, как создать файл для написания программы.
7. Расскажите, какие способы запуска программ на Python вы знаете. В чём их разница?
8. Что такое модуль Python?
9. Как подключить модуль к программе?
10. Как вывести информацию о системе на экран?

### **Упражнения для самостоятельного выполнения**

1. Создайте на диске «D:» рабочую папку «программирование». Наберите каждый пример из этого параграфа в отдельный файл и добейтесь их работоспособности. Все файлы сохраните в созданной папке.
2. Напишите программу, выводящую на экран 8 раз слово «Победа!».
3. Напишите программу, вычисляющую:  
   А) 2 в степени 20;  
   Б) 1\*2\*3\*4\*5\*6.  
   Каждое вычисление реализуйте в отдельном фале и сохраните их в папку «Программирование».

### **Параграф 2 Ввод и вывод данных. Переменные.**

Как вы уже знаете, для вывода информации на экран компьютера в Python используется функция print(). Внутри круглых скобок через запятую записываются элементы для вывода. Пример программы с использованием этой функции был приведён в предыдущем параграфе.

Для ввода данных в программу используется функция input(). Она считывает одну строку, т.е. набранный на клавиатуре пользователем текст вводится в программу после нажатия клавиши Enter. Эта строка попадает в переменную, имя которой указано слева от знака присваивания перед функцией input().

Пример 1. Программа запрашивает имя пользователя, считывает введённое имя и приветствует пользователя.

print(“Как вас зовут?”)

name =input()

print(“Здравствуйте, “, name, “!”)

Обратите внимание, что в конце программы нет строки с пустым вводом для задержки информации на экране компьютера. Если вы не используете командный файл для запуска программы, то эту строку следует добавить к её тексту.

Обычно самые простые программы, используемые в обучении, считывают информацию вводимую пользователем с клавиатуры, обрабатывают её и выводят результат обработки на экран. Информация, вводимая пользователем, хранится в переменных в виде их значений. Для того, чтобы поместить значение в переменную используют оператор присваивания = (равно). Слева от оператора присваивания в программах ставится имя переменной, например, name (как в примере 1). Справа от оператора присваивания может быть функция input() или выражение, вычисляющее присваиваемое значение. В частности, допустимо помещать в одну переменную содержимое другой, например, x =y.

Обратите внимание, что оператор присваивания не является равенством, к которому вы привыкли в курсе математики. Например, выражение X =x +2 с точки зрения математики представляет собой не имеющее корней уравнение, а в Python означает вполне допустимое присваивание. Для выполнения такого присваивания Python возьмёт значение, лежащее в переменной x, прибавит к нему число 2 и положит в ту же переменную x. Исходное значение бывшее в x при этом исчезнет. Заметим, что в этом примере переменная x должна быть числовым объектом, иначе к её значению нельзя будет прибавит число 2.

Формально говоря, в Python нет и переменных. Есть лишь имена, которые связаны с некоторыми объектами. В Python все данные называются объектами. Число 2 представляется объектом типа int, строка “Привет” – это объект типа str. Каждый объект относится к какому-то типу. Строки хранятся в объектах типа str, целые числа хранятся в объектах типа int, действительные числа — в объектах типа float. Тип объекта определяет, какие операции можно выполнять с объектами этого типа. Например, объекты типа int можно перемножать, складывать и, вообще, выполнять любые арифметические действия. Объекты типа str перемножать нельзя, а результат их сложения будет выглядеть иначе, чем результат сложения чисел.

Пример 2. Программа сложения двух значений, вводимых с клавиатуры.

Print(“Введите два значения”)

a =input()

b =input()

c =a +b

print(“Результат ”, c)

Если после запуска программы пользователь введёт в качестве первого значения число 2, а в качестве второго – число 3, то программа выведет «Результат 23».

Это произошло потому, что Python по умолчанию воспринял вводимые значения как объекты типа str. Сложение двух строк Python осуществляет приписыванием к левому слагаемому правого. Говорят, что для двух строк определена операция конкатенации (сложения). Обратите внимание, что в отличии от сложения чисел, сложение строк зависит от порядка следования слагаемых.

Для того, чтобы сложение было выполнено именно над числами, строки a и b следует преобразовать к числовому типу, например, Int. Подробнее об этом рассказано в следующем параграфе.

Ранее мы видели, что строку можно умножать на целое число используя операцию \* (умножения). При этом строка будет повторятся соответствующее количество раз.

Объект другого типа в Python можно преобразовать в строку, которая соответствует этому объекту. Для этого следует использовать функцию str(), передав ей в качестве аргумента объект, преобразуемый в строку.

Строка состоит из последовательности символов. Узнать количество символов (длину строки) можно при помощи функции len().

Пример 3. Программа выводит количество символов в вашем имени и количество цифр в числе, равном 2 в степени 1000.

print("Как вас зовут?")

name =input()

l =len(name)

s =str(2\*\*1000)

print("В вашем имени " +str(l) +" букв")

print(len(s))

Поясним, как работает эта программа.:

1. В первой строке программа выводит на экран фразу «Как вас зовут?».

2. Во второй строке программа с помощью функции input() позволяет пользователю ввести с клавиатуры своё имя и с помощью оператора присваивания помещает введённое имя в переменную name.

3. В третьей строке функция len() вычисляет длину строки name (т.е. вашего имени) и используя оператор присваивания помещает вычисленное значение в переменную l числового типа.

4. Здесь функция str() преобразует числовое выражение 2\*\*1000 в строку s. Это необходимо для того, чтобы далее вычислить длину этой строки (это и есть количество цифр в числе).

5. В пятой строке функция print() выводит на экран строку с информацией о количестве букв в введённом имени. Причём выводимая строка формируется с помощью операции конкатенации трёх составляющих её строк: "В вашем имени ", str(l) и " букв". Обратите внимание, что в первой строке перед закрывающей кавычкой поставлен символ пробела, а в последней строке символ пробела поставлен после открывающей кавычки. Это сделано для того, чтобы при выводе на экран перед и после цифры, означающей количество символов, были пробелы и цифра не сливалась с текстом. Сама цифра перед выводом на экран преобразуется из числового типа в строковый с помощью функции str().

6. В последней строке программы функция print() выводит на экран только количество цифр в вычисленном ранее и превращенном в строку числе. Обратите внимание, что здесь в качестве аргумента функции print() используется функция возвращающая число, а не строку. Подобное использование print() является допустимым, поскольку никакого сложения (конкатенации) объектов разного типа не происходит.

Напомним, что если у функции print() несколько аргументов, то они отделяются друг от друга запятой. Подобное использование нескольких аргументов не является их конкатенацией.

Строки используются для записи текстовой информации, а также произвольных последовательностей байтов. Строки поддерживают порядок размещения составляющих их элементов. Отдельные элементы или целые сегменты сохраняются и извлекаются исходя из позиций в строке. Объекты типа str (строка) позволяют находить срез (slice).

Срез — это извлечение из данной строки одного или нескольких символов (подстроки).

Существует три формы срезов. Самая простая форма среза состоит в извлечении одного символа из строки - s[i]. Такой срез состоит из одного символа строки, который имеет указанный в квадратных скобках номер. Обратите внимание, что в Python нумерация символов в строке начинается с числа 0. Например, если s =”Привет”, то s[0] – это буква «П», s[1] – это буква «р», а s[len(s) -1] – это последний символ строки «т».

Номера символов в строке (а также в других структурах данных) называются индексом. В языке Python индексы реализованы в виде смещений от начала строки и поэтому индексация начинается с 0: первый элемент имеет индекс 0, второй - 1 и так далее.

Если указать отрицательное значение индекса, то номер будет отсчитываться с конца, начиная с номера -1. Т.е. в слове «Привет» s[-1] – это символ «т».

Заметим, что в Python нет отдельного типа для одиночных символов. Каждый объект, который получается в результате описанного выше среза S[i] — это тоже строка типа str.

Если же номер символа в срезе строки S больше либо равен len(S), или меньше, чем -len(S), то при обращении к этому символу строки произойдет ошибка IndexError: string index out of range.

Второй тип среза реализуется с помощью двух индексов разделённых двоеточием. Срез S[a:b] возвращает подстроку из b -a символов, начиная с символа с индексом a, до символа с индексом b, не включая его. Например, если опять воспользоваться строкой s =«Привет», то срез S[1:4] возвратит подстроку «рив». Тот же результат получится, если взять срез S[-5:-2].

Можно использовать как положительные, так и отрицательные индексы в одном срезе, например, S[1:-1] — это подстрока без первого и последнего символов (срез начинается с символа с индексом 1 и заканчивается индексом -1, не включая его).

Если в описанном срезе опустить второй индекс, но оставить двоеточие, то срез берется до конца строки. Например, чтобы взять подстроку, состоящую из всех символов строки, кроме первого (его индекс равен 0), можно взять срез S[1:].

Аналогично, если опустить первый индекс, то можно взять срез от начала строки. Т.Е. выделить все символы, кроме последнего, можно используя срез S[:-1]. Срез S[:] совпадает с самой строкой S.

Любые операции среза со строкой создают новые строки и никогда не меняют исходную строку. В Python строки являются неизменяемыми объектами. Их невозможно изменить, а можно лишь старой переменной присвоить новую строку.

Если задать срез с тремя параметрами S[a:b:d], то третий параметр задает шаг, т.е. будут взяты символы с индексами a, a +d, a +2\*d и т.д. При задании значения третьего параметра, равному 2, в срез попадет каждый второй символ, а если взять значение среза, равное -1, то символы будут идти в обратном порядке. Можно перевернуть строку с помощью среза S[::-1]. Например, используя то же значение строки s =”Привет”, получим «тевирП».

Итак, над объектами типа str можно выполнять следующие операции:

Умножение строки на число;

Конкатенация (сложение, объединение) строк;

Взятие среза.

В языке Python используется динамическая типизация, т.е. типы данных определяются автоматически и их не требуется объявлять в программном коде. Однако, над объектом можно выполнять только те операции, которые применимы к его типу.

Другими словами, в языке Python не требуется объявлять переменные заранее. Переменная создается в тот момент, когда ей присваивается значение, при этом переменной можно присвоить значение любого типа, а при использовании внутри выражения имя переменой замещается ее фактическим значением. Кроме того, прежде чем появится возможность обратиться к переменной, ей должно быть присвоено какое-либо значение.

Таким образом, все описанные выше операции над строками в результате создают новую строку, поскольку строки в языке Python являются неизменяемыми. Например, невозможно изменить строку присвоением значения одной из ее позиций, но можно создать новую строку и присвоить ей то же самое имя.

Пример 4. Программа, получает от пользователя строку и выводит все символы этой строки с четными индексами (поскольку индексация начинается с 0, символы выводятся начиная с первого).

print("Введите строку")

s =input()

s =s[1::2]

print(s)

### **Контрольные вопросы**

1. Как осуществляется ввод и вывод информации в программе на языке Python?
2. Что такое объекты Python?
3. Что такое типы объектов в Python?
4. Какие типы объектов вы знаете?
5. Что такое строка в Python?
6. Какие операции над строками вы знаете?
7. Что такое конкатенация?
8. Что такое срез?
9. Какие виды срезов вы знаете?
10. Как преобразовать какой-либо тип к строке?
11. Расскажите как работает программа, приведённая в примере 4.

### **Упражнения для самостоятельного выполнения**

1. Наберите все программы, приведённые в качестве примеров в этом параграфе. Не забывайте, что если вы запускаете программу непосредственно, то в её конце необходим пустой вызов функции input() для задержки на экране результатов.
2. Напишите программу, которая запрашивает строку и выводит на экран её длину.
3. Напишите программу, которая запрашивает строку и выводит её третий и пятый символы.
4. Напишите программу, которая запрашивает строку и выводит её предпоследний символ.
5. Напишите программу, которая запрашивает строку и выводит пять первых её символов.
6. Напишите программу, которая запрашивает строку и выводит все её символы, кроме двух последних.
7. Напишите программу, которая запрашивает строку и выводит все её символы с нечетными индексами.
8. Напишите программу, которая запрашивает строку и выводит все её символы в обратном порядке.
9. Напишите программу, которая запрашивает строку и выводит все её символы через один в обратном порядке, начиная с последнего.

### **Параграф 3 Арифметические операции**

В языке Python существует несколько числовых типов:

целые числа (числа без дробной части);

Действительные числа (числа с десятичной точкой);

комплексные числа (числа с действительной и мнимой частью);

Рациональные числа (числа, представляемые парой целых чисел – числитель и знаменатель).

Здесь мы рассмотрим два наиболее часто используемых типа: целые числа int и действительные числа float. Эти типы являются базовыми и не требуют подключения дополнительных библиотек.

Целые числа типа int в Python поддерживают набор обычных арифметических операций:

+ (плюс) – сложение;

- (минус) – вычитание;

\* (звёздочка) – умножение;

\*\* (звёздочка звёздочка) – возведение в степень;

// (косая черта косая черта) –целая часть от деления;

% (процент) – остаток от деления.

Обратите внимание, что все перечисленные арифметические операции применяются к целым числам и возвращают целые значения.

Как правило, в программах арифметические действия осуществляются не над конкретными числами, а над переменными, имеющими числовые значения. Как уже говорилось, в Python с формальной точки зрения переменных нет – все они являются объектами. Однако, слово «переменная» прочно вошло в программирование и стало общеупотребительным, поэтому мы будем использовать именно его.

Напомним, что переменная в языке программирования - это область памяти компьютера, предназначенное для хранения значений того или иного типа. Основываясь на типе данных переменной, интерпретатор Python выделяет необходимое количество памяти и решает, что может находится в зарезервированной области.

Пример 1. Программа запрашивает два целых числа и выводит результаты выполнения всех возможных арифметических действий с ними.

print("Введите два целых числа")

x =int(input())

y =int(input())

z =x +y

print("Сумма равна ", z)

z =x -y

print("Разность равна ", z)

z =x\*y

print("Произведение равно ", z)

z =x\*\*y

print("Степень равна ", z)

z =x//y

print("Целая часть равна ", z)

z =x%y

print("Остаток равен ", z)

Будучи применённой к целым числам операция деления / (косая черта) возвращает действительное число типа float, даже если деление осуществилось нацело. Также функция возведения в степень возвращает значение типа float, если показатель степени — отрицательное число. При выводе на экран чисел типа float Python будет показывать дробную часть – несколько цифр после десятичной точки. Если дробная часть равна 0, то после десятичной точки будет выведен 0.

Действительные (вещественные) числа записываются в Python с десятичной точкой, а не с запятой, как принято при записи десятичных дробей в нашей стране. Для записи очень больших или очень маленьких по модулю чисел используется запись с плавающей точкой. В этом случае число представляется в виде произведения некоторой десятичной дроби, называемой мантиссой, и степени числа 10, причём показатель степени является целым (возможно отрицательным) числом. Числа с плавающей точкой в программах на языке Python записываются по следующему правилу: сначала записывается мантисса, затем идёт буква e, затем записывается целое число, являющееся показателем степени числа 10. Пробелы внутри такой записи не допускаются. Например, расстояние от земли до солнца будет записываться как 1.496e11, а масса молекулы воды 2.99e-23.

Обратите внимание, что если необходимо ввести с клавиатуры действительное число, то для преобразования типа следует использовать функцию float().

Пример 2. Программа вычисляет среднее арифметическое двух действительных чисел.

print("Введите два действительных числа")

x =float(input())

y =float(input())

z =(x +y)/2

print("Среднее арифметическое равно ", z)

Если в качестве аргумента функции int() указать действительное число с дробной частью, то она вернёт целое число отбросив дробную часть, т.е. произведёт округление в сторону нуля. В Python существует еще несколько полезных базовых функций для работы с целыми и действительными числами:

abs(x) - возвращает абсолютное значение (модуль) числа x;

round(x, n) - Округляет число x до n знаков после десятичной точки. Если второй аргумент n опущен, то округление производится до ближайшего целого числа, при этом, если дробная часть числа x равна 0.5, то число округляется до ближайшего четного числа;

bin(n) – Преобразует целое число n в строку, содержащую его двоичную запись;

oct(n) – Преобразует целое число n в строку, содержащую его восьмиричную запись;

hex(n) – Преобразует целое число n в строку, содержащую его шестнадцатиричную запись;

len(s) - Возвращает число элементов в объекте s;

ord(с) – возвращает код символа s;

chr(n) - возвращает односимвольную строку, код символа которой равен n.

Пример 3. Программа преобразует введённое целое число в двоичное, восьмиричное и шестнадцатиричное.

print("Введите целое число")

n =int(input())

s1 =bin(n)

s2 =oct(n)

s3 =hex(n)

print("Двоичная запись: ", s1)

print("Восьмиричная запись: ", s2)

print("Шестнадцатиричная запись: ", s3)

Для проведения более сложных вычислений в Python существует много дополнительных функций, собранных в библиотеку (модуль) math. Как вы уже знаете, для использования библиотечных функций в начале программы необходимо подключить соответствующий модуль инструкцией import. Существует несколько способов использования этой инструкции. Рассмотрим их на примерах.

Пример 4. Программа извлекает квадратный корень из введённого числа и округляет его различными способами.

import math

print("Введите число")

x =float(input())

y =math.sqrt(x)

z1 =round(y, 2)

z2 =math.ceil(y)

z3 =math.floor(y)

print("Корень без округления: ", y)

print("Обычное округление: ", z1)

print("Округление вверх: ", z2)

print("Округление вниз: ", z3)

Функции sqrt(), ceil() и floor() содержатся в модуле math, а функция round() является базовой, поэтому перед именем функции round() имя модуля не пишется, а перед остальными тремя следует писать имя модуля math, отделяя его от имени функции точкой.

Если же подключить модуль другим способом, то и перед библиотечными функциями писать имя модуля math будет не нужно. Для этого следует использовать инструкцию подключения модуля как показано в примере 5.

Пример 5. Программа запрашивает угол в градусах и выводит соответствующие значения основных тригонометрических функций этого угла.

from math import \*

print("Введите угол в градусах")

a =float(input())

a =radians(a)

y1 =sin(a)

y2 =cos(a)

y3 =tan(a)

y4 =1/y3

print("Синус =", y1)

print("Косинус =", y2)

print("Тангенс =", y3)

print("Котангенс =", y4)

Обратите внимание, что в конце инструкции подключения модуля math пишется символ \* (звёздочка). Вводимое пользователем значение угла в градусах помещается в переменную a, затем пересчитывается в радианы и помещается в ту же переменную a.

Приведём список некоторых полезных функций библиотеки math:

sqrt(x) – возвращает квадратный корень из числа x;

log(x) – возвращает натуральный логарифм числа x. При вызове с двумя аргументами log(x, b) возвращает логарифм числа x по основанию b;

sin(x) – возвращает синус угла x, задаваемого в радианах;

cos(x) – возвращает косинус угла x, задаваемого в радианах;

tan(x) – возвращает тангенс угла x, задаваемого в радианах;

asin(x) – возвращает арксинус x, выраженный в радианах;

acos(x) – возвращает арккосинус x, выраженный в радианах;

atan(x) - возвращает арктангенс x, выраженный в радианах;

degrees(x) - преобразует угол, заданный в радианах, в градусы;

radians(x) - преобразует угол, заданный в градусах, в радианы;

Pi - константа π;

e – константа e.

### **Контрольные вопросы**

1. Какие числовые типы Python вы знаете?
2. Какие арифметические действия над целыми числами вы знаете?
3. Как ввести в программу целое число?
4. Как ввести в программу действительное число?
5. Что такое число с плавающей точкой?
6. Какие базовые функции Python вы знаете?
7. Какие способы округления действительных чисел вы знаете?
8. Какие способы подключения библиотек (модулей) вы знаете? Чем они отличаются?
9. Какие функции библиотеки math вы знаете?

### **Упражнения для самостоятельного выполнения**

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности. Протестируйте правильность их работы на различных входных данных и изучите с помощью брайлевского дисплея выводимые на экран результаты.
2. Напишите программу, вычисляющую сумму трёх введённых чисел.
3. Напишите программу, вычисляющую площадь прямоугольного треугольника по введённым катетам.
4. n школьников делят k яблок поровну, неделящийся остаток остается в корзинке. Сколько яблок достанется каждому школьнику? Сколько яблок останется в корзинке? Программа получает на вход числа n и k и должна вывести искомое количество яблок (два числа).
5. В некоторой школе решили набрать три новых математических класса и оборудовать кабинеты для них новыми партами. За каждой партой может сидеть два учащихся. Известно количество учащихся в каждом из трех классов. Выведите наименьшее число парт, которое нужно приобрести для них. Программа получает на вход три натуральных числа: количество учащихся в каждом из трех классов.
6. Дано число n. С начала суток прошло n минут. Определите, сколько часов и минут будут показывать электронные часы в этот момент. Программа должна вывести два числа: количество часов (от 0 до 23) и количество минут (от 0 до 59). Учтите, что число n может быть больше, чем количество минут в сутках.
7. С начала суток часовая стрелка повернулась на угол в α градусов. Определите на какой угол повернулась минутная стрелка с начала последнего часа. Входные и выходные данные — действительные числа.
8. С начала суток часовая стрелка повернулась на угол в α градусов. Определите сколько полных часов, минут и секунд прошло с начала суток. Запишите ответ в три переменные и выведите их на экран.
9. Напишите программу, вычисляющую гипотенузу прямоугольного треугольника по двум введённым катетам.
10. Напишите программу, вычисляющую площадь круга по введённому радиусу. Ответ округлите до сотых.
11. Напишите программу, вычисляющую площадь треугольника по:  
    А) стороне и проведённой к ней высоте;  
    Б) двум сторонам и углу между ними;  
    В) трём сторонам.  
    Ответ округлите до целых.
12. Напишите программу, вычисляющую дальность полёта снаряда по его начальной скорости и углу наклона пушки к горизонту. Ответ округлите до метра.

## **Глава 6 Ветвление**

### **Параграф 1 Оператор условного перехода**

рассмотренные ранее примеры программ на языке Python имели линейную структуру, т.е. все инструкции этих программ выполнялись последовательно одна за другой, в соответствии с последовательностью записи. Каждая записанная в тексте программы инструкция обязательно выполнялась в свою очередь. Теперь будет рассмотрена программная реализация алгоритмической структуры ветвления на языке Python.

Пусть, например, необходимо по данному числу x определить его абсолютную величину (модуль). Программа должна вывести на экран значение переменной x, если x>0 или вывести -x в противном случае. Встроенную функцию abs() использовать запрещается. Линейная структура программы нарушается, поскольку в зависимости от справедливости условия x>0 должна быть выведена либо одна, либо другая величина.

Пример 1. Программа вывода модуля числа.

Print(“Вычисление модуля числа”)

x = int(input())

if x > 0:

print(x)

else:

print(-x)

В этой программе используется условная инструкция if (если). После слова if указывается проверяемое логическое условие (x > 0), завершающееся двоеточием. После этого идет блок (последовательность) инструкций, который будет выполнен, если условие истинно, в данном примере это вывод на экран величины x. Затем идет слово else (иначе), также завершающееся двоеточием, и блок инструкций, который будет выполнен, если проверяемое условие неверно, в данном случае будет выведено значение -x.

Условный оператор реализует описанную в главе 2 алгоритмическую конструкцию «ветвление» и изменяет порядок выполнения инструкций программы в зависимости от истинности или ложности некоторого условия. Синтаксическая схема условного оператора языка Python выглядит так:

if «условие»:

Блок инструкций 1

else:

Блок инструкций 2

Здесь «условие» представляет собой некоторое логическое выражение, которое может принять либо истинное либо ложное значение. Истинность условия проверяется интерпретатором языка. Если условие истинно, то выполняется блок инструкций 1, а блок инструкций 2 пропускается; если же условие ложно, то выполняется блок инструкций 2, а блок инструкций 1 пропускается.

Несколько инструкций в единый блок объединяются с помощью одинаковых отступов от начала строки. Обычно отступ создаётся четырьмя пробелами. Например, чтобы в случае истинности условия выполнилось три различных инструкции, запись этих инструкций следует начинать с четырёх пробелов каждую. В случае, если в какой-либо строке блока объединённых инструкций будет лишний пробел или наоборот их будет меньше необходимого, то при запуске программы интерпретатор выдаст ошибку. Рекомендуется использовать отступ в 4 пробела и не рекомендуется использовать в качестве отступа символ табуляции. Это одно из существенных отличий синтаксиса языка Python от синтаксиса большинства языков, в которых блоки выделяются специальными словами, например, нц кц в школьном алгоритмическом языке, begin end в Pascal или фигурными скобками в языке C.

Обратите внимание, что даже если блок инструкций состоит всего из одной инструкции, запись этой инструкции должна начинаться с четырёх пробелов.

Заметим, что для контроля количества пробелов удобно использовать брайлевский дисплей, на нём сразу видно с какой позиции начинается запись инструкции.

Приведём пример программы решения квадратного уравнения, написанной в соответствии с блок-схемой, рассмотренной в главе 2.

Пример 2. Программа решения квадратного уравнения.

print("Программа решения квадратного уравнения")

a =float(input("Введите первый коэффициент: "))

b =float(input("Введите второй коэффициент: "))

c =float(input("Введите третий коэффициент: "))

d =b\*b -4\*a\*c

if d < 0:

print("Действительных корней нет!")

else:

x1 =(-b +d\*\*0.5)/(2\*a)

x2 =(-b -d\*\*0.5)/(2\*a)

print("Корни уравнения:", x1, "и", x2)

На этом примере продемонстрирована ещё одна возможность функции input() – это возможность вывода текстового сообщения. Внутри круглых скобок в кавычках можно указывать текст запроса вводимых данных. Это позволяет обойтись без вызова функции print() для вывода на экран необходимых пояснений для пользователя.

В качестве условия после служебного слова if может применяться любое логическое выражение, результатом которого является либо истина, либо ложь. Это может быть выражения с операциями отношения, логическая переменная и сложное логическое условие, составленное из простых условий с помощью логических связок.

В качестве инструкций после двоеточия может применяться любая инструкция языка Python, в том числе и другой (вложенный) условный оператор if.

Как правило, в качестве проверяемого условия используется результат вычисления одного из следующих операторов отношения (сравнения):

< (Меньше) — условие истинно, если первый операнд меньше второго;

> (Больше) — условие истинно, если первый операнд больше второго;

<= (меньше или равно) - условие истинно, если первый операнд не превосходит второго;

>= (больше или равно) - условие истинно, если второй операнд не превосходит первого;

== (равно) – Условие истинно, если два операнда равны;

!= (неравно) – условие истинно, если два операнда неравны.

Например, условие x\*x < 1000 означает, что значение переменной x умноженное само на себя меньше числа 1000.

В языке Python Операторы сравнения можно объединять в цепочки, например, a ==b ==c или 1 <=x <=10. Это отличает Python от большинства других языков программирования, где для этого нужно использовать логические связки.

Операторы отношения предназначены для сравнения двух величин. Результат сравнения имеет логический (bool) тип. Значения логического типа может быть одним из двух:

True – истина;

False - ложь.

Если преобразовать логическое True к типу int, то получится 1, а преобразование False даст 0. При обратном преобразовании число 0 преобразуется в False, а любое ненулевое число в True.

При преобразовании str в bool пустая строка преобразовывается в False, а любая непустая строка в True.

Обратите внимание, что True и False пишутся с большой буквы. Если забыть про это и написать с малой буквы, то интерпретатор выдаст ошибку и программа не заработает.

Иногда нужно проверить одновременно не одно, а несколько условий. Например, проверить, является ли данное число четным можно используя условие n%2 ==0 (остаток от деления n на 2 равен 0), а если необходимо проверить, что два данных целых числа n и m являются четными, необходимо проверить справедливость одновременно двух условий: n%2 ==0 и m%2 ==0, для чего их необходимо объединить логической связкой and (логическое И):

n%2 ==0 and m%2 ==0

В языке Python существуют стандартные логические операторы:

And - логическое «И»;

or - логическое «ИЛИ»;

not - логическое отрицание «НЕ».

Логическая связка and является бинарным оператором, т.е. оператором с двумя операндами, и имеет вид x and y, где x и y – логические выражения. Оператор and возвращает True тогда и только тогда, когда оба его операнда имеют значение True. Во всех остальных случаях он возвращает False.

Логическая связка or также является бинарным оператором и возвращает True тогда и только тогда, когда хотя бы один операнд равен True. В остальных случаях возвращается False.

Логическая отрицание not является унарным (с одним операндом) оператором и имеет вид not x. Оператор not возвращает True, если операнд равен False, а если операнд равен True, то возвращается False.

Пример 3. Программа проверяет, являются ли оба введённых числа четными.

print("Проверка четности чисел")

x =int(input("Введите первое число: "))

y =int(input("Введите второе число: "))

if x%2 ==0 and y%2 ==0:

print("Оба числа чётные!")

else:

print("среди введённых чисел есть нечётное!")

В условной инструкции может отсутствовать слово else: и последующий блок. Такая инструкция называется неполным ветвлением или краткой формой ветвления. Например, если как в примере 1 необходимо вычислить абсолютную величину (модуль) x, то это можно сделать и следующим образом.

Пример 4. Программа вычисляет модуль числа, используя краткую форму инструкции ветвления и не используя встроенную функцию abs().

Print(“Вычисление модуля числа”)

X =int(input(“Введите число: ”))

if x < 0:

x =-x

print(x)

В этом примере переменной x будет присвоено значение -x, но только в том случае, когда x<0. А инструкция print(x) будет выполнена всегда, независимо от проверяемого условия, поскольку написана без отступа.

В конце параграфа приведём ещё один пример использования условной инструкции.

Пример 5. Программа определяет в какой координатной четверти лежит точка с координатами x и y.

print("В какой четверти лежит точка")

x =int(input("Введите абсциссу точки: "))

y =int(input("Введите ординату точки: "))

if x > 0 and y > 0:

print("Точка в первой четверти!")

if x < 0 and y > 0:

print("Точка во второй четверти!")

if x < 0 and y < 0:

print("Точка в третьей четверти!")

if x > 0 and y < 0:

print("Точка в четвёртой четверти!")

### **Контрольные вопросы**

1. Для чего предназначена условная инструкция?
2. Расскажите, как работает условная инструкция.
3. Каковы две формы записи условной инструкции?
4. Как объединяются несколько инструкций в один блок?
5. Может ли условная инструкция содержать в себе другие условные инструкции?
6. Нарисуйте и объясните блок-схему выполнения условной инструкции.
7. Какие логические связки вам известны? Как они работают?
8. Какие операции отношения вам известны?
9. Что является результатом выполнения операций отношения?
10. Объясните, почему в приведённом примере программы решения квадратного уравнения не подключается модуль math?

### **Упражнения для самостоятельного выполнения**

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности. Протестируйте правильность их работы на различных входных данных и изучите с помощью брайлевского дисплея выводимые на экран результаты.
2. Напишите программу, которая запрашивает число и, если оно положительное, выводит слово «Положительно», а если оно отрицательно, то выводит слово «Отрицательно».
3. Напишите программу, которая запрашивает два числа и выводит их в порядке возрастания.
4. Напишите программу, которая запрашивает три числа и выводит наибольшее из них.
5. Напишите программу, которая запрашивает три числа и выводит их в порядке возрастания.
6. Напишите программу, определяющую имеют ли два введённых с клавиатуры действительных числа одинаковый знак. При этом использовать можно только одну инструкцию ветвления. Результат вывести на экран в виде текста: «Числа одного знака» или «Числа разных знаков».
7. Напишите программу, определяющую принадлежит ли точка M(x, y) кругу радиуса r с центром в точке с координатами a, b. Координаты точки, радиус круга и координаты его центра вводятся с клавиатуры, а результат выдаётся на экран в виде текстового сообщения «Точка попала» или «Точка не попала».
8. Напишите программу, определяющую принадлежит ли точка M(x, y) кольцу с центром в начале координат, внешним радиусом R1 и внутренним радиусом R2. Координаты точки, внешний и внутренний радиусы вводятся с клавиатуры, а результат выдаётся на экран в виде текстового сообщения «Точка попала» или «Точка не попала».
9. Усовершенствуйте программу из примера 5 так, чтобы она отдельно обрабатывала случаи, когда точка попадает на оси координат. При этом должно выдаваться соответствующее сообщение «Точка лежит на оси абсцисс» или «Точка лежит на оси ординат».
10. Напишите программу, определяющую является ли частное введённых чисел a и b, округлённое до ближайшего целого чётным числом. Вывести на экран соответствующее текстовое сообщение.
11. Напишите программу, определяющую могут ли три введённых с клавиатуры числа быть сторонами треугольника. Результат вывести на экран в виде соответствующего текстового сообщения.
12. Напишите программу, которая запрашивает натуральное четырёхзначное число и определяет превосходит ли сумма цифр в записи этого числа их произведение. Результат вывести на экран в виде соответствующего текстового сообщения.
13. Напишите программу, которая запрашивает натуральное число и определяет является ли оно двузначным и кратным числу 3. Результат вывести на экран в виде соответствующего текстового сообщения.
14. Напишите программу, которая запрашивает четырёхзначное натуральное число и выводит сообщение «Число правильное», если сумма двух первых цифр этого числа равна произведению двух последних, и выводит сообщение «Число не правильное» в противном случае.
15. Напишите программу, которая запрашивает трёхзначное натуральное число и выводит сообщение «Число правильное», если каждая последующая его цифра, начиная со старшего разряда, больше предыдущей на 1. В противном случае выводится сообщение «Число не правильное».

### **Параграф 2 Вложенный оператор условного перехода**

Внутри инструкции ветвления можно использовать любые инструкции языка Python, в том числе и другую инструкцию ветвления. Таким образом, в языке Python можно использовать вложенные инструкции ветвления.

Обратите внимание, что при использовании структуры вложения блоки внутренних инструкций имеют вдвое больший отступ, т.е. восемь пробелов. Если во вложенную инструкцию вложить инструкцию третьего уровня, то отступ внутренних блоков, соответственно, увеличится втрое, станет равным двенадцати пробелам. Можно создавать и более глубокие структуры вложения, но в этой книги такие многоуровневые вложения рассматриваться не будут.

Приведем в качестве примера программу из предыдущего параграфа, но использующую вложенные инструкции.

Пример 1. Программа определяет в какой координатной четверти лежит точка с координатами x и y, используя вложенные инструкции if.

print("В какой четверти лежит точка")

x =int(input("Введите абсциссу точки: "))

y =int(input("Введите ординату точки: "))

if x > 0:

if y > 0:

print("Точка в первой четверти!")

else:

print("Точка в четвертой четверти!")

else:

if y > 0:

print("Точка во второй четверти!")

else:

print("Точка в третьей четверти!")

Обратите внимание, что в этом примере в программе используется только три условных инструкции.

Программу, определяющую координатную четверть, можно переписать используя “каскадную“ последовательность инструкций ветвления.

Пример 2. Программа определяет в какой координатной четверти лежит точка с координатами x и y, используя каскадную инструкцию ветвления.

print("В какой четверти лежит точка")

x =int(input("Введите абсциссу точки: "))

y =int(input("Введите ординату точки: "))

if x > 0 and y > 0:

print("Точка в первой четверти!")

elif x > 0 and y < 0:

print("Точка в четвертой четверти!")

elif y > 0:

print("Точка во второй четверти!")

else:

print("Точка в третьей четверти!")

В такой конструкции условия проверяются последовательно и выполняется блок инструкций, соответствующий первому из истинных условий. Если все проверяемые условия в последовательности ложны, то выполняется блок инструкций за ключевым словом else: (если этот блок существует).

Обратите внимание, что в этом примере в третьем по счету условии проверяется только y > 0, поскольку условие x < 0 выполнено заведомо. Ведь, если x > 0, то выполнилось бы первое условие и выполнение всей конструкции прекратилось бы.

Приведём ещё один пример использования каскадной инструкции ветвления.

Пример 3. Программа определяет день недели по его номеру.

print("День недели")

day =int(input("Введите номер дня недели: "))

if day ==1:

print("Понедельник")

elif day ==2:

print("Вторник")

elif day ==3:

print("Среда")

elif day ==4:

print("Четверг")

elif day ==5:

print("Пятница")

elif day ==6:

print("Суббота")

elif day ==7:

print("Воскресенье")

else:

print("Такого дня недели нет!")

Особое внимание при вводе текста программ на языке Python следует уделять отступам блоков инструкций. Правильно расставленные отступы являются не только требованием синтаксиса языка, но и делают программу более наглядной. Однако, при работе без визуального контроля использование отступов становится лёгким и наглядным только при наличии брайлевского дисплея и умелом использовании особого функционала программы JAWS for Windows.

Аналогично модулю math из стандартной библиотеки Python можно импортировать модуль random. Этот модуль предоставляет возможность получения случайных действительных чисел в диапазоне от 0 до 1, случайных целых чисел в заданном диапазоне, случайного выбора элементов последовательности и многое другое. Модуль random может использоваться, например, для разработки компьютерных игр, случайного выбора музыкального трека, в программах статистического моделирования и так далее. Рассмотрим два примера использования этого модуля.

Пример 4. Программа запрашивает у пользователя три числа от 1 до 6 и виртуально бросает игральную кость. Если хотя бы одно из введённых чисел совпадает с выпавшим, то пользователь выиграл, если совпадений нет, то пользователь проиграл.

import random

print("Угадай число")

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

b =int(input("Введите второе число: "))

c =int(input("Введите третье число: "))

k =random.randint(1,6)

if a ==k or b ==k or c ==k:

print("Поздравляю! Вы угадали! Выпало число", k)

else:

print("Вы не угадали, выпало число", k)

Пример 5. Программа реализует игру «камень, ножницы, бумага».

import random

print("Камень, ножницы, бумага.")

word =input("Введите слово: ")

s =random.choice(["камень", "ножницы", "бумага"])

p1 =word =="камень" and s =="ножницы"

p2 =word =="ножницы" and s =="бумага"

p3 =word =="бумага" and s =="камень"

q1 =word =="камень" and s =="бумага"

q2 =word =="ножницы" and s =="камень"

q3 =word =="бумага" and s =="ножницы"

if word ==s:

print("Ничья! Выпало слово", s)

elif p1 or p2 or p3:

print("Поздравляю! Вы выиграли! Выпало слово", s)

elif q1 or q2 or q3:

print("Вы проиграли! Выпало слово", s)

else:

print("Вводить можно только камень, ножницы или бумага!")

Если введённое слово совпадает с сгенерированным программой, то первая проверка в каскадной инструкции ветвления выводит сообщение о ничьей.

Одна из логических переменных p1, p2 и p3 содержит истину в том случае, если введённое пользователем слово является выигрышным по отношению к сгенерированному программой слову, а затем во второй проверке каскадной инструкции ветвления выводится сообщение о победе пользователя, если хотя бы одна из этих переменных содержит истину.

Аналогично, одна из логических переменных q1, q2 и q3 содержит истину, если введённое слово проигрывает сгенерированному и в третьей проверке условия каскадной инструкции выводится сообщение о проигрыше.

Если же пользователь ввёл слово не принадлежащее списку допустимых, срабатывает инструкция print(), стоящая после слова else, и выводится соответствующее сообщение.

Обратите внимание, что внутри круглых скобок функции random.choice() список слов для случайного выбора записывается в квадратных скобках. При этом каждое слово записывается в кавычках и через запятую.

### **Контрольные вопросы**

1. Что такое вложенные инструкции?
2. Что такое блок инструкций?
3. Как надо расставлять отступы для вложенных блоков?
4. Что такое каскадная инструкция ветвления?
5. Расскажите, как работает каскадная инструкция ветвления?
6. Чем отличается использование нескольких инструкций ветвления if от использования каскадной инструкции elif?
7. Для чего служит модуль random?
8. Как можно получить случайное слово из заданного списка?
9. Расскажите, как работает программа из примера 5.

### **Упражнения для самостоятельного выполнения**

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности. Протестируйте правильность их работы на различных входных данных и изучите с помощью брайлевского дисплея выводимые на экран результаты.
2. Напишите программу, которая запрашивает длины трёх отрезков и по формуле Герона вычисляет площадь треугольника с такими сторонами, если он существует. Если треугольник не существует, то на экран выводится соответствующее сообщение.
3. Напишите программу, которая запрашивает два числа, являющиеся размерами прямоугольного отверстия, а также диаметр шара. Если шар с данным диаметром проходит через отверстие, то выдаётся сообщение об успехе, в противном случае выдаётся сообщение о неудаче.
4. Напишите программу, которая запрашивает оценку (число от 2 до 5) и выдает на экран соответствующее сообщение:  
   2 – неудовлетворительно;  
   3 – удовлетворительно;  
   4 – хорошо;  
   5 – отлично.  
   Если введённое число выходит за указанный диапазон, то на экран выдаётся сообщение о не корректном вводе.
5. Напишите программу, выводящую название месяца по введённому номеру (числу от 1 до 12). Если введённое число выходит за указанный диапазон, то на экран выдаётся сообщение о некорректном вводе.
6. Напишите программу, которая запрашивает целое число от -99 до 99 и выводит его на экран прописью с учётом знака. если введённое число вне указанного диапазона, программа предлагает выполнить операцию преобразования самостоятельно.
7. Напишите программу, которая запрашивает три числа и выводит текстовое сообщение о количестве совпадающих среди них:   
   Совпадающих чисел нет;  
   Два совпадающих числа;  
   Совпадают все три числа.
8. Напишите программу, которая по введённым координатам двух клеток шахматной доски выводит сообщение о том, совпадает их цвет или нет. Координаты клетки – это два числа от 1 до 8, являющиеся номером строки (горизонтали) и номером столбца (вертикали). Если введённые координаты выходят за диапазон от 1 до 8, то программа выдаёт на экран сообщение о некорректном вводе.
9. Напишите программу, которая запрашивает координаты двух клеток шахматной доски. Если ладья может за один ход попасть из первой клетки во вторую, то выводится сообщение об успехе, в противном случае выводится сообщение о неудаче. Если введённые координаты выходят за диапазон от 1 до 8, то программа выдаёт на экран сообщение о некорректном вводе.
10. Напишите программу, которая запрашивает координаты двух клеток шахматной доски. Если король может за один ход попасть из первой клетки во вторую, то выводится сообщение об успехе, в противном случае выводится сообщение о неудаче. Если введённые координаты выходят за диапазон от 1 до 8, то программа выдаёт на экран сообщение о некорректном вводе.
11. Напишите программу, реализующую игру «Спортлото». Правила игры можно найти в сети Интернет.

## **Глава 7 Циклы**

### Параграф 1

### Цикл с параметром

Алгоритмическая конструкция цикл уже рассматривалась в главах 1 и 2. Напомним, что цикл состоит из управляющей структуры и многократно повторяющегося набора инструкций, называемого телом цикла. В языке Python Циклические алгоритмические конструкции бывают двух типов:

1. циклы с параметром (или со счетчиком), в которых тело цикла выполняется заранее заданное количество раз;

2. циклы с условием, в которых тело цикла выполняется, пока это условие истинно.

Цикл с условием рассматривается в следующем параграфе, а ниже будет описано использование цикла 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» (для).

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 в блок созданием одинакового отступа в начале каждой из объединяемых строк. Так же, как и в условной инструкции if, договоримся делать отступ из четырех пробелов.

Напомним, что строки символов в языке Python относятся к категории последовательностей, в том смысле, что символы, которые они содержат, имеют определенный порядок следования слева направо. Строка - это представитель класса объектов, называемых последовательностями. В языке Python есть и другие типы, являющиеся последовательностями, например, списки и кортежи.

Как уже говорилось выше, 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 еще инструкции после цикла, выполняются они. В примере 1 после окончания цикла других инструкций нет и программа завершается.

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

Обратите внимание, что в этом примере строка символов заключена в апострофы (одинарные кавычки). В языке Python нет разницы между использованием кавычек или апострофов для заключения строк текста, допустим и тот, и другой вариант. Заметим, также, что в языке Python нет специального типа для хранения единственного символа и одиночные символы хранятся как строки длинны 1.

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

colors ='Красный', 'Оранжевый', 'Желтый', 'Зеленый', 'Голубой', 'Синий', 'Фиолетовый'

for ch in colors:

print(ch)

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

Цикл for 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 определенным шагом.

Для повторения цикла определенное 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\*i)

В теле цикла внутри скобок функции print() расположены четыре аргумента, разделенные запятой. Сперва идёт строка ‘Квадрат числа’, затем переменная i, затем строка ‘равен’ и, наконец, арифметическое выражение i\*I (I умножить на i). 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) задает 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 введенного значения.

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

sum =0

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

sum =sum +i

print('Сумма равна', sum)

Обратите внимание на следующие особенности работы данного примера:

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 задает начальное значение параметра цикла, второй аргумент задаёт значение, до которого будет изменяться параметр цикла (не включая его), а третий аргумент — величину изменения параметра. Например, сделать цикл по всем нечетным 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 задает 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.

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')

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

### Упражнения для самостоятельного выполнения

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности. Протестируйте правильность их работы на различных входных данных и изучите с помощью брайлевского дисплея выводимые на экран результаты.
2. 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.
3. Напишите программу, которая запрашивает слово и выводит на экран количество букв «а» в нём.
4. Напишите программу, запрашивающую натуральное число и выводящую на экран все его делители.
5. 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.
6. Напишите программу, которая запрашивает натуральное число n и выводит на экран количество чисел, кратных 3 и не превосходящих n.
7. Напишите программу, которая запрашивает натуральное число и выводит на экран сумму цифр его записи.
8. Напишите программу, которая запрашивает два натуральных числа и определяет, являются ли они взаимно простыми. На экран выводится соответствующее текстовое сообщение.
9. Напишите программу, которая запрашивает слово и определяет, есть ли в нём удвоенная буква. На экран выводится соответствующее текстовое сообщение.
10. 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…

### Параграф 2 Цикл с проверкой условия

Второй вариант реализации циклической конструкции – 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 определенный 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 исполнения 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 «\*» (звездочка). В первой строке выводится 5 звездочек, во второй – 4 и т.д. В последней пятой строке выводится одна звездочка.

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 передается 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 счетчиком. Заметим, что 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 введённого натурального числа n.

n =int(input('Введите число '))

length =0

while n > 0:

n//=10

length +=1

print('Во введенном 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 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 введенных 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 введенного числа', answer)

Обратите внимание на следующие особенности работы данного примера:

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 нее 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.

Напомним, 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 звездочкой, а в конце она не нужна.

### Контрольные вопросы

1. Какой тип алгоритмической структуры необходимо применить, если последовательность инструкций должна быть выполнена определенное количество раз?
2. Какой тип алгоритмической структуры необходимо применить, если последовательность инструкций должна выполняться или не выполняться в зависимости от некоторого условия?
3. Объясните как работает цикл с проверкой условия.
4. Что такое счетчик цикла?
5. Как работает ветвь else: в циклах?
6. Для чего служит инструкция break?
7. Для чего служит инструкция continue?
8. Какую роль играет символ «\*» (звездочка в инструкции print() примера 2?
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.

### Упражнения для самостоятельного выполнения

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности. Протестируйте правильность их работы на различных входных данных и изучите с помощью брайлевского дисплея выводимые на экран результаты.
2. Запишите алгоритм работы программы перевода десятичного числа в двоичное.
3. Запишите алгоритм работы программы, раскладывающей натуральное число в произведение простых множителей.
4. Напишите программу, которая запрашивает натуральное число и выводит его минимальный делитель, отличный от 1.
5. Напишите программу, которая запрашивает числа и выводит по окончании работы количество введённых чисел. Для окончание работы программы вводится число 0.
6. Напишите программу, которая запрашивает числа и после каждого ввода выводит на экран их произведение. Для окончания работы программы вводится число 0.
7. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран их среднее арифметическое. Для завершения работы программы вводится число 0.
8. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран максимальное из введённых чисел. Для завершения работы программы вводится число 0.
9. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран номер минимального из введённых чисел. Для завершения работы программы вводится число 0.
10. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран сумму четных введённых чисел. Для завершения работы программы вводится число 0.
11. Напишите программу, которая запрашивает числа и по окончании работы выводит на экран количество введённых чисел, больших предыдущего. Для завершения работы программы вводится число 0.
12. 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.

### Параграф 3 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 примере.

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).

Обратите внимание на следующие особенности работы данного примера:

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 whileинструкция 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 нее инструкций программы больше нет.

Циклы являются очень удобным инструментом обработки данных, содержащихся в массивах. Массив — это определенное 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 по отношению к спискам.

Массивы бывают одномерными и многомерными. Размер массива ограничивается только объёмом рабочей памяти компьютера. В информатике массив называется одномерным, если для получения доступа к его элементам достаточно одной индексной переменной.

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.

Обратите внимание, что индексы элементов массива заключаются в квадратные скобки и начинаются с 0.

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 приемы создания и заполнения массивов на примерах.

Пример 2. 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])

Обратите внимание на следующие особенности работы данного примера:

1. Вторая строка x =[] говорит, что в программе будет использоваться массив (список) 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, причем третий параметр (шаг) есть число отрицательное, что позволяет изменять счетчик цикла в обратном порядке от значения n -1 до 0.

Пример 3. 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)

Обратите внимание, что в теле первого цикла второй инструкцией идёт функция print() для контрольного вывода на экран созданного случайным образом массива.

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

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 4. 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.

Пример 5. 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.

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.

### Контрольные вопросы

1. Что такое вложенные циклы?
2. Что такое массивы?
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. Сколько элементов может содержать массив?
8. Расскажите как можно заполнить числами массив.
9. Зачем используется метод append?
10. Расскажите как работает «Пузырьковый алгоритм».
11. Расскажите как работают программы, приведённые в каждом из примеров этого параграфа.

### Упражнения для самостоятельного выполнения

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности. Протестируйте правильность их работы на различных входных данных и изучите с помощью брайлевского дисплея выводимые на экран результаты.
2. Напишите программу, которая сначала запрашивает десять чисел в одномерный массив, а затем складывает отдельно все положительные элементы этого массива, отдельно отрицательные элементы и выводит на экран результаты.
3. 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.
4. 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.
5. 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. Результат выводится на экран.
6. 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 2-х и более положительных чисел находящихся рядом. Результат выводится на экран.
7. 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, то есть 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.
8. 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».
9. 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 нечетное значение. Результат выводится на экран. Если нечётных элементов в массиве нет, то выводится сообщение «Не найдено».
10. 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.

## Глава 8 Рекурсия

### Параграф 1 Создание функций

В этом параграфе будет рассмотрена процедура создания собственных функций, т.е. функций, которые разрабатывает сам программист. По способу использования они ничем не отличаются от изученных ранее встроенных в Python функций, например, таких, как print(), input() или len().

Предположим, что некоторый алгоритм требует несколько раз вычислять наибольшее (максимальное) из трёх чисел. В этом случае программа, реализующая этот алгоритм, должна содержать соответствующий фрагмент кода для определения наибольшего числа. Приведём пример программы, содержащей подобный фрагмент.

Пример 1. Программа вычисляет наибольшее из трёх введённых целых чисел и выводит его на экран.

a =int(input('Введите первое число '))

b =int(input('Введите второе число '))

c =int(input('Введите третье число '))

if a >=b and a >=c:

maximum =a

if b >=a and b >=c:

maximum =b

if c >=b and c >=a:

maximum =c

print('Максимальное число', maximum)

В этом примере программа запрашивает три целых числа и находит максимальное из них, но даже если предположить, что при реализации упомянутого выше алгоритма нет необходимости каждый раз запрашивать эти числа (они будут вычисляться самой программой), то многократно включать в текст программы фрагмент из трёх инструкций ветвления достаточно трудоёмко. В этом случае можно воспользоваться операцией копирования фрагмента текста в текстовом редакторе. Однако, если данный фрагмент содержал ошибку, то при копировании она также продублируется. К тому же, подобное копирование фрагментов программного кода существенно увеличивает размер самой программы и ухудшает её читаемость.

Чтобы избежать многократного использования в тексте программы одного и того же фрагмента кода, в языках программирования существуют функции. Функции — это такие участки кода, которые изолированы от остальной программы и выполняются только тогда, когда вызываются.

Функции обладают некоторыми общими свойствами:

1. Функции принимают один или несколько аргументов (в некоторых случаях функции не имеют аргументов).
2. Функции возвращают одно или несколько значений (в некоторых случаях функции не возвращают значение).

Например, функция len() принимает один аргумент (строку) и возвращает одно значение (её длину). Функция print() принимает переменное число аргументов и ничего не возвращает, а в качестве результата своей работы выводит значение аргументов на экран.

Рассмотрим функцию, которая принимает три числа и возвращает максимальное из них.

Пример 2. Функция с тремя числовыми аргументами, возвращающая максимальное значение из этих трёх чисел.

def maximum(a, b, c):

if a >=b and a >=c:

return a

if b >=a and b >=c:

return b

if c >=b and c >=a:

return c

Обратите внимание на следующие правила оформления функций, разрабатываемых программистом:

1. Функция начинается с ключевого слова def, после которого идёт имя функции с перечисленными в круглых скобках переменными, являющимися аргументами. В конце строки ставится двоеточие «:».

2. Тело функции оформляется в виде блока, т.е. пишется с отступа (4 пробела). В случае вложенных конструкций отступ соответствующим образом увеличивается.

3. Для возвращения значения используется ключевое слово return с указанием переменной (или конкретной константы), содержащей возвращаемое значение. Если функция ничего не возвращает, то ключевое слово return отсутствует.

Для того, чтобы продемонстрировать работу описанной выше функции её следует включить в законченную программу.

Пример 3. Программа вычисляет наибольшее из трёх введённых целых чисел и выводит его на экран. Аналогично примеру 1, но с использованием функции.

def maximum(a, b, c):

if a >=b and a >=c:

return a

if b >=a and b >=c:

return b

if c >=b and c >=a:

return c

a =int(input('Введите первое число '))

b =int(input('Введите второе число '))

c =int(input('Введите третье число '))

print('Максимальное число', maximum(a, b, c))

В этом примере основная программа отделена от функции пустой строкой. Обратите внимание, что описание функции должно идти до её использования в коде основной программы.

Заметим, что в языке программирования Python уже встроена функция max() для определения максимума из любого количества чисел, т.е. эта функция с переменным числом аргументов.

Приведем ещё один пример создания функций.

Пример 4. Программа запрашивает натуральное число и выводит его факториал на экран.

def factorial(x):

res =1

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

res \*=i

return res

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

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

Внутри функции можно использовать переменные, инициализированные вне этой функции. Такие переменные (созданные вне функции, но доступные внутри функции) называются глобальными.

Пример 5. Программа используя функцию f(), не имеющую параметров, выводит с её помощью на экран значение глобальной переменной x, присваиваемое в самой программе после описания функции.

def f():

print(x)

x =1

f()

Здесь переменной x присваивается значение 1, и функция f() выводит это значение на экран, несмотря на то, что в описании функции f() эта переменная не инициализируется. Однако, в момент вызова функции f() переменной x уже присвоено значение, поэтому функция может вывести его.

Если инициализировать переменную внутри функции, использовать её вне функции нельзя. В этом случае Python выдаст ошибку «NameError: name 'x' is not defined». Такие переменные, созданные внутри функции, называются локальными. Эти переменные становятся недоступными после выхода из функции.

Пример 6. Программа не будет работать и выдаст ошибку.

def f():

x =1

f()

print(x)

Если изменить значение глобальной переменной внутри функции, вне функции оно останется прежним. Это сделано в целях защиты глобальных переменных от случайного изменения внутри функции.

Пример 7. Программа изменяет значение глобальной переменной внутри функции.

def f():

x =1

print(x)

x =0

f()

print(x)

В результате работы этой программы на экран будут выведены числа 1 и 0. Несмотря на то, что значение переменной x изменилось внутри функции, вне функции оно осталось прежним.

Таким образом, если внутри функции изменяется значение некоторой переменной, то переменная с данным именем становится локальной, и ее модификация не приведет к изменению глобальной переменной с тем же именем. Другими словами, интерпретатор Python считает переменную локальной для данной функции, если в описании этой функции есть хотя бы одна инструкция, модифицирующая значение данной переменной. При этом даже если инструкция, модифицирующая переменную никогда не будет выполнена, интерпретатор это проверить не может, и переменная все равно считается локальной.

Чтобы внутри функции было возможно изменить значение глобальной переменной, необходимо объявить эту переменную внутри функции как глобальную при помощи ключевого слова global:

Пример 8. Программа изменяет внутри функции значение глобальной переменной.

def f():

global x

x =1

print(x)

x =0

f()

print(x)

При выполнении этой программы на экран будет выведено 1 1, т.к. переменная x объявлена, как глобальная, и ее изменение внутри функции приводит к тому, что и вне функции новое значение будет доступна.

Заметим, что при практическом программировании лучше не изменять значения глобальных переменных внутри функции. Если функция должна поменять значение некоторой переменной, лучше использовать инструкцию return и вернуть необходимое значение в основную программу, где его можно присвоить данной переменной. Если следовать этому правилу, то функции получаются независимыми от кода, и их можно легко копировать из одной программы в другую.

С помощью инструкции return функция может возвращать несколько значений, при этом возвращаемые значения заключаются в квадратные скобки и разделяются запятой, т.е. оформляются в виде списка:

Return [a, b, c]

В этом случае результат вызова функции можно будет использовать во множественном присваивании:

K, l, m =f()

### Контрольные вопросы

1. Что такое функция в программировании?
2. Какие встроенные функции языка Python вам известны?
3. Расскажите, как следует создавать собственную функцию в языке Python.
4. Могут ли в описании функции встречаться вложенные блоки инструкций?
5. Может ли инструкция return встречаться в описании функции несколько раз?
6. Может ли инструкция return вернуть сразу несколько значений?
7. Что такое локальная переменная?
8. Что такое глобальная переменная?
9. Можно ли внутри функции изменять значение глобальной переменной?

### Упражнения для самостоятельного выполнения

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности. Протестируйте правильность их работы на различных входных данных и изучите с помощью брайлевского дисплея выводимые на экран результаты.
2. Напишите программу, запрашивающую три действительных числа и выводящую на экран среднее по значению число. При этом программа должна состоять из функции , выполняющей все вычисления, и вызывающей части, запрашивающей данные и выводящей на экран результат.
3. Напишите программу, запрашивающую три натуральных числа и выводящую на экран наибольшее и наименьшее число из введённых. При этом программа должна состоять из функции , выполняющей все вычисления и возвращающей два значения, а также вызывающей части, запрашивающей данные и выводящей на экран результат.
4. Напишите программу, вычисляющую факториал натурального числа, так, чтобы все вычисления содержались в отдельной функции.
5. Напишите программу, вычисляющую число Фибоначчи по его номеру, так, чтобы все вычисления содержались в отдельной функции.
6. Напишите программу, заполняющую массив целыми числами из диапазона от -1000 до 1000 и выводящую на экран индекс максимального элемента массива. При этом все вычисления должны выполняться отдельной функцией.
7. Напишите программу, заполняющую массив целыми числами из диапазона от 0 до 1000 и выводящую на экран группы индексов, так, чтобы в каждой группе индексы соответствовали совпадающим друг с другом элементам массива. При этом все вычисления должны выполняться отдельной функцией.
8. Напишите программу, которая по одному запрашивает целые числа, вычисляет и выводит на экран их произведение и сумму и завершает работу после того, как будет введено число, совпадающее с одним из ранее введённых. Вычисления произведения и суммы, а также вывод их значений на экран, должно осуществляться отдельной функцией.

### Параграф 2 Примеры применения рекурсии

Предположим, что требуется прочитать некоторый текст, содержащий незнакомые слова. Для выяснения значения этих незнакомых слов необходимо пользоваться толковым словарем. Когда в тексте встречается незнакомое слово, то его объяснение ищется в словаре. В объяснении слова могут, в свою очередь, встретиться незнакомые слова, которые также нужно будет найти в словаре и так далее.

Такую задачу можно решить с помощью следующего алгоритма:

1. Если встретилось незнакомое слово, Найдите его в словаре.

2. Прочитайте статью, объясняющую значение этого слова.

3. Если объяснение понятно, то есть статья не содержит слов, вызывающих затруднение, вернитесь в предыдущий текст и продолжите чтение с места, где было прервано чтение.

4. Если в объяснении встречается незнакомое слово, то прекратите чтение этого объяснения, запомните место прекращения и выясните значение слова, придерживаясь правил 1 - 4.

Такой алгоритм называют рекурсивным, т.е. использующим себя. В рекурсивном алгоритме содержится ссылка на самого себя.

Обратите внимание на следующие две важнейшие особенности рекурсивного алгоритма:

1. Рекурсивный алгоритм должен содержать, по крайней мере одну, терминальную ветвь, т.е. условие окончания алгоритма (пункт 3 примера, приведённого выше).

2. Когда алгоритм доходит до рекурсивной ветви, т.е. обращения к самому себе (пункт 4 из примера, приведённого выше), то процесс выполнения приостанавливается, и запускается новый аналогичный алгоритм, но уже на следующем уровне. Прерванный же алгоритм запоминается. Он будет ждать и начнет исполнение лишь по окончании более высокого по уровню алгоритма. В свою очередь новый алгоритм может приостановиться, войти в режим ожидания и т.д. Таким образом образуется цепочка прерванных процессов, из которых выполняется лишь последний в настоящий момент времени процесс, а по окончании его работы продолжает выполняться предшествовавший ему.

Рекурсия – это такой способ организации вычислительного процесса, при котором функция в ходе выполнения составляющих ее инструкций обращается сама к себе.

При каждом очередном входе в рекурсивную функцию ее локальные параметры размещаются в особом образом организованной области памяти – стеке. Целиком весь процесс выполнен, когда стек опустеет или, другими словами, все прерванные процессы выполнятся.

При использовании рекурсии необходимо обращать особое внимание на условие выхода из рекурсивной функции в нужный момент.

Ещё одним примером рекурсивного алгоритма может служить алгоритм поиска файлов с заданным расширением находящихся в заданном каталоге и всех его подкаталогах:

1. Вывести список всех файлов, находящихся в текущем каталоге и имеющих заданное расширение.

2. Если в текущем каталоге есть вложенные подкаталоги, то обработать каждый из них по этому алгоритму.

Приведенный алгоритм является рекурсивным, поскольку для обработки подкаталога вызывает сам себя.

Рекурсия полезна, прежде всего, в случаях, когда основную задачу можно разделить на подзадачи, имеющие ту же структуру, что и первоначальная задача. Функции, реализующие рекурсивный алгоритм, называются рекурсивными.

Приведём несколько примеров реализации рекурсивных алгоритмов на языке Python.

Пример 1. Программа запрашивает натуральное число и используя рекурсивную функцию вычисляет сумму всех натуральных чисел, не превосходящих введённого.

def sum\_n(n):

if n ==1:

return 1

else:

return sum\_n(n -1) +n

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

print('Сумма натуральных чисел равна', sum\_n(k))

Действительно, сумма первых n натуральных чисел равна сумме первых (n -1) натуральных чисел плюс n, а при n =1 сумма равна 1. Условие n ==1 представляет собой условие выхода из рекурсии.

Пример 2. Программа запрашивает натуральное число и, используя рекурсивную функцию, вычисляет его факториал.

def factorial(n):

if n ==1:

return 1

else:

return factorial(n -1)\*n

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

print('Факториал числа', k, 'равен', factorial(k))

Действительно, факториал числа n равен произведению n на факториал числа (n -1). В свою очередь, факториал числа (n -1) равен произведению (n -1) на факториал числа (n -2) и т.д.

Обратите внимание, что функция вызывает сама себя только в том случае, если значение полученного аргумента не равно единице. Если значение аргумента равно единице, то функция себя не вызывает, а возвращает значение 1, и рекурсивный процесс завершается.

Как и в предыдущем параграфе, описание функции (рекурсивной) отделено от основной программы пустой строкой. Строка пропущена лишь для наглядности и при самостоятельном программировании этого можно не делать.

Не следует путать рекурсию с итерацией – повторяющимся выполнением определённой последовательности операций до тех пор, пока не будет удовлетворяться некоторое условие. Цикл с проверкой условия реализует итерационный алгоритм. Большинство циклических алгоритмов можно реализовать двумя способами: итерацией и рекурсией.

Обратите внимание, что использование рекурсивной формы организации алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы. Это является основным достоинством рекурсивной формы реализации алгоритма.

Однако, у рекурсии есть и существенные недостатки:

1. Если глубина рекурсии очень велика, то программа будет требовать во время выполнения много памяти – это может привести к переполнению стека.

2. Рекурсивные алгоритмы, как правило, выполняются более медленно.

3. При программировании рекурсивных алгоритмов велика вероятность ошибок, способных вынудить программиста к перезагрузке компьютера.

Таким образом, если задача имеет очевидное не рекурсивное решение, то следует выбрать именно его. Рекурсивные функции являются мощным механизмом в программировании, однако, они не всегда эффективны. Часто использование рекурсии приводит к ошибкам. Наиболее распространенная из таких ошибок состоит в том, что рекурсия никогда не заканчивается, т.е. цепочка вызовов рекурсивной функции никогда не завершается и продолжается, пока не закончится свободная память в компьютере.

Приведём две наиболее распространенные ошибки, приводящие к бесконечной рекурсии:

1. Неправильное оформление выхода из рекурсии. Например, если в рекурсивной программе вычисления факториала не организовать проверку n ==1, то функция factorial(1) вызовет функцию factorial(0), далее будет вызвана функция factorial(-1) и т.д. Таким образом, процесс никогда не завершится.

2. Рекурсивный вызов с неправильными аргументами. Например, если функция factorial(n) будет вызывать factorial(n), то также получится бесконечная цепочка вызовов, поскольку значение аргумента не изменилось.

Итак, при разработке рекурсивной функции необходимо прежде всего предусматривать условие завершения рекурсивного процесса и следить за тем, чтобы это условие когда-либо выполнялось и процесс завершался.

Приведём ещё один пример типичной рекурсии. В предыдущей главе уже была рассмотрена задача вычисления членов последовательности чисел Фибоначчи. Эта последовательность была впервые предложена В 1202 году итальянским математиком Леонардом Пизанским, известным под именем Фибоначчи. Напомним, что первый член последовательности равен 1, второй равен 1, а каждый последующий равен сумме двух предыдущих:

1, 1, 2, 3, 5, 8, 13, 21, 34, …

Числа Фибоначчи вычисляются так:

f(1) =1

f(2) =1

f(n) =f(n -1) +f(n -2)

Это позволяет построить рекурсивный алгоритм вычисления чисел Фибоначчи.

Пример 3. Рекурсивная программа вычисления числа Фибоначчи по его номеру.

def fib(n):

if n ==1 or n ==2:

return 1

else:

return fib(n -1) +fib(n -2)

k =int(input('Введите номер числа Фибоначчи '))

print('Число Фибоначчи с номером', k, 'равно', fib(k))

### Контрольные вопросы

1. Что такое рекурсия?
2. Какую функцию называют рекурсивной?
3. Что такое итерация?
4. Какие преимущества имеет рекурсия?
5. Какие недостатки имеет рекурсия?
6. Что следует обязательно предусматривать при разработке рекурсивных функций?
7. Приведите примеры рекурсивных алгоритмов.

### Упражнения для самостоятельного выполнения

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности. Протестируйте правильность их работы на различных входных данных и изучите с помощью брайлевского дисплея выводимые на экран результаты.
2. Напишите рекурсивную программу, запрашивающую натуральное число n и вычисляющую сумму всех дробей вида: 1 +1/2 +1/3 + … +1/n. Результат выводится на экран с округлением до сотых.
3. Напишите рекурсивную программу, запрашивающую основание и показатель степени, вычисляющую данную степень и выводящую результат на экран.

В следующих задачах нет требования использовать рекурсию. Вопрос о её применении остаётся на усмотрение программиста.

1. Напишите программу, которая создаёт массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 1000 включительно. Элемент массива называется хорошим, если это двузначное число, причём цифра в разряде десятков больше, чем цифра в разряде единиц. Программа находит и выводит на экран сумму всех хороших элементов массива. Если хороших элементов не найдено, выводится соответствующее сообщение.
2. Напишите программу, которая создаёт массив из 30 элементов. Элементы массива могут принимать целые значения от 150 до 200 — рост учащихся выпускного класса. Программа вычисляет и выводит на экран средний рост среди учащихся выпускного класса, входящих в школьную баскетбольную команду (в команду входят все учащиеся, чей рост больше 180 сантиметров). Если в классе нет игроков баскетбольной команды, то выводится соответствующее сообщение.
3. Напишите программу, которая создаёт массив из 2019 элементов. Элементы массива могут принимать целые значения от 0 до 1000 включительно. Программа вычисляет и выводит на экран количество таких элементов массива, которые равны среднему арифметическому двух элементов, расположенных сразу после него. Например, в массиве из 6 элементов, равных соответственно 2, 3, 1, 5, 6, 4, есть три таких элемента, они расположены на первом, втором и четвёртом месте и равны 2, 3 и 5. Если таких элементов не найдено, то выводится соответствующее сообщение.
4. Напишите программу, которая создаёт массив из 2019 элементов. Элементы массива могут принимать целые значения от -100 до 100 включительно. Программа вычисляет и выводит на экран сумму тех элементов этого массива, чётность которых совпадает с чётностью максимального элемента. Например, в массиве из 6 элементов, равных соответственно 2, 3, 1, 5, 6, 4, максимальный элемент чётный (6), значит, ответом будет сумма чётных элементов этого массива 2 +6 +4 =12.
5. Напишите программу, которая создаёт массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 10000 включительно. Программа вычисляет сумму элементов массива, меньших 200 и при этом кратных 5, а затем заменяет каждый такой элемент на число, равное найденной сумме. В качестве результата программа выводит на экран изменённый массив, каждый элемент выводится с новой строчки. Например, для исходного массива из шести элементов:  
   204  
   115  
   27  
   20  
   305  
   4  
   программа должна вывести следующий массив:  
   204  
   135  
   27  
   135  
   305  
   4

## Глава 9 Звук и графика на языке Python

Материал этой главы выходит за рамки школьной программы, т.е. носит характер факультативного. Поэтому приступать к его изучению лучше только тогда, когда вы твердо усвоили предыдущий материал и достаточно хорошо владеете невизуальными приёмами работы на персональном компьютере под управлением операционной системой Windows. Работа многих функций, изложенных в этой главе, зависит от версии установленной на вашем компьютере операционной системы, а также от её настроек.

### Параграф 1 Работа со звуком

Для работы со звуком на языке Python необходимо подключение к программе соответствующего библиотечного модуля. Таких модулей существует достаточно много. В этом параграфе будет рассмотрен модуль winsound, входящий в дистрибутив интерпретатора Python. Используя этот модуль можно создавать простейшие программы работы со звуком, однако, для более сложных программ следует использовать другие, устанавливаемые отдельно, библиотеки.

Модуль winsound позволяет организовать звуковой интерфейс для приложений в операционной системе Windows. Он обеспечивает доступ к базовому звуковому оборудованию, предоставляемому платформой операционной системы. Приведём краткое описание нескольких функций, содержащихся в модуле winsound.

winsound.Beep(frequency, duration) – эта функция воспроизводит звук заданной частоты и длительности. Первый параметр frequency задаёт частоту в герцах воспроизводимого звука и должен находиться в диапазоне от 37 до 32767. Второй параметр duration указывает время в миллисекундах, в течении которого будет воспроизводиться звук.

Пример 1. Программа воспроизводит звук с частотой 440 Гц длительности 5 секунд.

import winsound

print('Воспроизведение звука')

winsound.Beep(440, 5000)

Обратите внимание, что имя функции Beep() пишется с большой буквы. Если написать его с малой или все буквы написать большими, то программа не заработает. При вводе текста программы внимательно следите за раскладкой (русская/английская) и за регистром (большие/малые) клавиатуры.

winsound.MessageBeep(type) - эта функция воспроизводит звук, указанный в реестре Windows. Аргумент type определяет, какой именно звук воспроизводить. С помощью реестра операционной системы Windows осуществляется сопоставление приведённых ниже имен и звуков, воспроизводимых в процессе работы через колонки. Если в реестре некоторое имя отсутствует, то воспроизводится системный звук по умолчанию, кроме того случая, когда указан флаг SND\_NODEFAULT. Все операционные системы Win32 должны поддерживать следующие имена:

-1 (минус единица) – «простой звуковой сигнал»;

MB\_ICONASTERISK – звёздочка;

MB\_ICONEXCLAMATION – восклицание;

MB\_ICONHAND – критическая ошибка;

MB\_ICONQUESTION – вопрос;

MB\_OK – обычный звук.

Пример 2. Программа воспроизводит «простой звуковой сигнал».

import winsound

print('Воспроизведение простого звукового сигнала')

winsound.MessageBeep(-1)

winsound.PlaySound(sound, flags) – эта функция воспроизводит записанный на диск звук. Первый аргумент может быть именем файла, системным псевдонимом звука или принимать значение None. Интерпретация этого аргумента зависит от значения флага, являющегося вторым аргументом. Приведём некоторые возможные значения этого флага:

winsound.SND\_FILENAME – означает, что первый параметр указывает на wav-файл;

winsound.SND\_LOOP – повторно воспроизводит звук;

winsound.SND\_ASYNC – позволяет вернуться к исполнению основной программы, воспроизводя звук асинхронно;

winsound.SND\_NODEFAULT – если указанный звук не удаётся найти, то звук , определённый в операционной системе по умолчанию, не воспроизводится;

winsound.SND\_NOSTOP – запрещает прерывать воспроизводимые в данный момент звуки.

Пример 3. Программа запрашивает имя wav-файла и воспроизводит его.

import winsound

filename =input('Введите имя wav-файла ')

winsound.PlaySound(filename, winsound.SND\_FILENAME)

Заметим, что вводить имя wav-файла следует с расширением, причём этот файл должен находиться в той же папке, что и вызывающая его программа. Файлы других звуковых форматов (например, mp3) этой функцией воспроизводиться не будут.

Обратите внимание на следующие особенности, которые необходимо учитывать при работе с модулем winsound:

1. Имена функций пишутся в латинской раскладке с большой буквы и при работе с брайлевским дисплеем к их написанию будут добавлены точки 7 и 8.

2. При работе с брайлевским дисплеем символ подчеркивания «\_» отображается в компьютерном (восьмиточечном) брайле точками 4 +5 +6, а знак «.» (точка) точкой 3.

3. Работа всех примеров существенно зависит от конкретных настроек операционной системы вашего компьютера.

### Контрольные вопросы

1. Для чего нужен модуль winsound?
2. Какие функции для работы со звуком модуля winsound вам известны?
3. Какую роль выполняет каждый из аргументов функции Beep()?
4. Что такое системные звуки?

### Упражнения для самостоятельного выполнения

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности.
2. Найдите в сети Интернет информацию о других модулях языка Python для работы со звуком.
3. Напишите программу, воспроизводящую все системные звуки последовательно.
4. Напишите программу, воспроизводящую два звука одновременно.
5. Напишите программу, имитирующую звук сирены.
6. Оснастите программы из предыдущих параграфов звуковыми оповещениями о ходе работы и о её завершении.

### Параграф 2 Работа с элементарными графическими объектами

В этом параграфе будет коротко рассмотрен один из способов работы с графическими объектами на языке Python.

Так же, как и в случае работы со звуком для работы с графикой необходимо использовать соответствующий библиотечный модуль. В состав дистрибутива Python для этих целей входит модуль tkinter. Это графическая библиотека, позволяющая создавать программы с оконным интерфейсом и использовать разнообразные графические примитивы. Данный модуль является кроссплатформенной библиотекой и может быть использован в большинстве распространённых операционных систем (Windows, Linux, Mac OS X и др.).

В литературе часто встречается написание этой библиотеки Tkinter (с большой буквы) именно так это название писалось ранее. Однако, начиная с версии Python 3.0 библиотека переименована в tkinter (с малой буквы) и при использовании Python версии 3.0 и выше в тексте программ писать следует именно так. В ветви Python 2.X используется вариант написания с большой буквы.

Работа с библиотекой tkinter основано на идеях объектно-ориентированного программирования, что выходит за рамки данной книги. Здесь будут рассмотрены только самые простые примеры без подробного объяснения принципов их работы. Для полного понимания работы с этой библиотекой необходимо освоить принципы объектно-ориентированного программирования.

В tkinter визуальные объекты называются виджетами (widget, от англ. window gadget). Это стандартизированный компонент графического интерфейса, с которым взаимодействует пользователь. Это могут быть кнопки, поля редактирования, меню и др. Здесь будут рассматриваться лишь графические примитивы, такие как эллипс (овал) или прямая линия.

При разработке программ с использованием tkinter следует придерживаться следующего алгоритма:

1. Подключить модуль tkinter. Сделать это можно одним из двух стандартных способов:  
import tkinter  
или  
from tkinter import \*

2. Инициализировать графический интерфейс с помощью инструкции  
root =Tk()

3. Создать поле для рисования графических объектов с помощью инструкции  
canvas =Canvas(root, width=300, height=300)  
Эта инструкция создаст поле размером 300 на 300 пикселей. Изменять размеры поля можно в зависимости от условия решаемой задачи.

4. Разместить поле для рисования в окне с помощью инструкции  
canvas.pack()

5. Создать графический примитив или другой виджет. Например,  
circle =canvas.create\_oval(10, 10, 290, 290, fill =’red’)  
Эта инструкция создаст круг красного цвета вписанный в квадрат с вершинами в точках (10; 10( и (290; 290).

6. Создать бесконечный цикл для задержки изображения на экране  
root.mainloop()

Библиотека tkinter является событийно-ориентированной. В приложениях такого типа имеется главный цикл обработки событий. В tkinter такой цикл реализуется с помощью метода mainloop. Для явного выхода из интерпретатора и завершения цикла обработки событий используется метод quit. В приводимых здесь примерах данный метод применяться не будет, для завершения программы следует использовать команду Alt +F4.

Пример 1. Программа рисует на экране круг красного цвета.

from tkinter import \*

root =Tk()

canvas =Canvas(root, width=300, height=300)

canvas.pack()

circle =canvas.create\_oval(10, 10, 290, 290, fill =’red’)

root.mainloop()

Обратите внимание, что закрывать окно с результатами работы этой программы следует командой Alt +F4.

В окне для рисования графических объектов точка с координатами (0, 0) расположена в левом верхнем углу. Ось x направлена вправо, а ось y – вниз. Очевидно, что координаты в графическом окне отличаются от привычных координат на координатной плоскости. При расчете расположения и размеров геометрических фигур (графических примитивов) следует учитывать это отличие.

С помощью метода canv.create\_polygon() можно рисовать замкнутые ломаные линии (многоугольники). В начале списка аргументов указывается пара целых чисел, определяющих первую точку ломаной. Далее идёт пара чисел, определяющая координаты второй точки и т.д. После перечисления координат вершин ломанной указываются ширина и цвет линии, которой данная ломанная будет изображаться. В частности, с помощью этой функции можно рисовать треугольники, прямоугольники, параллелограммы и др.

Пример 2. Программа рисует на экране параллелограмм, стороны которого не параллельны границам экрана.

from tkinter import \*

width\_line =5

x1 =100

y1 =100

x2 =250

y2 =250

x3 =350

y3 =300

x4 =x1 +(x3 -x2)

y4 =y1 +(y3 -y2)

root =Tk()

canv =Canvas(root, width =1200, height =600, bg ='black')

canv.pack()

canv.create\_polygon(x1, y1, x2, y2, x3, y3, x4, y4, width =width\_line, outline ='#fff')

root.mainloop()

Обратите внимание на следующие особенности работы данного примера:

1. Подключение библиотеки tkinter осуществляется с помощью инструкции  
from tkinter import \*  
что позволяет при использовании функций не указывать название библиотеки.

2. Ширина линии, которой рисуется параллелограмм, содержится в переменной width\_line и присваивается переменной width.

3. Координаты трёх вершин параллелограмма указываются в явном виде, а координаты четвертой вершины вычисляются.

4. Цвет линии, которой рисуется параллелограмм, указывается в виде шестнадцатиричного числа #FFF (ярко-белый цвет).

Для рисования основных графических примитивов используются следующие функции библиотеки tkinter:

canv.create\_line() – рисует прямую линию;

canv.create\_oval()- рисует эллипс (овал);

canv.create\_polygon() – рисует много угольник;

canv.create\_rectangle() – рисует прямоугольник;

canv.create\_text() – выводит текст в графическом режиме.

В функциях рисования графических примитивов используются следующие переменные:

width – ширина линии;

fill – цвет заливки;

outline – цвет линии;

smooth – сглаживание;

text – текст для вывода (только для функции canv.create\_text());

font – шрифт текста (только для функции canv.create\_text()).

Заметим, что функция canv.create\_line() может принимать сколь угодно много параметров. Если параметров более двух, то будет нарисована ломанная линия с началом в первой точке и концом в последней. Отличие от результата работы функции canv.create\_polygon() состоит в том, что первая и последняя точки не соединяются.

В конце параграфа приведём более сложный пример. Для лучшего понимания текста программы, он разделён пустыми строками на логически завершённые блоки и снабжён комментариями.

Пример 3. Программа рисует на координатной плоскости параболу и касательную к ней.

from tkinter import \*

root = Tk()

canv=Canvas(root, width=1220, height=750, bg="black")

canv.pack()

x0 =600

y0 =550

startx =-2.5

endx =2.5

step =0.1

k =80

# Оси координат

width\_line =5

canv.create\_line(10,y0,1200,y0, width =width\_line, fill ='#fff', arrow ='last', arrowshape =(40, 55, 20))

canv.create\_line(x0, 10, x0, 670, width =width\_line, fill ='#fff', arrow ='first', arrowshape =(40, 55, 20))

# Координатная сетка

width\_line =1

xl =x0 -k

xr =x0 +k

for i in range(6):

canv.create\_line(xl, 110, xl, 670, width =width\_line, fill ='#fff')

canv.create\_line(xr, 110, xr, 670, width =width\_line, fill ='#fff')

xl -=k

xr +=k

yt =y0 -k

for i in range(5):

canv.create\_line(80, yt, 1120, yt, width =width\_line, fill ='#fff')

yt -=k

canv.create\_line(80, y0 +k, 1080, y0 +k, width =width\_line, fill ='#fff')

canv.create\_text(x0 -(3\*k//10), y0 +(2\*k//5), text ='O', fill ='#fff', font =('Helvectica', '30'))

canv.create\_text(x0 +k//2, 100, text ='Y', fill ='#fff', font =('Helvectica', '40'))

canv.create\_text(1120, y0 +k//2, text ='X', fill ='#fff', font =('Helvectica', '40'))

# График квадратичной функции

width\_line =3

points =[]

startx +=step

while startx <=endx:

points.append(startx\*k +x0)

points.append(-(startx\*\*2)\*k +y0)

startx +=step

canv.create\_line(points, width =width\_line, fill ='#00f', smooth =1)

# Касательная

points.clear()

x1 =-0.3

x2 =3.3

for i in x1,x2:

points.append(i\*k +x0)

points.append((1 -2\*i)\*k +y0)

canv.create\_line(points, width =width\_line, fill ='#00f')

# Точка на графике

points.clear()

x1 =x0 +k

y1 =y0 -k

points.append(x1 -width\_line\*2)

points.append(y1 -width\_line\*2)

points.append(x1 +width\_line\*2)

points.append(y1 +width\_line\*2)

canv.create\_oval(points, width =1, outline ='#f00', fil ='#f00')

root.mainloop()

### Контрольные вопросы

1. Какая библиотека из дистрибутива интерпретатора Python используется для работы с графикой?
2. Расскажите, какие шаги необходимо выполнить для рисования графических примитивов.
3. Расскажите, в чём разница между двумя способами подключения библиотечных модулей.
4. Зачем нужно использовать метод mainloop()?
5. Расскажите, как устроена система координат графического окна. В чём её отличие от координатной плоскости?
6. Какие функции вывода графических примитивов вам известны?
7. Расскажите, как работает функция рисования прямой линии.
8. В чём разница между функцией canv.create\_line() и функцией canv.create\_polygon()?

### Упражнения для самостоятельного выполнения

1. Наберите все программы, приведённые в примерах этого параграфа, и добейтесь их работоспособности.
2. Напишите программу, рисующую ромб с диагоналями.
3. Напишите программу, рисующую домик с окном.
4. Напишите программу, рисующую правильный шестиугольник, вписанный в окружность.
5. Напишите программу, рисующую пятиконечную звезду красного цвета на белом фоне.

## Глава 10 Справочная информация

### Параграф 1 Справочник по вводу специальных символов с помощью брайлевского дисплея

В этом параграфе будут даны основные приёмы работы с помощью брайлевского дисплея Focus 40 Blue и программы невизуального доступа JAWS for Windows, а также комбинации кнопок (точек) клавиатуры Перкинса для ввода некоторых специфических знаков.

Опишем расположения элементов управления на брайлевском дисплее Focus 40 Blue. На верхней панели дисплея дальше от вас расположена 8-клавишная клавиатура Перкинса. Её кнопки имеют достаточно большие размеры и расположены не в линию, а так, чтобы было удобно размещать на них пальцы при работе. Кнопки-точки расположены по четыре под каждую руку. Под левой рукой расположены точки, считая от центра: под указательным пальцем точка 1, под средним пальцем точка 2, под безымянным пальцем точка 3, под мизинцем точка 7. Под правой рукой, считая от центра, расположены соответственно точки 4, 5, 6 и 8.

Так же на верхней панели дисплея ближе к вам находятся 40 восьмиточечных брайлевских клеток. Именно здесь будет выводится текстовая информация при работе устройства.

Над каждой брайлевской клеткой (ячейкой) расположена маленькая кнопка управления курсором. Эти кнопки также называют кнопками роутинга.

слева и справа от брайлевских ячеек расположены вертикально вытянутые Навигационные кнопки-качельки.

Над кнопками-качельками расположены круглые кнопки переключения режимов. Они находятся в том же ряду, что и кнопки роутинга.

Под брайлевскими ячейками по центру расположена горизонтально вытянутая кнопка Пробел. При работе нажимать ее следует большими пальцами левой или правой руки.

На передней торцевой панели дисплея под пробелом на выступе корпуса расположены кнопки левый Shift и правый Shift. Обратите внимание, что в отличие от обычной клавиатуры, в клавиатурных комбинациях брайлевского дисплея эти кнопки различаются.

Также на передней панели симметрично слева и справа от кнопок Shift расположены кнопки выбора. Они отмечены рельефными кружками.

По краям передней панели симметрично слева и справа находятся левая и правая кнопки панорамирования. Они отмечены двойными рельефными стрелками.

Между кнопками выбора и кнопками панорамирования также симметрично находятся прямоугольные кнопки-качельки.

Нажимать элементы управления, расположенные на передней панели, следует большими пальцами.

Используя кнопки брайлевского дисплея, можно выполнить любую команду операционной системы Windows или текстового редактора, которая выполнима с помощью клавиш обычной клавиатуры. Т.е. можно имитировать нажатия любых клавиш и их сочетаний с помощью кнопок брайлевского дисплея. Для этого используются и кнопки для брайлевского ввода, и кнопки, расположенные на передней торцевой панели дисплея. При записи таких команд кнопки клавиатуры Перкинса будем обозначать цифрами, соответствующими номерам брайлевских точек.

Обычное сочетание клавиш состоит из одного или нескольких модификаторов (Shift, Control, Alt и пр.), а также одной функциональной клавиши, одной буквенно-цифровой клавиши, курсорной стрелки и т.д.

Имитация ввода сочетания клавиш обычной клавиатуры на брайлевском дисплее часто происходит в два последовательных нажатия (двухтактная команда). Сначала надо нажать и отпустить сочетание кнопок брайлевского дисплея, соответствующее одному или нескольким модификаторам. Затем нажать и отпустить сочетание кнопок, соответствующее функциональной клавише, букве, цифре и т.д.

Каждому модификатору соответствует кнопка клавиатуры Перкинса брайлевского дисплея от 1 до 7, причем любой модификатор вводится в сочетании с кнопкой 8 и кнопкой пробел. Ниже приведены соответствия кнопок брайлевского дисплея и клавиш-модификаторов обычной клавиатуры:

Функциональные клавиши – 1;

Insert – 2;

Control – 3;

Windows – 4;

Клавиша JAWS – 5;

Alt – 6;

Shift – 7.

Например, чтобы ввести сочетание клавиш CTRL +Shift +v, следует нажать кнопки 3 +7 +8 вместе с кнопкой пробел, отпустить их, а затем нажать кнопки 1 +2 +3 +6 (Латинская буква v). Обратите внимание, что буква v вводится без точки 8.

Для ввода команд, содержащих функциональные клавиши, следует также выполнить два шага:

1. Одновременно нажать и отпустить кнопки 1 +8 +пробел, при необходимости вместе с ними нажать кнопку, соответствующую какому-либо модификатору.

2. Ввести латинскую букву от a до l, где буква a соответствует функциональной клавише F1, b – F2 и т.д. буква k соответствует F11, а буква l – F12. Латинские буквы в этих командах вводятся «как есть», без каких-либо дополнительных точек.

Например, чтобы ввести сочетание клавиш Ins +F2, следует нажать кнопки 1 +2 +8 +пробел, отпустить их, а затем ввести вторую часть команды 1 +2 (латинская буква «b»). Чтобы ввести команду Alt +F4, следует выполнить двухтактную команду 1 +6 +8 +Пробел, отпустить эти кнопки и ввести вторую часть команды 1 +4 +5 (латинская буква «d»).

Для переключения между русским и английским языками следует использовать команду 2 +3 +4 +5 +7 +Пробел. JAWS каждый раз будет называть включаемую этой командой трансляционную таблицу: «Rus\_Unicode» или «Us\_Unicode», дублируя речевое сообщение на брайлевском дисплее.

Обратите внимание, что в режиме ввода на английском языке латинские буквы и команды, содержащие латинские буквы, вводятся без Точки 8.

Приведем список некоторых комбинаций кнопок брайлевского дисплея для работы с текстовым редактором:

Читать строку - Точки 1 +4 +Пробел;

Перейти на предыдущую строку – Точка 1 +Пробел;

Перейти на следующую строку - Точка 4 +Пробел;

Читать текущее слово - Точки 2 +5 +Пробел;

Перейти на предыдущее слово - Точка 2 +Пробел;

Перейти на следующее слово - Точка 5 +Пробел;

Читать текущий символ - Точки 3 +6 +Пробел;

Перейти на предыдущий символ - Точка 3 +Пробел;

Перейти на следующий символ - Точка 6 +Пробел;

Escape - Точки 1 +3 +5 +6 +пробел;

Alt (для входа в меню) - Правый Shift +Точка 2;

Клавиша контекстного меню - Правый Shift +Точка 2 +Пробел;

CapsLock (включение и выключение режима заглавных букв) - Правый Shift +Точка 7 +Пробел;

BackSpace (удаление предыдущего символа) – Точка 7;

Enter - Точка 8;

Tab - Точки 4 +5 +Пробел;

Shift +Tab - Точки 1 +2 +Пробел;

Home - Точки 1 +3 +Пробел;

End - Точки 4 +6 +Пробел;

Вставить из буфера обмена - Левый Shift +Точки 1 +2 +3 +6;

Скопировать в буфер обмена - Левый Shift +Точки 1 +4;

Вырезать в буфер обмена - Левый Shift +Точки 1 +3 +4 +6;

Отмена последнего действия - Левый Shift +Точки 1 +3 +5 +6;

Удалить - Левый Shift +Точки 1 +4 +5.

С помощью брайлевского дисплея можно вводить не только команды, но и текст. Обратите внимание, что брайлевский дисплей для отображения и для ввода информации использует компьютерный (восьмиточечный) брайль. Он имеет несколько отличий от привычного шеститочечного брайля.

Буквы русского и латинского алфавитов вводятся в соответствии с классической (шеститочечной) системой Брайля, но для ввода латинских букв без переключения раскладки необходимо к букве добавить точку 8. Для ввода большой русской буквы к ней добавляется точка 7, а для ввода большой латинской буквы к ней добавляются, соответственно, точки 7 и 8.

Цифры вводятся сниженными, т.е. каждая брайлевская точка в записи цифры снижается на одну позицию. Например, для ввода цифры 2 нажимайте точки 2 +3, а для ввода цифры 7 – точки 2 +3 +5 +6.

Знаки препинания и специальные символы имеют написание, отличное от классического (шеститочечного) брайля:

«.» (точка) – точка 3;

«,» (запятая) – точка 6;

«:» (двоеточие) – точки 4 +6;

«!» (восклицательный знак) – точка 5;

«?» (вопросительный знак» - точки 1 +4 +5 +6;

«;» (точка с запятой) – точки 2 +3 +7;

«“» (кавычки) – точка 4;

«‘» (апостроф) – точки 2 +3 +6 +7;

«\_» (подчёркивание) – точки 4 +5 + 6;

«#» (решетка) – точки 3 +4 +5 +6;

«+» (плюс) – точки 2 +3 +5 +7;

«/» (косая черта( - точки 3 +4;

«\*» (звездочка) – точки 3 +5 +7;

«%» (процент) – точки 1 +4 +6;

«=» (равно) – точки 1 +2 +3 +4 +5 +6;

«<» (меньше) – точки 5 +6;

«>» (больше) – точки 4 +5;

«$» (доллар) – точки 4 +6 +7;

«@» (собака) – точки 3 +4 +5 +7;

«~» (тильда) – точки 1 +2 +4 +5 +6.

Круглые скобки и знак «-« (минус) пишутся также, как в классическом шеститочечном брайле.

### Параграф 2 Справочник по клавиатурным командам

В этом параграфе приведён список наиболее популярных при работе без визуального контроля клавиатурных команд, вводимых с помощью стандартной клавиатуры.

Клавиатурные команды программы JAWS for Windows:

5 (на дополнительной клавиатуре) – произносит символ под курсором;

5 быстро нажатая дважды – читает символ под курсором фонетически, т.е. произносит имя, начинающееся на данную букву;

4 – произносит символ слева от курсора;

6 – произносит символ справа от курсора;

Ins +5 – произносит текущее слово;

Ins +5 быстро дважды – произносит текущее слово по буквам;

Ins +6 – произносит следующее слово;

Ins +4 – произносит предыдущее слово;

Ins +Стрелка вверх – произносит текущую строку;

Ins +Стрелка вверх дважды быстро – произносит текущую строку по буквам;

Alt +5 – произносит текущее предложение;

Alt +Стрелка вверх – перемещает курсор на начало предыдущего предложения и произносит его;

Alt +Стрелка вниз – перемещает курсор на начало следующего предложения и произносит его;

CTRL +5 – произносит текущий абзац;

Ins +Стрелка вниз – произносит весь текст;

Ins +PageUp – произносит текст от положения курсора до конца строки;

Ins +Shift +стрелка вниз – произносит выделенный текст;

Стрелка влево - Прокрутка назад во время непрерывного чтения;

Стрелка вправо - Прокрутка вперёд во время непрерывного чтения;

PageUp - Увеличить темп речи во время непрерывного чтения;

PageDown - Уменьшить темп речи во время непрерывного чтения;

Alt +Ctrl +PageUp - Временно увеличить темп речи не во время непрерывного чтения;

Alt +Ctrl +PageDown - Временно уменьшить темп речи не во время непрерывного чтения;

Alt +Win +Ctrl +PageUp - увеличить темп речи не во время непрерывного чтения с сохранением;

Alt +Win +Ctrl +PageDown - Уменьшить темп речи не во время непрерывного чтения с сохранением;

Ctrl +Ins +Стрелка вниз - Начать обзорное чтение;

Ctrl +Shift +Ins +Стрелка вниз - Изменить настройки обзорного чтения;

Ins +5 (на основной клавиатуре) - Назвать цвет символа и фона;

Ins +F - Назвать шрифт и параметры форматирования;

Ins +End – произносит верхнюю строку окна;

Ins +PageDown – произносит нижнюю строку окна;

Ins +W – вывести список «горячих клавиш» активного приложения;

Ins +H – вывести список «горячих клавиш» программы JAWS for Windows;

CTRL - Прервать речь.

Обратите внимание, что все цифры в этих командах набираются на дополнительной клавиатуре, расположенной в правой части основной клавиатуры. Иногда ее называют калькулятором. Исключение составляет команда определения цвета фона и символа, цифра 5 в ней набирается на основной клавиатуре.

Команды операционной системы Windows:

Ctrl +стрелка вправо – перемещает курсор на слово вправо, помещает его на первый символ и произносит данное слово;

Ctrl +стрелка влево – перемещает курсор на слово влево, помещает его на первый символ и произносит данное слово;

Home – перемещает курсор в начало строки;

End – перемещает курсор в конец строки;

Стрелка вверх – перемещает курсор на строку вверх и произносит её;

Стрелка вниз –перемещает курсор на строку вниз и произносит её;

Ctrl +стрелка вниз – перемещает курсор на один абзац вниз и произносит его;

Ctrl +стрелка вверх – перемещает курсор на один абзац вверх и произносит его;

Ctrl +Home – перемещает курсор в начало текста;

Ctrl +End – перемещает курсор в конец текста;

Shift +стрелка вправо – выделяет следующий символ;

Ctrl +Shift +стрелка вправо – выделяет следующее слово;

Ctrl +Shift +стрелка вниз – выделяет следующий абзац;

Shift +стрелка влево – выделяет предыдущий символ;

Ctrl +Shift +стрелка влево – выделяет предыдущее слово;

Ctrl +Shift +стрелка вверх– выделяет предыдущий абзац;

Ctrl +X – вырезает выделенный фрагмент и помещает его в буфер обмена;

Ctrl +C – копирует выделенный фрагмент в буфер обмена;

Ctrl +V – вставляет фрагмент в текст из буфера обмена;

Del – удаляет выделенный фрагмент;

Ctrl +Z – отменяет последнее действие.

Обратите внимание на последовательность нажатия клавиш в приведённых клавиатурных командах. Удобнее нажимать их в той последовательности, в какой они даны в тексте книги. Это обезопасит от снятия выделения. Помните, что нажатие любой клавиши при выделенном фрагменте текста может привести либо к снятию выделения, либо к вводу символа и потере выделенного фрагмента.