

№ 4701 МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОМПЬЮТЕРНЫХ НАУК

Кафедра инженерной кибернетики

О.В. Андреева

О.И. Ремизова

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

Учебник

Рекомендовано редакционно-издательским
советом университета



Москва 2022

УДК 004.43
А65

Рецензенты:

канд. техн. наук, доц. *С.В. Никифоров* (РГГУ);
канд. техн. наук, доц. *С.В. Солодов*

Андреева, Ольга Владимировна.

А65 Основы алгоритмизации и программирования на языке Python : учебник / О.В. Андреева, О.И. Ремизова. – Москва : Издательский Дом НИТУ «МИСиС», 2022. – 149 с.
ISBN 978-5-907560-22-2

Рассматриваются вопросы алгоритмизации и программирования. Дается описание современного языка программирования *Python* в объеме, необходимом для иллюстрации основных понятий программирования. Рассматриваются базовые структуры алгоритмов, типовые алгоритмы работы с массивами, основные приемы программирования.

Предназначен для изучения основ алгоритмизации и начального знакомства с программированием на языке *Python* студентами 1-го курса всех направлений подготовки, а также для самостоятельного изучения.

УДК 004.43

ISBN 978-5-907560-22-2

© Андреева О.В.,
Ремизова О.И., 2022
© НИТУ «МИСиС», 2022

ОГЛАВЛЕНИЕ

Предисловие	5
Введение.....	7
Глава 1. Основы алгоритмизации и программирования	8
1.1. Понятие алгоритма	8
1.1.1. Понятие о структурном подходе к разработке алгоритмов	8
1.1.2. Метод пошаговой детализации	11
1.2. Основные понятия программирования	12
1.3. Основы языка Python	16
1.3.1. Переменные. Типы данных.....	16
1.3.2. Константы (литералы)	18
1.3.3. Математические операторы. Выражения	19
1.3.4. Логические операторы. Выражения.....	21
1.3.5. Основные операторы Python	22
1.3.6. Ввод-вывод данных	28
1.3.7. Построение программного кода	35
1.4. Этапы решения задач.....	36
1.5. Среда разработки	39
1.5.1. Создание нового проекта	42
1.5.2. Сохранение проекта.....	43
1.5.3. Открытие существующего проекта.....	43
Глава 2. Разработка программ циклической структуры	45
2.1. Циклы по счетчику	45
2.2. Циклы по условию	52
2.3. Вложенные циклы. Вычисление рядов.....	54
Вопросы для самопроверки	57
Задания для самостоятельного выполнения.....	58
Глава 3. Организация разветвлений. Разветвления в цикле.....	63
3.1. Организация разветвлений. Условный оператор. If-блок.....	63
3.2. Составление программ для обработки потока данных.....	67
Вопросы для самопроверки	69
Задания для самостоятельного выполнения.....	70

Глава 4. Структурированные типы данных. Типовые алгоритмы обработки и их реализация	73
4.1. Списки	73
4.1.1. Одномерные массивы	75
4.1.2. Алгоритмы обработки одномерных массивов	78
4.1.3. Матрицы. Типовые алгоритмы обработки матриц	96
4.2. Словари	111
4.3. Множества	116
4.4. Кортежи	118
Вопросы для самопроверки	118
Задания для самостоятельного выполнения	119
Глава 5. Функции	129
Вопросы для самопроверки	136
Задания для самостоятельного выполнения	137
Заключение	142
Библиографический список	143
Приложение. Основные операторы	144

ПРЕДИСЛОВИЕ

Современное развитие общества предполагает широкое использование компьютерных технологий в различных сферах деятельности.

Умение использовать разнообразные возможности, предоставляемые компьютером, обеспечивают конкурентоспособность специалиста.

Настоящий учебник предназначен для приобретения компетенций в области алгоритмизации задач и разработки приложений с использованием современного языка *Python 3.7*, включенного в среду разработки *.NET FrameWork*, являющуюся средством быстрой разработки приложений. Приобретаемые при изучении этого пособия навыки являются необходимыми для дальнейшего овладения возможностями, предоставляемыми языком, в полном объеме.

В учебнике рассматривается комплекс вопросов, связанных с решением задач на компьютере, включая формализацию постановки, алгоритмизацию, современные методы разработки, отладки и тестирования программ, выбор структуры данных и пр. В каждой главе выделены и систематизированы типовые алгоритмы для решения соответствующих задач, в том числе для обработки текстов и решения других задач «невывчислительного» характера.

Учебник содержит большое количество примеров, облегчающих восприятие и освоение материала, и заданий различной степени сложности для самостоятельного выполнения, что позволит оценивать уровень освоения указанных умений и навыков как минимальный, средний или повышенный.

При этом следует отметить, что в рамках настоящего учебника авторы не ставили цели знакомства со всеми возможностями, предоставляемыми языком *Python*, а рассматривали лишь ограниченное подмножество средств, необходимых для решения предлагаемых задач в рамках изучаемого курса. Дополнительные сведения можно почерпнуть в официальном пособии от *Python Software Foundation*, которое является одним из основных источников информации для разработчиков, использующих *Python*, оно доступно по ссылке <https://www.python.org/psf/>

Учебник содержит введение, пять глав и приложение.

Глава 1 в основном посвящена теоретическим вопросам алгоритмизации и знакомству с базовыми приемами программирования. Здесь дается представление об алгоритмическом методе, используемом при решении задач на компьютере, и современных подходах к разработке алгоритмов и программ; рассматриваются основные аспекты и этапы решения задач на компьютере, излагаются приемы и методы разработки и проверки правильности программ. Также дается краткое описание основных понятий языка *Python*, начальные сведения об использовании интегрированной среды разработки *Python 3.7* и технологии *.NET Framework*.

Главы 2–5 посвящены приобретению практических навыков. В каждой главе приводятся типовые алгоритмы и приемы решения задач соответствующего класса, проиллюстрированные примерами программ, а также контрольные вопросы, которые могут использоваться как для самопроверки, так и для контроля преподавателем.

В учебник включены задания для самостоятельного выполнения трех уровней сложности:

- задания I уровня требуют решения простых задач, которые сводятся к типовым алгоритмам и по существу представляют собой упражнения, направленные на приобретение навыков использования этих средств;
- задания II уровня содержат более сложные задачи, требующие использования типовых алгоритмов в сочетании;
- задания III уровня содержат, как правило, задачи, близкие к реальным, которые требуют самостоятельной модификации типовых алгоритмов, а также некоторого творческого подхода, и рекомендуются для выполнения более подготовленным студентам.

Наличие заданий трех уровней позволяет в значительной степени индивидуализировать процесс обучения.

Учебник согласован с программой курса «Алгоритмизация и программирование» (для студентов института ИТКН), а также программами дисциплины «Информатика» (для студентов остальных направлений).

ВВЕДЕНИЕ

Язык программирования *Python* – один из наиболее популярных и быстро развивающихся языков программирования в мире (третий по популярности в 2019 г., согласно данным *GitHub*). Он применяется для решения самых разных задач, включая веб-программирование и анализ данных. Кроме того, это один из подходящих языков для реализации машинного обучения.

Python – простой язык, но его возможности велики. Изучив его, вы получите в свое распоряжение эффективный инструмент для исследования, обнаружения и даже визуализации данных.

Python является простым в изучении и при этом мощным языком программирования. Он содержит эффективные структуры данных высокого уровня и простой, но эффективный подход к объектно-ориентированному программированию. Элегантный синтаксис и динамический ввод *Python*, а также интерпретируемый характер делают его идеальным языком для сценариев и быстрого развития приложений во многих областях на большинстве платформ.

Объектно-ориентированное программирование по сравнению с традиционными алгоритмическими языками существенно расширяет возможности и облегчает разработку сложных программных систем в значительной степени благодаря наличию большого количества библиотек, при использовании которых отпадает необходимость в написании кодов многих типовых алгоритмов.

Однако при решении научно-технических задач алгоритмическая составляющая является весьма существенной. В настоящем учебнике вопросам алгоритмизации и изучению базовых приемов программирования уделяется значительное внимание. Также рассматриваются и используются и некоторые объектные свойства языка, на наглядных примерах демонстрируются предоставляемые ими новые возможности.

ГЛАВА 1. Основы алгоритмизации и программирования

1.1. Понятие алгоритма

Алгоритмом называется четкое описание последовательности действий, которые необходимо выполнить для решения задачи. Практически решение любой задачи требует получения результата по заданным исходным данным. То есть можно сказать, что алгоритм описывает последовательный процесс преобразования исходных данных в результат.

Разработать алгоритм решения задачи означает разбить задачу на последовательно выполняемые шаги (этапы), причем результаты выполнения предыдущих этапов могут использоваться при выполнении последующих. При этом должны быть четко указаны как содержание каждого этапа, так и порядок выполнения этапов. Отдельный этап (шаг) алгоритма представляет собой либо другую, более простую задачу, алгоритм решения которой разработан ранее, либо должен быть достаточно простым и понятным без объяснений.

Четко сформулированная последовательность правил, описывающих этот процесс, и является алгоритмом.

1.1.1. Понятие о структурном подходе к разработке алгоритмов

Основные структуры алгоритмов – это ограниченный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий.

Приводимые ниже структуры рекомендуются при использовании так называемого структурного подхода к разработке алгоритмов и программ. Структурный подход предполагает использование только нескольких основных структур, комбинация которых дает все многообразие алгоритмов и программ.

К основным структурам относятся следующие.

1. *Следование* (рис. 1.1). Последовательное размещение блоков и групп блоков. В программе реализуется последовательным размещением операторов.

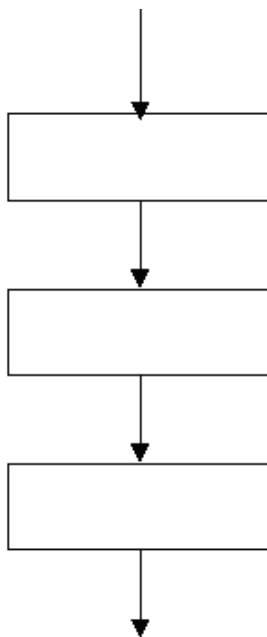


Рис. 1.1. Типовая структура Следование

2. *Цикл*. Применяется при необходимости выполнить какие-либо вычисления несколько раз до выполнения некоторого условия. *Тело цикла* – та последовательность действий, которая выполняется многократно (в цикле). *Начальные присваивания* – задание начальных значений тем переменным, которые используются в теле цикла. Особенность цикла в том, что эта структура может реализоваться в двух вариантах. В первом случае цикл всегда выполняется хотя бы один раз, так как первая проверка условия выхода из цикла происходит после того, как тело цикла выполнено (рис. 1.2, а). Во втором случае проверка условия выхода из цикла производится до выполнения тела цикла, и если при первой проверке условие выполняется, то тело цикла не выполняется ни разу (рис. 1.2, б).

3. *Разветвление* (рис. 1.3) применяется, когда в зависимости от условия нужно выполнить либо одно, либо другое действие. *Действие 1* или *Действие 2* может в свою очередь содержать несколько этапов.

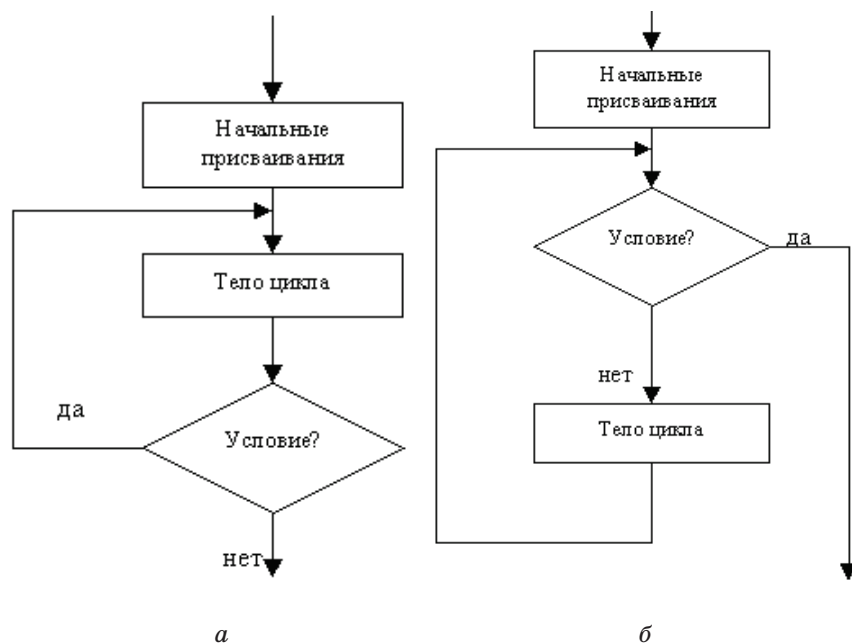


Рис. 1.2. Типовая структура Цикл: *а* – цикл с постусловием; *б* – цикл с предусловием

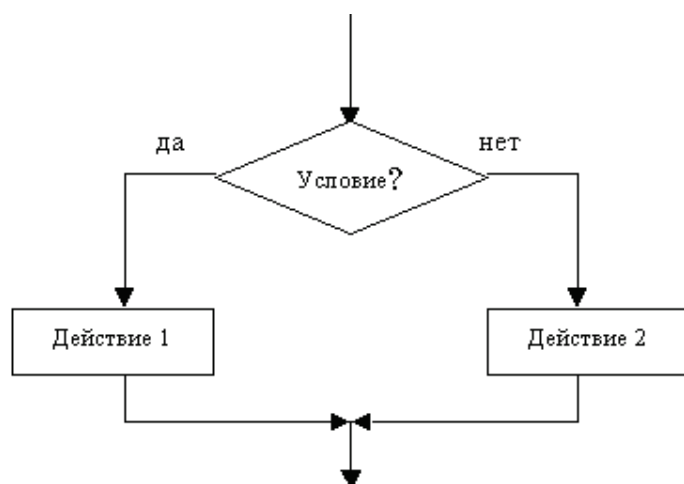


Рис. 1.3. Типовая структура Разветвление

4. *Обход* (рис. 1.4). Частный случай разветвления, когда одна ветвь не содержит никаких действий.

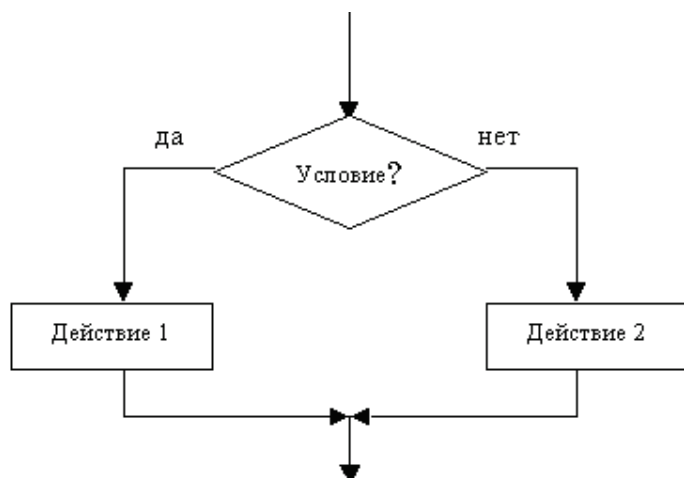


Рис. 1.4. Типовая структура Обход

Особенностью всех приведенных структур является то, что они имеют один вход и один выход, и их можно соединять друг с другом в любой последовательности. В частности, каждая структура может содержать любую другую структуру в качестве одного из блоков.

Обычно при составлении схемы алгоритма (программы) блоки размещаются друг под другом в порядке их выполнения. Возврат назад осуществляется только на циклах. Это дает простую и наглядную схему алгоритма, по которой легко составить программу.

1.1.2. Метод пошаговой детализации

Одним из приемов разработки алгоритма решения более сложных задач является *метод пошаговой детализации*, когда первоначально продумывается и фиксируется общая структура алгоритма без детальной проработки отдельных его частей, но при этом используются лишь основные структуры алгоритмов. Блоки, требующие дальнейшей детализации, обозначают-

ся пунктирной линией. Далее прорабатываются (детализируются) отдельные блоки, не детализированные на предыдущем шаге. То есть на каждом шаге разработки уточняется реализация фрагмента алгоритма (программы), и, таким образом, на каждом шаге мы имеем дело с более простой задачей. Полностью закончив детализацию всех блоков, мы получим решение всей задачи в целом. Описанный метод пошаговой детализации называется также *программированием сверху вниз*.

Однако в некоторых случаях стремление во что бы то ни стало остаться в рамках структурного подхода приводит к необоснованному усложнению программы и потере ее наглядности и естественности. Если учесть, что структурное программирование имеет целью не подчинить программы каким-то правилам, а сделать их более удобными для восприятия, то в ряде случаев оказывается целесообразным отдать предпочтение ясности и естественности программы.

1.2. Основные понятия программирования

Решение любой задачи на компьютере предполагает наличие программы и данных, которые этой программой должны быть обработаны, и в соответствии с этим перед нами встают две задачи: как описать порядок действий, необходимых для решения задачи, и как представить данные (числа и др.) в памяти компьютера.

Самой простой структурой данных является переменная (простая переменная). *Переменную* можно представить себе как ящик с биркой, на которой написано имя (рис. 1.5). (В памяти компьютера это – ячейка памяти, имеющая определенный адрес.) Обращение к переменной осуществляется по имени. В переменной может одновременно храниться одно значение. При занесении в переменную другого значения старое значение пропадает.

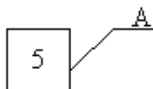


Рис. 1.5. Схематичное представление переменной

Действия записываются с помощью операторов. Существует три основных оператора, с помощью которых (а также их различных сочетаний) можно в принципе описать порядок решения любых задач, которые допускают решение на компьютере. Это оператор присваивания, условный оператор и оператор передачи управления.

Оператор присваивания имеет следующий вид:

Переменная = *выражение*,

где знак «=» обозначает операцию присваивания, *выражение* аналогично знакомому из математики алгебраическому выражению (здесь речь идет о числовом выражении), задающему правило получения значения, а *переменная* – имя переменной, в которую будет помещен результат.

Оператор присваивания выполняется следующим образом: сначала вычисляется *выражение*, стоящее справа, а затем результат помещается в *переменную*, имя которой указано слева от знака операции присваивания.

Например:

$A = 5$	Выражение состоит из одной константы. В переменную A засылается значение 5.
$B = A + 1$	Выражение представляет собой сумму переменной и константы. Если A имеет значение 5, то B будет присвоено значение 6.
$B = B + 1$	Если перед этой операцией переменная B имела значение 6, то после ее выполнения B будет равно 7.
$A = B$	Если B имеет значение 7, то после выполнения этой операции A будет иметь то же значение 7 (старое значение пропадает), при этом значение B не изменяется.

Условный оператор в самом простом случае имеет вид

if (*условие*):
 оператор

Выполняется он следующим образом: если условие удовлетворяется, то выполняется *оператор*, если условие не удов-

летворяется, то *оператор* не выполняется (просто осуществляется переход к следующей по порядку операции).

Например:

```
if (A > 5):  
    A = A - 1
```

Если перед выполнением этого оператора переменная *A* имела, например, значение 7, то условие ($A > 5$) удовлетворяется, оператор после двоеточия выполняется и значение *A* уменьшается на 1 (т. е. станет равно 6). Если же *A* имело бы, например, значение 3, то оператор не будет выполняться и *A* сохранит старое значение.

Отметим, что оператор безусловного перехода *goto*, позволяющий изменить порядок выполнения операторов и перейти при необходимости в любую точку программы, в явном виде в *Python* отсутствует.

Вообще говоря, с помощью перечисленных операторов может быть описано решение любой задачи (допускающей решение на компьютере). Все остальные операторы являются только их расширением и (или) комбинацией. Более полно операторы языка *Python* будут рассмотрены ниже.

Перед составлением программы для решения задачи на компьютере необходимо в первую очередь до конца осмыслить задачу, т.е. понять, что нужно получить в качестве результата и какие для этого имеются исходные данные. После этого программа решения задачи, как правило, может быть укрупненно представлена в виде трех последовательных этапов: ввод исходных данных, преобразование исходных данных в результат, печать результата.

Далее необходимо разработать *алгоритм* решения задачи, т.е. представить процесс ее решения (процесс преобразования исходных данных в результат) в виде последовательности простых шагов (этапов). При этом должны быть четко указаны как содержание каждого этапа, так и порядок выполнения этапов. Отдельный этап (шаг) алгоритма представляет собой либо другую, более простую, задачу, алгоритм решения которой разработан ранее, либо должен быть достаточно простым и понятным без дополнительных пояснений.

Чтобы разработать алгоритм, нужно хорошо представить себе ход решения задачи. При этом полезно решить задачу

самому (на бумаге) для каких-либо наборов данных, не требующих громоздких вычислений, запоминая выполняемые действия так, чтобы далее эти действия формализовать, т.е. записать в виде последовательности четких правил. Понятия алгоритма и программы разграничены не очень четко. Обычно программой называют окончательный вариант алгоритма решения задачи, ориентированный на конкретного исполнителя (в данном случае в качестве исполнителя выступает компьютер).

Решение любой задачи на компьютере может быть описано при помощи ограниченного набора простых операций (см. выше). Эти операции при разработке алгоритма группируются, образуя типичные последовательности действий, называемые *основными структурами алгоритма*. Имеется ограниченный набор таких структур, рекомендуемых при использовании *структурного подхода* к разработке алгоритмов и программ, когда можно получить все многообразие алгоритмов и программ из комбинаций нескольких основных структур.

К основным структурам относятся: *следование* – последовательное размещение блоков и групп блоков; *цикл* – многократное выполнение блока или группы блоков, *разветвление* – после проверки некоторого условия выбирается один из двух или более путей вычислительного процесса, и после выполнения любого из них процесс опять сводится в одно русло.

При разработке алгоритма (и программы) полезно использовать *метод пошаговой детализации*, когда первоначально продумывается и фиксируется общая структура алгоритма (программы) без детальной проработки отдельных его частей, с использованием лишь основных структур алгоритма. Далее детализируются отдельные блоки, требующие дополнительной проработки после предыдущего шага (с использованием также только основных структур алгоритма), т.е. на каждом шаге разработки уточняется реализация фрагмента алгоритма (программы), и, таким образом, на каждом шаге мы имеем дело с более простой задачей. Полностью закончив детализацию всех блоков, мы получим решение всей задачи в целом. Описанный метод пошаговой детализации называется также *программированием сверху вниз*.

Современные алгоритмические языки содержат средства для реализации типовых структур алгоритмов, что позволяет при составлении программ оставаться в рамках структурного подхода, получая в результате правильно написанные программы, готовые к выполнению.

1.3. Основы языка Python

1.3.1. Переменные. Типы данных

Переменная является простейшей структурой данных. *Имя (идентификатор)*, используемое для обозначения переменных (а также других объектов), может содержать латинские буквы (заглавные и строчные буквы в именах различаются) и цифры, а также знак подчеркивания (`_`), при этом начинаться должно с буквы (использование некоторых других символов в рамках данного учебника не рассматривается). Например, `a4`, `alf`, `c_n`, `b`, `B` (последние два имени в программе будут обозначать разные переменные).

Обратите внимание на то, что типы переменных явно не объявляются. Причина в том, что *Python* – это язык с *динамической типизацией*, то есть тип переменной определяется присваиваемыми ей данными, например:

```
a = 12
d = 12.0
```

В *Python* целые числа создаются с помощью встроенного типа данных `int`, а вещественные – как экземпляры типа `float`. Встроенная функция *Python* `type()` возвращает тип данных переменной. Следующий код выводит типы на экран:

```
print(type(a))
print(type(d))
```

В данном случае результатом будет

```
<class 'int'>
<class 'float'>
```

Тип данных определяет место для хранения переменной, расположение, в котором будет выделена память для переменной во время выполнения программы, способ представления значения переменной в памяти (и, соответственно, способ

преобразования из внешнего представления, т.е. записи в программе или во вводимых данных, во внутреннее), множество допустимых значений, а также набор операций, которые можно выполнять с этими данными. Так, значения переменных *целых типов* представляются в памяти как двоичные числа, полученные непосредственно переводом из десятичной системы счисления в двоичную. Целые числа в памяти представляются точно, и при выполнении операций с целыми числами никаких ошибок не возникает (кроме некоторых случаев деления).

Вещественные числа в памяти представляются следующим образом. Сначала число приводится к нормализованному виду, когда целая часть числа равна 0, первая цифра после запятой является значащей, а положение запятой в числе определяется значением показателя степени 10. Например, число 0,086 в нормализованной форме имеет вид $0.86 \cdot 10^{-1}$, число 123,45 – $0.12345 \cdot 10^3$. При этом цифры, расположенные в нормализованной записи после точки, называются *мантиссой*, а показатель степени 10 – это *порядок*. В памяти отдельно представляется мантисса и отдельно порядок. При этом количество бит, предназначенных для мантиссы, определяет точность представления, а количество бит, предназначенных для порядка, определяет диапазон представляемых чисел. Если количество цифр в двоичном представлении мантиссы превышает количество отведенных под нее разрядов, то последние двоичные цифры теряются и число оказывается представленным в памяти приближенно. Кроме того, при выполнении арифметических операций ошибки могут накапливаться. Таким образом, в общем случае вещественные числа в памяти представляются приближенно и их сравнение на точное равенство невозможно. Обычно вещественные переменные используются для обозначения величин, полученных в результате измерений, которые всегда имеют некоторую погрешность, либо полученных в результате вычислений.

В языках программирования обычно существуют две разновидности типов: *типы значений* и *ссылочные типы*. Переменные, основанные на типах значений, содержат непосредственно значения. Переменные ссылочных типов сохраняют ссылки (адреса в специально выделенной памяти) на фактиче-

ские данные. Таким образом, в переменной может храниться или значение, или объектная ссылка.

Значение – это двоичное представление данных, тип данных определяет способ представления значения переменной в памяти.

Объектная ссылка – адрес памяти. При создании объекта память для него выделяется специальной области памяти, переменная хранит только ссылку на расположение объекта.

В *Python* все переменные имеют ссылочный тип.

Существует несколько видов типов данных – встроенные и не встроенные. Встроенные – те типы, которые встроены в интерпретатор, не встроенные – типы данных, которые можно импортировать из других модулей. В программах настоящего учебника будем использовать переменные только встроенных типов: `int` (принимаящие целые значения), `double` (вещественные значения), `str` (строка символов), `bool` (принимаящие значения `True` (истина) или `False` (ложь)). В математическом представлении `True = 1`, `False = 0`), `complex` (комплексные числа).

В *Python* также есть тип `None type` – тип, представляющий отсутствие значения:

```
z = None
print(type(z))
Получим
<class 'NoneType'>
```

13.2. Константы (литералы)

Литералы используются в тексте программы для обозначения числовых значений, строк символов или логических констант. Другими словами, литерал представляет собой постоянное значение (константу), у которого нет имени.

Целочисленные литералы записываются в привычном виде: последовательность цифр со знаком `+` или `-` (знак `+` можно опустить). Например: `25`, `- 7`, `+553`.

Литералы с плавающей точкой могут быть записаны в виде целой и дробной частей, разделенных точкой (целая часть может отсутствовать, если она равна `0`; если дробная часть

равна 0, то после точки должен быть записан 0) или в экспоненциальной форме. В последнем случае число состоит из мантиссы (вещественной константы в десятичном представлении) и порядка – целого числа, перед которым располагается буква *e* или *E*. Например: $2.3E - 2$ обозначает число $2.3 \cdot 10^{-2}$ (или 0,023). Здесь 2.3 – мантисса, -2 – порядок. Например: 8.1, .23, 3.0, -5.3 , $1E - 1$, $5e4$.

Комплексные литералы записываются в виде суммы вещественной и мнимой частей: $4 + 2j$, $7 + j$. Если вещественная часть равна 0, ее можно опустить, например $2j$. Однако если мнимая часть равна 0 или 1, то ее опускать нельзя! Например: $7 + 0j$, $4 + 1j$.

Строковые литералы используются для представления текстовых строк. Это любая последовательность символов, заключенная в одинарные или двойные кавычки. Например, "X = ", 'Город Москва', "Результаты вычислений".

С помощью символа `\` (обратный слеш) можно указать некоторые специальные и управляющие символы. Последовательность `\` и следующего за ним символа называется *управляющей последовательностью*. Например, наличие в строке последовательности `\n` означает переход на новую строку, `\t` – горизонтальную табуляцию и т.п.

Логические литералы: True, False.

None – неопределенное значение переменной.

1.33. Математические операторы. Выражения

Выражения (в данном случае речь идет только о числовых выражениях) аналогичны привычным алгебраическим выражениям. Выражение может содержать символы арифметических операций: сложения $+$, вычитания $-$, умножения $*$, деления $/$ и вычисления остатка при целочисленном делении $\%$ и другие (полный список приведен в табл. П.1) и операнды, над которыми эти операции выполняются. Операндами могут быть литералы, переменные и обращения к функциям, результатом которых является одно значение. Если в выражении несколько операторов, то арифметические действия выполняются в соответствии со следующими приоритетами: оператор

возведения в степень (**), операторы умножения и деления (*, /, //, %), операторы сложения и вычитания (+, -). Операции одного приоритета выполняются по порядку слева направо. Дополнительный приоритет устанавливается при помощи скобок (как и в алгебраических выражениях). В частном случае выражение может состоять из одной константы или переменной. Если операндом какой-либо операции является обращение к методу (функции), то сначала выполняется это обращение, чтобы получить в качестве операнда числовое значение. Основные методы библиотеки `math` приведены в табл. 1.1.

Полный список приоритетов приведен в табл. П.8.

Таблица 1.1

Некоторые функции библиотеки `math`

Функция	Математическая запись
<code>math.pow(x, a)</code>	x^a
<code>math.sin(x)</code>	$\sin x$
<code>math.cos(x)</code>	$\cos x$
<code>math.tan(x)</code>	$\operatorname{tg} x$
<code>math.fabs(x)</code>	$ x $
<code>math.atan(x)</code>	$\operatorname{arctg} x$
<code>math.exp(x)</code>	e^x
<code>math.log(x)</code>	$\ln x$
<code>math.log10(x)</code>	$\lg x$
<code>math.sqrt(x)</code>	\sqrt{x}
<code>math.factorial(x)</code>	$x!$

Также можно использовать встроенные функции (табл. 1.2).

Таблица 1.2

Встроенные функции

Обозначение	Описание
<code>abs(a)</code>	Модуль числа a
<code>divmod(a,b)</code>	Получение пары чисел ($a // b, a \% b$)
<code>pow(a,b[,c])</code>	a в степени b . Если указано число c , тогда производится целочисленное деление на число c

К вещественным числам (`float`) также можно применить методы для приведения к целочисленному типу (табл. 1.3).

Таблица 1.3

Функции округления

Обозначение	Описание
<code>round(a)</code>	Округление до ближайшего целого
<code>int(a)</code>	Отсечение дробной части

Например:

```
round(16.76)
17
int(123.823)
123
```

Над целыми числами (`int`) можно производить битовые операции (`&`, `|`, `^`, `<<`, `>>`, `~`), список побитовых операторов приведен в табл. П.4.

Также целые числа можно переводить в другие системы счисления используя следующие методы:

- `bin(a)` – перевод числа в двоичную систему счисления;
- `hex(a)` – перевод числа в 16-тиричную систему счисления;
- `oct(a)` – перевод числа в 8-миричную системы счисления.

После перевода числа в другую систему счисления, перед ним обычно записывается символьное обозначение системы, в которой записано число:

- двоичная система обозначается символами `"0b"`,
- 16-тиричная – `"0x"`,
- 8-миричная – `"0o"`.

Например:

```
bin(3)
'0b11'
hex(123)
'0x7b'
oct(15)
'0o17'
```

1.3.4. Логические операторы. Выражения

Логические данные имеют тип `bool`.

К логическим данным могут применяться логические операторы. Результатом выполнения логического оператора яв-

ляется логическое значение (`True` или `False`). Логические операторы и результат их применения приведены в табл. П.5. Константы и переменные логического типа могут входить в состав *логического выражения*. Кроме этого, в логическом выражении могут использоваться в качестве операндов отношения. *Отношение* – это два арифметических выражения, соединенных знаком операции отношения `<`, `<=`, `>`, `>=`, `==` (равно), `!=` (не равно), `<>` (не равно).

Например, логическое выражение

```
(x <= 5.5) and (x >= -5.5)
```

будет иметь значение `True`, если выполняются оба условия, т.е. `x` принадлежит отрезку `[-5.5, 5.5]`, и `False` – в ином случае.

Заметим, что заключать отношения в скобки необязательно, так как они имеют более высокий приоритет, но в таком виде выражение имеет более ясный смысл.

В логическом выражении могут присутствовать операции трех типов: арифметические (в левой или правой частях отношения, выполняются в первую очередь), операции отношения и логические. Логические операции имеют самый низкий приоритет (см. табл. П.8). Из них в первую очередь выполняется операция отрицания (`!`), далее операция «логическое "И"» (`and`) и в последнюю очередь – «логическое "ИЛИ"» (`or`).

13.5. Основные операторы Python

Программа состоит из операторов.

К основным операторам отнесем операторы, которые позволяют описать типовые структуры алгоритмов:

оператор присваивания, при помощи которого происходит изменение значений переменных программы;

условный оператор реализует разветвление, т.е. переход на тот или другой блок (последовательность операторов) кода;

операторы цикла, реализующие повторяющиеся действия.

Перечисленные операторы являются управляющими операторами, так как они управляют последовательностью исполнения строк программы.

Рассмотрим более подробно каждый из перечисленных операторов.

Оператор присваивания имеет вид

Переменная = выражение

При выполнении оператора присваивания вычисляется значение выражения в правой части и присваивается переменной в левой части. При употреблении оператора присваивания необходимо следить за тем, чтобы к моменту его выполнения переменные, входящие в правую часть, т.е. в выражение, были определены (имели конкретные значения).

Заметим, что *Python* существует так называемое *множественное присваивание*, например:

```
a = b = c = 1
a, b, c = 1, 2, "привет!"
```

Условный оператор позволяет выбрать одну из двух ветвей вычислительного процесса. Общий вид оператора

```
if условие:
    операторы1
[else:
    операторы2]
```

Условие после *if* является логическим выражением, значением которого является одно из двух логических значений *True* или *False*. В первом случае выполняются *операторы1*, во втором – *операторы2*. Вторая ветвь (*операторы2*) может отсутствовать. Об этом говорит наличие квадратных скобок в определении оператора. (Здесь и далее то, что заключено в квадратные скобки, не является обязательным.) После выполнения какой-либо одной из ветвей условный оператор считается выполненным.

Например:

```
i = 15
if i <= 10:
    x = 5
else:
    x = 0
print(x)
```

После *if* или после *else* возможно наличие нескольких операторов, эти операторы образуют *составной оператор*, или *блок*. Например:

```
i = 5
if i <= 10:
    x = 5
    print(x)
else:
    x = 0
    print(x)
```

Каждый из двух блоков условного оператора (наличие обоих блоков необязательно) может содержать любые операторы, в том числе условные операторы. Если условный оператор входит в состав блока *операторы1*, выполняемого после *if*, то он записывается и выполняется по общим правилам. Если проверка условия осуществляется после *else*, то используется оператор *else if*, использование которого демонстрирует следующая схема:

```
if условие1:
    операторы1
elif условие2:
    операторы2
[else:
    операторы3]
```

Если истинно *условие1*, то выполняются *операторы1* и условный оператор заканчивает работу. Если *условие1* ложно, то проверяется *условие2*, и если оно истинно, то выполняются *операторы2*, в противном случае выполняются *операторы3*, если присутствует последний *else* или конструкция не выполняет никаких действий. Последний *else* всегда относится к последнему *if*, для которого еще не было соответствующего *else*. Например:

```
a = -4
if a > 5:
    x = 1
elif a > 0:
    x = 2
else:
    x = 3
print(x)
```


Заметим, что условный оператор может содержать несколько `elif`.

Конструкции `if-elif-else` можно использовать на разных уровнях вложенности для более сложных программ:

```
if условие1:
    if условие2_1:
        операторы2_1
    elif условие2_2:
        операторы2_2
    elif ...
    ...
...
else:
    ...
elif условие2:
    операторы2
elif ...
...
else:
    ...
```

Операторы цикла. В языке *Python* есть два основных способа организации цикла.

Оператор `while` реализует цикл по условию с проверкой условия до первого прохождения цикла (*цикл с предусловием*).

Общий вид цикла:

```
while условие:
    операторы
```

Операторы выполняются пока условие имеет значение `True`.

Например:

```
i = 1
z = 1
while z < 100:
    z = z * i
    i = i + 1
print(i)
```

На экран будет выведено число (6), факториал которого впервые превысит число 100.

Заметим, что оператор `while` никак не изменяет значения переменной цикла, поэтому такое изменение, если оно необходимо, должно быть организовано самим программистом.

Возможно принудительное прерывание выполнения цикла при помощи *оператора прерывания* `break`. Например:

```
v = 0
while v <= 10:
    v += 1
    if v > 7:
        break
    print(v)
```

Результат выполнения программы:

```
1
2
3
4
5
6
7
```

Здесь мы использовали оператор `break` для выхода из цикла до его окончания.

Оператор `continue` возобновляет выполнение цикла с первого оператора, игнорируя следующие за ним операторы:

```
while v <= 10:
    v += 1
    if v == 7:
        continue
    print(v)
```

Результат выполнения программы:

```
1
2
3
4
5
6
8
```

```

9
10
11

```

Здесь мы использовали оператор `continue`, чтобы пропустить следующий за ним оператор `print(v)`.

Проверить цикл на экстренный выход (`break`) можно с помощью `else`, например:

```

while var <= 10:
    var += 1
    if var == 17:
        break
    print(var)
else:
    print("цикл полностью выполнен")

```

Если экстренного выхода не было, т.е. оператор `break` не был выполнен, блок операторов, вложенный в `else`, выполняется.

Оператор `for` предназначен для выполнения одного оператора или группы (блока) операторов заданное количество раз. Общий вид цикла:

```

for переменная in последовательность:
    операторы1
[else:
    операторы2]

```

Цикл `for` менее универсальный, но работает быстрее, чем цикл `while`. Он способен проходить по любому итерируемому объекту, будь то списки, словари, кортежи, строки и не требует ручного увеличения счетчика итераций:

```

for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]:
    print(i)

```

Также для организации цикла можно использовать *генератор списков*. Для этих целей применим функцию `range()`, которая автоматически генерирует последовательность целых чисел в заданном диапазоне.

Функция `range()` генерирует арифметические прогрессии (объекты диапазонов). Общий вид:

```

range(нач._знач., кон._знач., [, шаг])
range([нач._знач.,] кон._знач.)

```

По умолчанию *нач._знач* = 0, *шаг* = 1. Например:

```
for i in range(5):  
    print(i)
```

приведет к результату

```
0  
1  
2  
3  
4
```

Заметим, что заданная конечная точка никогда не входит в генерируемую последовательность; `range(5)` генерирует 5 значений, индексируемых как обычная последовательность длиной 5. Возможно установить другое число в качестве начала диапазона или указать другое приращение (даже отрицательное), такое приращение называют «шагом»), например:

```
range(5, 10) генерирует числа 5, 6, 7, 8, 9;  
range(0, 10, 3) генерирует числа 0, 3, 6, 9;  
range(-10, -100, -30) генерирует числа -10, -40, -70.
```

С помощью `range()` также можно осуществлять перебор списка по индексам (подробнее это будет рассмотрено в гл. 4).

13.6. Ввод-вывод данных

Ввод данных с клавиатуры осуществляется с помощью функции `input()`. Функция считывает клавиши, вводимые пользователем на клавиатуре, и возвращает их в виде строки

```
v = input()
```

Функция `input()` всегда возвращает вводимые значения как строки (`str`), даже если введенное значение является числом, оно возвращается из функции `input()` как строка. Например:

```
x = input()  
print(type(x))
```

Если выполнить этот код и ввести значение «5», на экране отобразится `<class 'str'>`, несмотря на то, что введено числовое значение. Чтобы преобразовать значение в целочисленную переменную, можно воспользоваться функцией `int()`:

```
x = int(input())  
print(type(x))
```

Этот код выводит `<class 'int'>` для значения 5. Если в значении может быть дробная часть, можно использовать функцию `float` аналогичным образом:

```
x = float(input())
```

Так же осуществляется преобразование и к остальным числовым типам.

Обратите внимание на то, что если в функцию `int()` будет передано значение, не являющееся целым числом (например, 5.0), это приведет к ошибке и, следовательно, прекращению работы программы. Убедитесь в этом самостоятельно.

Функция `input()` может также вызываться с параметром, например:

```
v = int(input(«введите целое число»))
```

При этом строковый литерал (заключенный в одинарные или двойные кавычки) будет выведен на экран без изменений. Тогда ввод числа 5 будет выглядеть так:

```
введите целое число5
```

Обратите внимание, что выполнение программы

```
print("введите целое число")
```

```
v = int(input())
```

приведет к *почти* такому же результату. Разница в том, что функция `print()` (по умолчанию) добавляет в выходные данные символ новой строки и ввод числа 5 будет выглядеть так:

```
введите целое число
```

```
5
```

Вывод данных осуществляется с использованием оператора `print()`.

Общий вид оператора:

```
print(список вывода)
```

Например:

```
P = 5
```

```
auto = «BMW»
```

```
print ("Этот автомобиль -", auto, P, "-й серии")
```

```
Этот автомобиль - BMW 5 -й серии
```

Простейшие варианты вывода значений отдельных переменных использованы в приведенных выше программах.

Вывод может быть организован с использованием формата. В этом случае оператор вывода имеет следующий вид:

```
print(["строка формата"]% (список вывода))
```

В *списке вывода* перечисляются через запятую элементы списка. В качестве элементов списка вывода в общем случае могут фигурировать имена переменных, литералы или выражения, которые перед выводом будут вычислены.

В *строке формата* для каждого выводимого значения указывается код форматирования (основные коды форматирования приведены в табл. 1.4).

Например:

```
g = "зеленые"
```

```
print("У меня висят %s занавески" % g)
```

Будет напечатано:

У меня висят зеленые занавески

```
bus = 45
```

```
metro = 50
```

```
print("если я еду на работу на автобусе, а затем  
на метро, стоимость поездки составляет %d рублей" %  
(bus+metro))
```

Будет напечатано (в одну строку):

если я еду на работу на автобусе, а затем на метро,
стоимость поездки составляет 95 рублей

Если же мы хотим вывести текст в несколько последовательных коротких строк, то для перехода на новую строку можно использовать управляющую последовательность `\n`, например:

```
bus = 45
```

```
metro = 52
```

```
print("если я еду на работу на автобусе,\n а затем  
на метро, \n стоимость поездки составляет %d рублей" %  
(bus+metro))
```

Будет напечатано:

если я еду на работу на автобусе,
а затем на метро,
стоимость поездки составляет 95 рублей

Наличие в строке символа `\t` аналогично нажатию клавиши [Tab] (табулирование):

```
trans1 = "автобус"
trans2 = "трамвай"
bus = 45
tram = 40
print("%s \t %i руб. \n%s \t %d руб. " %
(trans1,bus,trans2,tram))
```

В результате будет напечатано:

```
автобус      45 руб.
трамвай      40 руб.
```

Таблица 1.4

Основные коды форматирования

Код	Отображаемый тип данных
<code>%d, %i, %u</code>	Целое число
<code>%s</code>	Строка текста
<code>%f, %F</code>	Вещественное число
<code>%r</code>	Литерал <i>Python</i>

Для многократного вывода одного и того же можно использовать символ умножения (*):

```
print ("-" * 10)
```

Получим

```
- - - - -
```

```
a = 25
```

```
b = 36
```

```
c = 38
```

```
d = 49
```

```
e = "Длина Удава из известного мультфильма составляет: "
```

```
f = "или"
```

```
g = "попугаев."
```

```
h = "Какой ответ правильный?"
```

```
k = "Правильный ответ: «
```

```
m = k + "%d " % c
```

```
print ("%s %d, %d, %d %s %d %s" % (e, a, b, c, f, d, g))
```

```
print (h)
```

```
print ("-" * 30)
```

```
print (m + g)
Длина Удава из известного мультфильма составляет:
25, 36, 38 или 49 попугаев.
Какой ответ правильный?
- - - - -
Правильный ответ: 38 попугаев
```

Однако в настоящее время на смену синтаксису % приходит метод `str.format`. (Отметим, что это не бинарный оператор над строками, а метод объекта *строка*, принимающий ряд параметров; подробнее о методах и объектах будет рассказано в следующих главах.) Общий вид:

```
print ("строка формата".format(список вывода))
```

Например, вместо

```
first = 'Иван'
```

```
last = 'Иванов'
```

```
print("Меня зовут %s %s" % (first, last))
```

```
Меня зовут Иван Иванов
```

можно использовать

```
print("Меня зовут {} {}".format(first, last))
```

Преимуществами использования `format` является, например, возможность использования параметров в произвольном порядке:

```
print("Меня зовут {1} {0}".format(first, last))
```

```
Меня зовут Иванов Иван
```

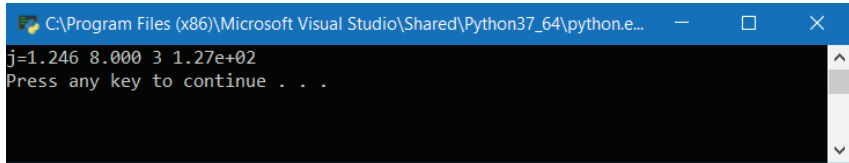
Также появляется возможность использовать значение одного элемента списка вывода несколько раз.

При использовании `format` элементы списка вывода автоматически индексируются (нумерация начинается с 0), и в строке формата каждому выводимому элементу списка соответствует пара фигурных скобок {}, в которых можно указать как номер элемента в списке, так и формат его представления при выводе. По умолчанию (если номер не указан), элементы выводятся в заданном порядке. Например:

```
x = 127.356345
```

```
y = 1.24567
```

```
print("j={2:.4} {1:.3f} {0} {3:.2e}".format(3, 8, y, x))
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.e...
j=1.246 8.000 3 1.27e+02
Press any key to continue . . .
```

Здесь выведенная строка соответствует использованному формату: выведено `j=`, далее второй (предпоследний) элемент списка по формату для вещественных чисел в поле размером 4 (т.е. с тремя цифрами после точки), затем через пробел первый элемент списка по формату для вещественных чисел с тремя цифрами после точки, следующим через пробел выводится начальный элемент списка без указания форматного кода и наконец последний (третий) элемент списка по формату с порядком с двумя цифрами после точки в представлении мантиссы.

Рассмотрим еще несколько примеров.

Вывод значения в заданное количество позиций (обратите внимание, что при выводе вещественных чисел происходит преобразование в соответствии с правилами округления!):

```
x = 77.658904
print('{0:.3}'.format(x))
77.7
```

Здесь число выводится в вещественном виде в поле размером 3 (3 позиции без учета десятичной точки).

Однако можно задать размер поля для вывода:

```
x = 17.68
s = 'symbol'
print('{0:10}'.format(x))
print('{0:10}'.format(s))
17.68
symbol
```

Обратите внимание, что по умолчанию символные переменные при выводе выравниваются по левой границе поля (оставшиеся позиции заполняются пробелами), а числовые – по правой. Если нужно задать выравнивание по нужной границе поля, то используются знаки `<`, `>` и `^` для выравнивания по левой границе, по правой границе и по центру поля соответственно:

```
print('{0:<10}'.format(x))
17.68
print('{0:>10}'.format(s))
symbol
```

Вот, например, выравнивание текста по ширине (^) поля размером 13:

```
print('{0:*^13}'.format('Таблица'))
***Таблица***
```

Также можно использовать вывод по ключевым словам (с использованием именованных аргументов):

```
print("The price is ${number}.".format(number=35))
The price is $35
print('Меня зовут {name}. Мой любимый предмет - {dis}'.format(name = 'Лена', dis = 'информатика'))
Меня зовут Лена. Мой любимый предмет - информатика.
```

Именованные параметры можно также развернуть из словаря (подробнее о словарях будет рассказано далее), используя оператор **:

```
person = {'first':'Иван', 'last':'Иванов'}
print("Меня зовут {first} {last}".format(**person))
Меня зовут Иван Иванов
```

А также допустимо совместное использование именованных и позиционных аргументов:

```
person = {'first':'Иван', 'last':'Иванов'}
print("Меня {0} {first} {last}".format('зовут', **person))
Меня зовут Иван Иванов
```

В заключение заметим, что хотя внутри {} нельзя использовать исполняемый *Python*-код, вместо этого существует отдельный очень простой микроязык (отличающийся от *Python* в целом). Но есть и небольшие исключения: например, можно получить значения атрибутов / свойств через точку (.), значение итерируемого объекта (списка, строки) по индексу, используя [], и некоторые другие.

Детально все обозначения форматов для `str.format` описаны в Предложении по расширению *Python* PEP 3101 (Расширенное форматирование строк).

1.3.7. Построение программного кода

Программа состоит из операторов. Обычно каждый оператор (логическая строка) располагается на одной строке экрана (физическая строка). Однако можно расположить в одной строке несколько операторов, разделяя их символом точка с запятой (;), например:

```
v = 1  
print(v)
```

то же, что и

```
v = 1; print(v)
```

Явное объединение строк. В случае очень длинных строк можно осуществлять перенос оператора на следующую строку, используя в конце строки обратный слэш (\) как символ переноса. Это называется явным объединением строк:

```
v =\  
1; print(v)
```

Неявное объединение строк. В отдельных случаях, когда, например, в строке есть открывающая круглая, квадратная или фигурная скобка, но нет закрывающей, использование обратного слэша не обязательно. Это называется неявным объединением строк:

```
v = 1; print(  
v)
```

Отступы. В *Python* отступы в начале строки очень важны. Они используются для группировки операторов. Это означает, что операторы, идущие вместе, должны иметь одинаковый отступ. Каждый такой набор операторов называется блоком (составным оператором). Неправильные отступы могут приводить к возникновению ошибок! Например:

```
v = 1  
print(v)
```

приведет к ошибке, так как в начале второй строки есть один пробел.

Не смешивайте пробелы и символы табуляции в отступах, поскольку не на всех платформах это работает корректно. Обычно рекомендуется использовать одиночную табуляцию или четыре

пробела для каждого уровня отступа. Выберите один стиль для отступов и используйте его постоянно!

Комментарии. Текст, расположенный в строке после символа `#`, является комментарием и при выполнении программы игнорируется (т.е. не влияет на выполнение программы). Комментарий может располагаться и в конце строки, достаточно перед ним разместить `#`.

```
v = 1 #начальное присваивание
#печать результата
print(v)
```

Более подробно правила написания кода описаны в *Python PEP 8* (Руководство по написанию кода на *Python*).

1.4. Этапы решения задач

Решение практически любой задачи с использованием компьютера предполагает выполнение следующих этапов.

Первым этапом является четкая формулировка задачи (обычно на профессиональном языке), выделение исходных данных для ее решения и точные указания относительно того, какие результаты и в каком виде должны быть получены.

Второй этап – формальная (математическая) постановка задачи, т.е. представление ее в виде уравнений, соотношений, ограничений и т.п. При решении инженерных задач этот этап, как правило, является обязательным. (Некоторые задачи, например задачи обработки текстов, не требуют математической постановки.)

Третий этап – выбор метода решения. Выбор метода определяется решаемой задачей. Могут использоваться и другие соображения: наличие готовых программ, возможность получения необходимой точности решения, сочетание с другими методами, используемыми при решении данной задачи и т.д. Выполнение этого этапа требует некоторого кругозора как в области программирования, так и в области используемых методов.

Четвертый этап – разработка алгоритма на основе выбранного метода (методов). При выборе алгоритма желательно рассмотреть и проанализировать несколько вариантов, прежде чем сделать окончательный выбор. Следует обратить внима-

ние на тесную взаимосвязь третьего и четвертого этапов, так как алгоритм в большой степени определяется выбранным методом, хотя один и тот же метод в свою очередь может быть реализован различными способами.

При разработке алгоритма решения сложной задачи следует использовать метод пошаговой детализации, используя при этом только типовые структуры алгоритма. Следует максимально использовать существующие типовые или разработанные ранее алгоритмы для отдельных подзадач.

Последовательные этапы разработки алгоритма можно фиксировать, используя конструкции современных алгоритмических языков, реализующие типовые структуры алгоритма. Полностью закончив детализацию, мы получим готовую к выполнению программу.

Пятый этап – выбор структуры данных. От выбора способа представления данных зависит и алгоритм их обработки. Поэтому четвертый и пятый этапы взаимосвязаны. Нужно выбирать структуру данных наиболее естественную для решаемой задачи, использовать массивы для представления данных, когда это наиболее очевидный способ их организации, и пр.

Шестой этап – собственно программирование, т.е. запись разработанного алгоритма на языке программирования. Если разработка алгоритма выполнена хорошо, то программирование принципиальных трудностей не вызывает, однако при разработке программы целесообразно придерживаться некоторых правил, использование которых облегчит ее отладку и дальнейшее использование.

1. Программа должна быть универсальной, т.е. не зависящей от конкретного набора данных. Если данные представлены в виде массивов, то в качестве исходных данных следует рассматривать не только сами данные, но и их количество, для того чтобы одна и та же программа могла быть использована для обработки массивов различных размеров.

Универсальная программа должна обрабатывать вырожденные случаи (например, число элементов вектора равно 0 или 1) и печатать сообщение об ошибке, если размер массива превысил допустимое значение (использованное при объявлении массива).

2. Вместо числовых литералов (констант) лучше использовать переменные. Если в программе используются константы, то при их изменении нужно изменять в исходной программе каждый оператор, содержащий прежнюю константу. Эта процедура отнимает много времени и часто вызывает ошибки. В программе следует предусмотреть контроль вводимых данных (в частности, программа не должна выполняться, если данные выходят за пределы допустимого диапазона).

3. Некоторые простые приемы позволяют повысить эффективность программы (т.е. уменьшить количество выполняемых операций и время работы программы). К таким приемам относятся:

- использование операции умножения вместо возведения в степень для низких степеней;
- арифметическое выражение, которое несколько раз вычисляется в программе с одними и теми же данными, лучше вычислить один раз и присвоить его значение переменной, которую и использовать везде вместо арифметического выражения;
- при организации циклов в качестве границ индексов использовать переменные, а не выражения, которые вычислялись бы при каждом прохождении цикла;
- особое внимание обратить на организацию циклов, убрав из них все повторяющиеся с одинаковыми данными вычисления и выполняя их до входа в цикл.

4. Программа должна содержать комментарии, позволяющие легко проследить за логической взаимосвязью и функциями отдельных ее частей.

При написании программы следует заботиться о ее структуре так, чтобы программа была удобочитаемой. В частности, в программе должны быть хорошо видны циклы и другие конструкции. Для этого операторы, как правило, размещают в отдельных строках. Если весь цикл размещается в одной строке, то она не должна содержать других операторов.

Седьмой этап – отладка и тестирование программы – это проверка правильности работы программы и исправление обнаруженных ошибок.

Для выполнения этого этапа необходимо подготовить тесты. *Тест* – это специально подобранные исходные данные в сово-

купности с теми результатами, которые должна выдавать программа при обработке этих данных. Разработка тестов – трудоемкая работа, часто требующая выполнения ручных просчетов. При подготовке тестов нужно стремиться обеспечить проверку всех ветвей программы.

Далее даются некоторые рекомендации по тестированию и отладке.

1. Проверьте работу программы (или отдельных ее частей) вручную. Это особенно полезно для начинающих.

2. Проводите тестирование и отладку отдельно для логически самостоятельных частей программы.

3. Используйте отладочную печать в наиболее ответственных частях программы. Операторы вывода при этом располагайте в отдельных строках так, чтобы можно было легко убрать их из программы после окончания отладки.

4. При тестировании, если это возможно, используйте меньшие объемы данных, чем те, на которые рассчитана программа.

Восьмой этап – счет по готовой программе и анализ результатов. Этот этап является итогом выполнения всех предыдущих этапов и служит подтверждением (или опровержением) их правомерности. После этого этапа возможно потребуется пересмотр самого подхода к решению задачи и возврат к первому этапу для повторного выполнения всех этапов с учетом приобретенного опыта.

Девятый этап – составление документации. Формальные требования к документации программ описаны в ЕСПД (Единая система программной документации – это серия ГОСТов, описывающих правила по составлению документации к программам).

1.5. Среда разработки

Интегрированная среда разработки (IDE). Язык *Python* является языком высокого уровня. Это означает, что программа пишется на языке, приближенном к естественному человеческому языку, а затем автоматически переводится на язык машинных кодов. Перевод с языка высокого уровня на машинный язык выполняет компилятор. В результате получается файл, который можно сохранить на диске.

В программе, как правило, используются различные компоненты из библиотек. Большие программы состоят из нескольких исходных файлов, и каждый такой файл компилируется в отдельный объектный модуль. Объединить все это в одну программу может компоновщик. Результат работы компоновщика – это исполняемая программа. Такой файл уже может выполняться автономно, без участия исходной программы и компилятора.

В принципе, можно использовать обычный текстовый редактор для ввода кода программы, а затем – отдельные программы компилятора и компоновщика, вызвав их из командной строки операционной системы. Но гораздо удобнее использовать специальное приложение, в котором объединены все инструменты для разработки и отладки программ.

Интегрированная среда разработки (IDE – Integrated Development Environment) – это специальное приложение, которое позволяет упростить разработку программ на языке высокого уровня.

Она обычно включает:

- текстовый редактор – чтобы набрать текст программы, сохранить его на диске, редактировать. Такой редактор обычно еще и выделяет элементы кода разными цветами, подсказывает возможные ошибки синтаксиса, а иногда и создает фрагменты кода автоматически;
- компилятор (или интерпретатор) и компоновщик – чтобы перевести программу в машинный код;
- средства отладки и запуска программ – чтобы обеспечить удобство поиска ошибок;
- стандартные библиотеки, содержащие многократно используемые элементы программ;
- справочную систему и др.

Для языка *Python* существует достаточно много различных сред разработки.

IDLE – редактор, поставляемый вместе с *Python*. Это базовый, упрощенный режим программирования на *Python*. Это хороший редактор для начала программирования и понимания основ языка. **IDLE** предоставляет возможности для автозавершения кода, подсветку синтаксиса, подбор отступа и ба-

зовый встроенный отладчик. Этот редактор легкий в освоении и подходит для начинающих, но так как в нем не хватает продвинутых функций, то он не подходит для реализации сложных проектов.

Sublime Text – редактор, который работает с несколькими языками программирования. Он прост в использовании, компактен и эффективен, тонко настраивается и дополняется пакетами для отладки, автозавершения кода, линтинга.

Visual Studio Code – бесплатный редактор кода от *Microsoft* для *Windows*, *Linux* и *MacOS*. Его возможности – отладка, подсветка синтаксиса, интеллектуальное завершение кода, предопределенные фрагменты кода, рефакторинг и интеграция с *Git*. Поддерживаются различные языки программирования. Для начала работы с *Python* может понадобиться несколько дополнительных пакетов, но установить их довольно просто. Редактор постоянно обновляется.

Jupyter Notebook – это веб-приложение с открытым исходным кодом, позволяющее создавать документы с выполняемым интерактивно кодом, уравнениями, визуализациями, простым текстом. Он используется для очистки и преобразования данных, численного и статистического моделирования, визуализации данных, машинного обучения и многого другого. Этот редактор – хороший вариант для начала работы с наукой о данных и машинным обучением. Файлами можно поделиться с кем угодно, они помогают эффективнее работать с кодом. В *Jupyter Notebook* можно работать с каждым блоком кода отдельно, есть возможность использовать разметку.

PyCharm – это интегрированная среда разработки специально для *Python*, она имеет широкий набор возможностей, таких как автозавершение и инспекция кода, подсветка ошибок, исправления, отладка, система контроля версий и рефакторинг. *IDE* доступна на *Microsoft Windows*, *Linux* и *MacOS*.

Spyder – это мощная научная интегрированная среда программирования, написанная на *Python* для *Python*. Она разработана учеными, инженерами и аналитиками данных для них самих. *Spyder* обладает уникальным сочетанием возможностей. Продвинутое редактирование, анализ, отладка и профилирование сочетаются с возможностями исследования дан-

ных, интерактивного выполнения, глубокой инспекции кода и красивой визуализацией.

Notepad++ – это редактор текста и исходного кода, работающий на *Microsoft Windows*. Поддерживается редактирование с вкладками, что позволяет работать с несколькими открытыми файлами в одном окне. Название проекта происходит от оператора инкремента языка *C*. *Notepad++* распространяется как свободное программное обеспечение. Редактор поддерживает множество языков программирования и может быть полезным. Для того чтобы сделать редактор функциональным для программирования на *Python*, нужно установить дополнительные пакеты.

Рассмотрим более подробно работу с *Python* в среде *Visual Studio*.

1.5.1. Создание нового проекта

1. Создать новый проект *Visual Studio Python* (рис. 1.6). (Возможны некоторые отличия в зависимости от установленной версии *Visual Studio*.) Запустить *Visual Studio* и выбрать команду «Создание проекта» (рис. 1.7).

2. В рамках этого проекта выбрать «Приложение *Python*»:

Начало работы

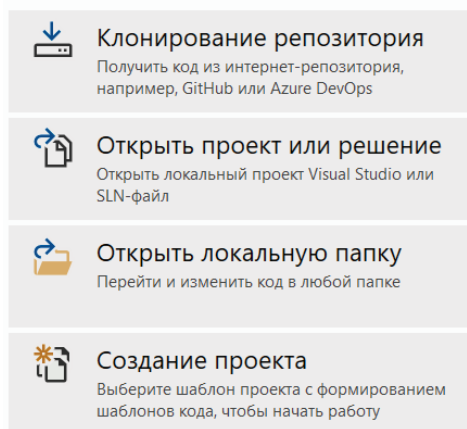


Рис. 1.6. Запуск *Visual Studio*

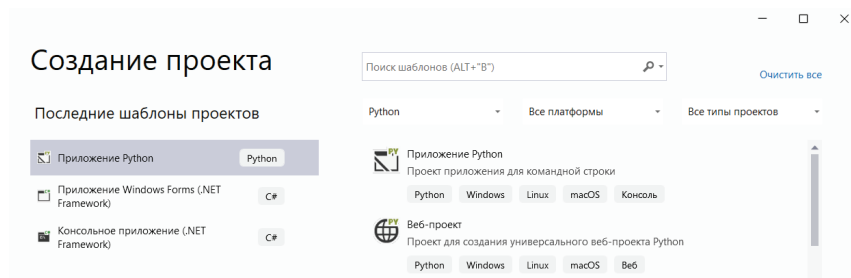


Рис. 1.7. Создание проекта

3. Настроить новый проект (рис. 1.8). Здесь можно задать имя проекта (или принять имя по умолчанию).

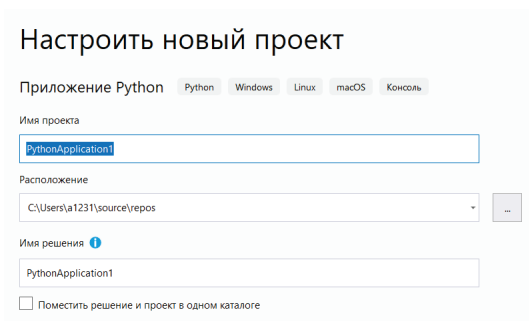


Рис. 1.8. Настройка проекта

4. В открывшемся интерактивном окне *Python* (рис. 1.9) набрать программный код.

1.5.2. Сохранение проекта

Для сохранения проекта в меню «Файл» выберите команду «Сохранить».

1.5.3. Открытие существующего проекта

В меню «Файл» выберите команду «Открыть» / «Решение или проект». В открывшемся окне выбрать `PythonApplication1` (или другое имя, которое вы выбрали для приложения на этапе создания) и нажать кнопку «Открыть» (рис. 1.10).

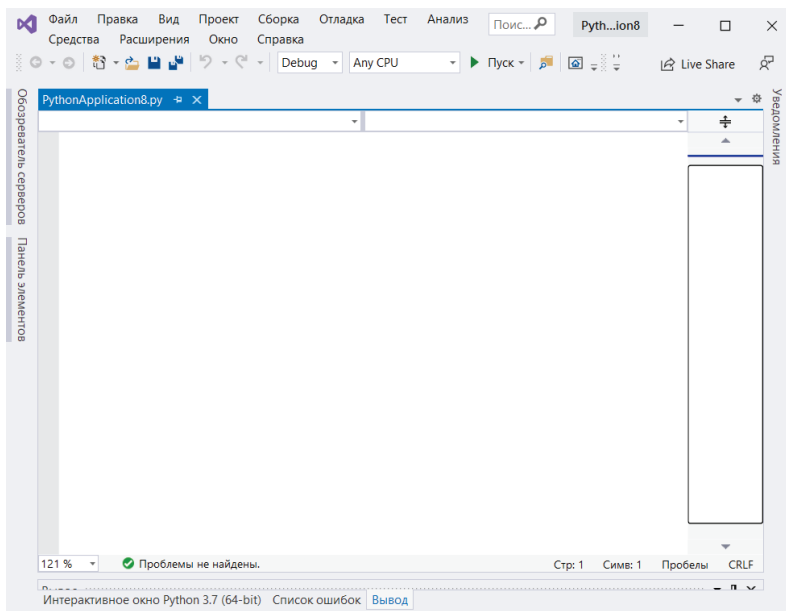
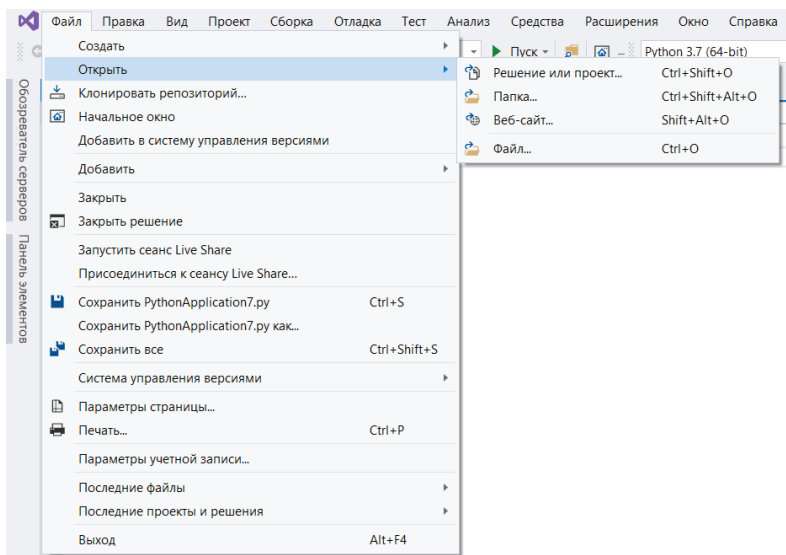
Рис. 1.9. Интерактивное окно *Python*

Рис. 1.10. Открытие проекта

ГЛАВА 2. Разработка программ циклической структуры

Циклом называется многократно повторяющаяся последовательность действий (операторов). Цикл – типовая структура алгоритма, характерная для компьютерных программ.

2.1. Циклы по счетчику

Рассмотрим вначале задачи, в которых количество повторений цикла определяется до начала его выполнения.

При решении таких задач необходимо следующее.

1. Выделить повторяющиеся действия и записать их в общем виде. (Обращайте внимание на порядок выполнения действий! От него зависят начальные присваивания используемых переменных, а также параметры цикла!)
2. Выбрать вид цикла.
3. Определить параметры цикла.

Пример 2.1. Вычислить сумму $s = 1 + 2 + 3 + \dots + 10$.

Организуем вычисления так, чтобы на каждом шаге выполнялись простые действия, в данном случае это прибавление к сумме очередного слагаемого. Вначале необходимо обнулить переменную, в которой будет накапливаться сумма ($s = 0$), а затем на каждом шаге добавлять к сумме очередной член суммы, т.е. многократно выполнять операцию

$$s = s + i,$$

где $i = 1, 2, 3, \dots, 10$.

Таким образом, в качестве управляющей переменной цикла в данном случае можно использовать сам член суммы, который изменяется в заданных пределах (от 1 до 10) с заданным шагом 1. Тогда программа будет иметь следующий вид.

Вариант 1а

```
i = 1  
S = 0  
while i <= 10:
```

```
S = S + i
i = i + 1
print(S)
55
```

Вариант 1б

```
i = 1
S = 0
while i <= 10:
    i = i + 1
    S = S + i
print(S, '\nОтвет неверный!')
65
Ответ неверный!
```

Обратите внимание, что изменив порядок действий внутри цикла, мы получаем неверное решение! Измените начальные присваивания используемых переменных и (или) параметры цикла для получения верного ответа самостоятельно.

Также можно создать соответствующую последовательность, используя функцию `range()`.

Вариант 2

```
S = 0
for i in range(11):
    S = S + i
print(S)
55
```

Пример 2.2. Вычислить сумму четных чисел от 2 до 20.

Вариант 1

```
i = 2
S = 0
while i <= 20:
    S = S + i
    i = i + 2
print(S)
110
```

Вариант 2

```
S = 0
for i in range(21,2):
```

```
S = S + i
print(S)
110
```

Пример 2.3. Вычислить $s = 3 + 3^2 + 3^3 + \dots + 3^8$.

На каждом шаге алгоритма необходимо прибавлять к сумме очередное слагаемое

$$(s = s + 3^i),$$

где $i = 1, 2, 3, \dots, 8$.

Вариант 1

```
i = 1
S = 0
while i <= 8:
    S = S + 3**i
    i = i + 1
print(S)
9840
```

Вариант 2

```
S = 0
for i in range(1, 9):
    S = S + 3**i
print(S)
9840
```

Хотя операция возведения в степень в языке *Python* присутствует, ее использование при решении данной задачи нецелесообразно. Рассмотрим вариант решения, в котором следующий член суммы будем получать из предыдущего путем домножения его на 3. В качестве управляющей переменной цикла можно использовать показатель степени, изменяющийся в заданных пределах от 1 до 8 с шагом, равным 1. (Обратите внимание, что переменная i в этом случае не участвует непосредственно в вычислениях, а используется только для подсчета количества повторений цикла.) Программа, таким образом, будет иметь следующий вид:

```
i = 1
S = 0
```

```
a = 1
while i <= 8:
    a = a * 3
    S = S + a
    i = i + 1
print(S)
```

или

```
S = 0
a = 1
for i in range(8):
    a *= 3
    S = S + a
print(S)
```

Замечание. Выражение для получения очередного члена последовательности из предыдущего называется *рекуррентной формулой*.

Использование рекуррентной формулы при решении задач всегда предпочтительнее, так как существенно сокращает время выполнения программы.

Пример 2.4. Вычислить $s = 5/8 + 7/10 + \dots + 31/34$.

Для организации цикла в этом случае можно использовать числитель, изменяющийся от 5 до 31 с шагом 2, а знаменатель выразить через числитель (он отличается от числителя на 3):

```
S = 0
for i in range(5, 32, 2):
    S = S + i / (i + 3)
print("{0:.5}".format(S))
11.591
```

Возможны и другие способы организации вычислений. Предложите их самостоятельно.

Для организации цикла в данном примере можно использовать специальную переменную (счетчик), определяющую номер итерации. Для определения общего количества повторений цикла (n) можно использовать формулу

$$n = (i_{\text{кон}} - i_{\text{нач}}) / h + 1.$$

В данном случае при $i_{\text{кон}} = 31$, $i_{\text{нач}} = 5$, $h = 2$ получаем $n = 14$. Тогда программа будет иметь вид

```
S = 0
i1 = 5
i2 = 31
h = 2
n = int((i2 - i1) / h + 1)
a = i1
for i in range(n):
    S = S + a / (a + 3)
    a += h
print("{0:.5}".format(S))
```

Здесь при вычислении n использовано преобразование типа (указанием типа перед выражением), так как `range()` принимает только целые аргументы.

Пример 2.5. Вычислить $s = \frac{2}{3} \sum_{i=1}^{25} (-1)^i \frac{2i}{i^2 + 2}$.

Вычисление каждого члена суммы целесообразно осуществлять в два приема. Сомножитель $(-1)^i$ (обозначим его через p) может вычисляться рекуррентно. Для следующего члена необходимо вычислять $p = -p$ и далее очередной член суммы получать умножением его абсолютной величины на p . Программа таким образом будет иметь вид

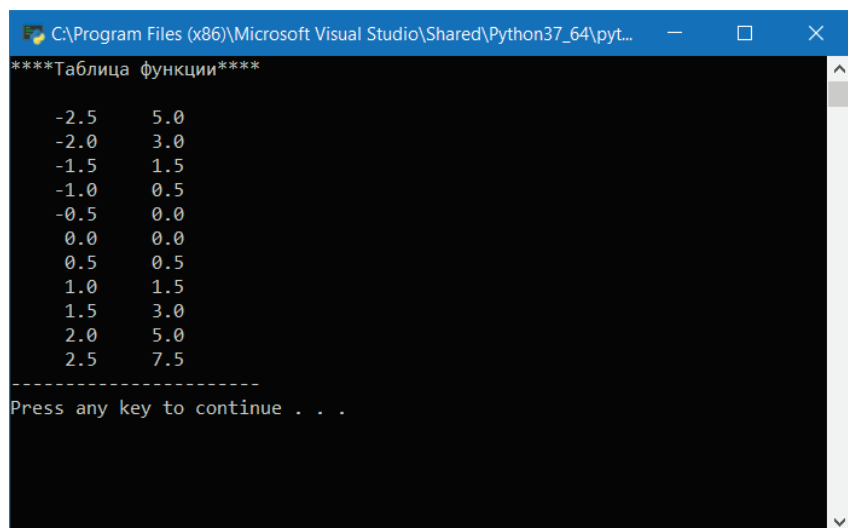
```
S = 0
p = 1
for i in range(1, 26):
    p = -p
    a = p * 2 * i / (i * i + 2)
    S = S + a
S = 2 / 3 * S
print("{0:.4}".format(S))
-0.2273
```

Пример 2.6. Получить таблицу значений функции $y = x^2 + 0,5x$ при изменении x в пределах от $xh = -2,5$ до $xk = 2,5$ с шагом $h = 0,5$.

Вычислим количество точек на отрезке и организуем цикл по номеру точки.

Вариант 1

```
xh = -2.5
xk = 2.5
h = 0.5
n = int((xk - xh) / h + 1)
x = xh
print('{0:*^23}\n'.format('Таблица функции'))
for i in range(1,n+1):
    y = x * x + 0.5 * x
    print("{0:8}{1:8}".format(x,y))
    x += h
print('-'*23)
```



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\pyt...
***Таблица функции***
-2.5    5.0
-2.0    3.0
-1.5    1.5
-1.0    0.5
-0.5    0.0
 0.0    0.0
 0.5    0.5
 1.0    1.5
 1.5    3.0
 2.0    5.0
 2.5    7.5
-----
Press any key to continue . . .
```

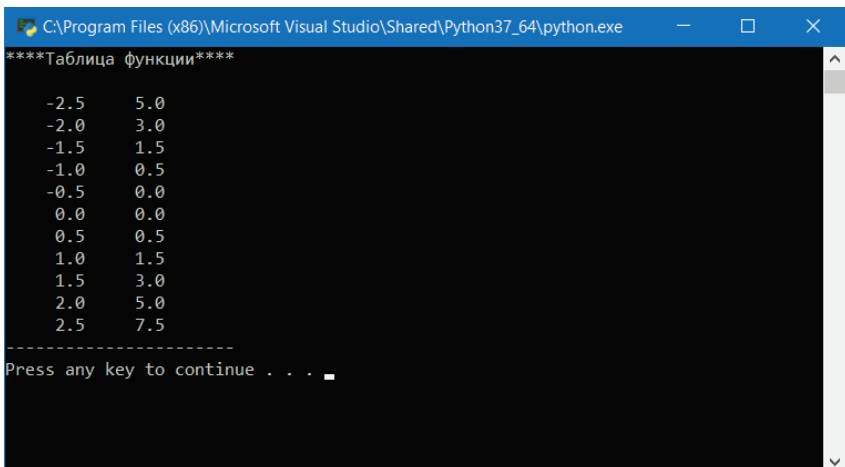
Обратите внимание на то, что вывод значений аргумента и функции осуществляется внутри цикла, так как после выхода из цикла все значения (кроме последнего) будут потеряны.

Замечание. Для определения количества значений аргумента при его изменении в пределах от xh до xk с шагом h используется формула $n = (xk - xh) / h + 1$. В программе для получения в качестве n целого значения использовано преобразование типа.

Рассмотрим второй вариант программы, когда в качестве управляющей переменной цикла используется аргумент x . Однако при этом следует иметь в виду, что x является вещественной величиной, поэтому при многократном выполнении операции $x = x + h$ может накапливаться ошибка вычислений и при сравнении $x \leq xk$ (для выхода из цикла) может оказаться, что x формально (на небольшую величину) больше xk . Тогда последняя точка не будет вычислена. Чтобы этого не произошло, необходимо сравнивать x с величиной, несколько превосходящей xk , например $xk + 0,0001$. При этом программа будет иметь следующий вид.

Вариант 2

```
xh = -2.5
xk = 2.5
h = 0.5
n = int((xk - xh) / h + 1)
x = xh
print('{0:*^23}\n'.format('Таблица функции'))
while x <= xk + 0.0001:
    y = x * x + 0.5 * x
    print("{0:8}{1:8}".format(x,y))
    x += h
print('-'*23)
```



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
***Таблица функции***
-2.5    5.0
-2.0    3.0
-1.5    1.5
-1.0    0.5
-0.5    0.0
0.0     0.0
0.5     0.5
1.0     1.5
1.5     3.0
2.0     5.0
2.5     7.5
-----
Press any key to continue . . .
```

Пример 2.7. Вычислить $p = n!$ при $n = 8$.

Вычисление факториала можно организовать в цикле, домножая каждый раз значение p на очередной сомножитель:

```
i = 1
p = 1
while i <= 8:
    p = p * i
    i = i + 1
print(p)

p = 1
for i in range(8):
    p = p * (i + 1)
print(p)
40320
```

Замечание. Обратите внимание, что начальное значение p равно 1, а не 0, так как в противном случае результат умножения не изменялся бы, а оставался равным 0.

В заключение отметим, что для решения задач, в которых количество повторений цикла определяется до начала его выполнения, как правило, можно использовать как цикл `for`, так и цикл `while`. При этом предпочтение отдается той конструкции, которая в большей степени способствует наглядности, ясности и естественности («читаемости») программного кода. Вместе с тем нужно принимать во внимание, что цикл `for` работает быстрее, но при этом `range()` может принимать только целые аргументы.

2.2. Циклы по условию

Циклы по условию используются тогда, когда количество повторений цикла неизвестно и в ряде случаев является искомой величиной при решении задачи.

Пример 2.8. Определить количество (n) членов арифметической прогрессии

$$s = a + (a + h) + (a + 2h) + \dots + (a + nh),$$

сумма которых впервые превысит заданное число p .

На каждом шаге алгоритма нужно добавлять к сумме s очередной член $m = a + ih$ ($s = s + m$), $i = 1, 2, \dots$, но при этом перед каждым прибавлением очередного члена проверять условие $s \leq p$. Как только в первый раз условие не будет выполнено, т.е. $s > p$, необходимо выйти из цикла.

Далее приводится программа для решения задачи при $a = 2$, $h = 3$, $p = 41$. В программе текущее значение номера члена суммы обозначено через n . Значение этой переменной, при котором впервые оказалась $s > p$, и будет результатом решения задачи:

```
s = 0
n = 0
a = 2
h = 3
p = 41
while s <= p:
    m = a + n * h;
    s = s + m
    n = n + 1
print(n)
6
```

Пример 2.9. Вычислить $s = \cos x + \frac{\cos 2x}{2} + \dots + \frac{\cos nx}{n}$ при $x = 0,5$. Суммирование прекратить, когда очередной член суммы по абсолютной величине будет меньше заданного $\varepsilon = 0,0001$. Для вычисления косинуса подключим библиотеку `math`.

```
import math
x = 0.5
eps = 0.0001
s = 0
n = 1
a = 1
while abs(a) > eps:
    a = math.cos(n*x) / n
    s = s + a
    n = n + 1
print('{0:.4}'.format(s))
0.7109
```

Пример 2.10. Корабль должен преодолеть путь в 3000 км. В первый день он прошел 200 км. Каждый следующий день он будет проделывать путь на 5 % больше, чем в предыдущий день. Через какое время он прибудет в порт назначения?

Обозначим путь одного дня через p . Вначале $p = 200$. Путь следующего дня вычисляется как $p = p + 0,05p$, т.е. $p = 1,05p$. Это значение прибавляем к суммарному пути s : $s = s + p$. Количество дней обозначим через n и будем увеличивать его каждый раз на 1:

```
s = 0.0
p = 200.0
n = 0
while s < 3000:
    s+= p
    n+= 1
    p*= 1.05
print(n)
12
```

2.3. Вложенные циклы. Вычисление рядов

Пример 2.11. Вычислить $s = \sum_{i=0}^{12} \frac{(2x)^i}{i!}$ при x , изменяющемся в пределах от 0,1 до 1 с шагом 0,05.

Вначале ограничимся вычислением суммы при заданном значении x . Здесь член суммы необходимо вычислять рекуррентно. Для вывода рекуррентной формулы выпишем выражения для двух последовательных членов суммы, например $(i-1)$ -го и i -го, и, разделив i -й член на $(i-1)$ -й, получим выражение, на которое необходимо домножить $(i-1)$ -й член для получения i -го. Итак,

$$a_{i-1} = \frac{(2x)^{i-1}}{(i-1)!}, \quad a_i = \frac{(2x)^i}{i!}, \quad \frac{a_i}{a_{i-1}} = \frac{2x}{i}.$$

Таким образом, чтобы получить i -й член из предыдущего $(i-1)$ -го члена, его нужно домножить на $2x/i$.

Вычисление суммы для фиксированного значения x может быть осуществлено следующим образом:

```

x = 0.1
s = 1
a = 1
for i in range(1, 13):
    a = a * 2 * x / i
    s = s + a
print("{0:8.1} {1:8.5}".format(x,s))

```

Эта последовательность операторов должна быть выполнена в цикле по x . Для этого используем цикл по номеру точки:

```

xh = 0.1
xk = 1.0
h = 0.05
n = int((xk - xh) / h + 1)
x = xh
for j in range(n):
    s = 1
    a = 1
    for i in range(1,13):
        a = a * 2 * x / i
        s = s + a
    print("{0:8.2f} {1:<8.5f}".format(x,s))
    x+=h

```

Такая структура программы, когда цикл выполняется внутри другого цикла, называется *вложенными циклами*. Далее приводится результат выполнения программы.

```

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
0.10  1.22140
0.15  1.34986
0.20  1.49182
0.25  1.64872
0.30  1.82212
0.35  2.01375
0.40  2.22554
0.45  2.45960
0.50  2.71828
0.55  3.00417
0.60  3.32012
0.65  3.66930
0.70  4.05520
0.75  4.48169
0.80  4.95303
0.85  5.47395
0.90  6.04965
0.95  6.68589
1.00  7.38905
-----
Press any key to continue . . .

```

Пример 2.12. Вычислить сумму

$$s = 1 + \frac{x^2}{2} + \frac{x^4}{8} + \dots + (-1)^{n-1} \frac{(2n-1)x^{2n}}{(2n)!}.$$

для значений x , изменяющихся в пределах от 0,2 до 1 с шагом 0,2. Суммирование прекращать, когда очередной член суммы по абсолютной величине станет меньше $\varepsilon = 0,0001$.

Задача сводится к организации вложенных циклов. Внешний цикл по счетчику обеспечивает изменение x . Во внутреннем цикле осуществляется вычисление суммы бесконечного ряда.

Член суммы a_i имеет более сложный вид, чем в предыдущих примерах. Его целесообразно представить в виде двух сомножителей:

$$a_i = c_i(2i - 1),$$

где $c_i = (-1)^{i-1} \frac{x^{2i}}{(2i)!}$ будем вычислять по рекуррентной формуле,

выражая последующий член через предыдущий:

$$c_i = -c_{i-1} \frac{x^2}{((2i-1)2i)}.$$

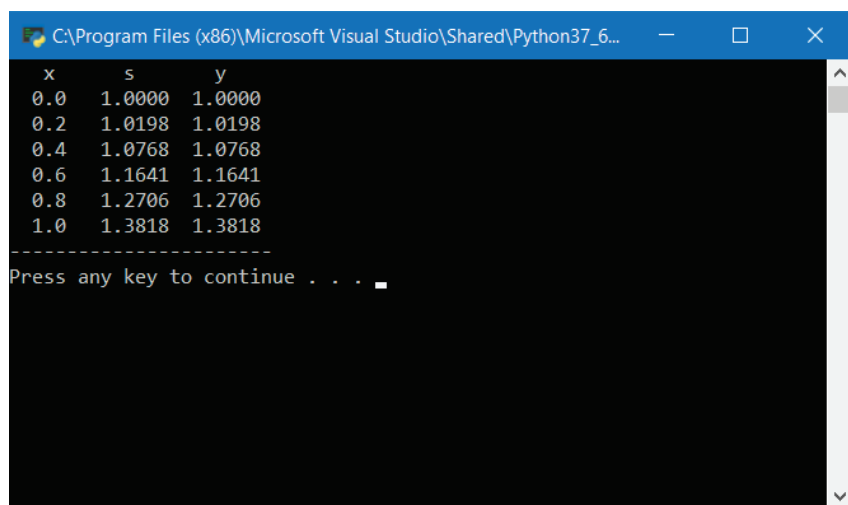
Число значений x на отрезке от 0 до 1 с шагом 0,2 равно 6. В программе для контроля при каждом значении x вычисляется также функция $y = \cos x + x \sin x$, которая приближенно может быть представлена в виде указанной суммы:

```
import math
xh = 0.0
xk = 1.0
h = 0.2
eps = 0.0001
n=int((xk - xh) / h + 1)
x = xh
print('{0:*^33}\n'.format('Таблица функции'))
print('{0:^7}{1:^8}{2:^8}'.format('x', 's', 'y'))
```



```
for j in range(n):
    s = 1
    c = -1
    i = 1
    a = 1
    while abs(a) >= eps:
        c = -c * x * x / ((2 * i - 1) * 2 * i)
        a = c * (2 * i - 1)
        s = s + a
        i += 1
    y = math.cos(x) + x * math.sin(x)

print("{0:^8.3}{1:<8.4f}{2:<8.4f}".format(x, s, y))
x += h
```



x	s	y
0.0	1.0000	1.0000
0.2	1.0198	1.0198
0.4	1.0768	1.0768
0.6	1.1641	1.1641
0.8	1.2706	1.2706
1.0	1.3818	1.3818

Press any key to continue . . .

Вопросы для самопроверки

1. В чем особенности цикла `for`? Когда он используется?
2. Для чего применяется функция `range()`?
3. Когда и для чего используется цикл `while`?
4. Как можно организовать досрочный выход из цикла? В каких случаях это используется?
5. Для чего используется оператор `continue`?

6. Как можно проверить цикл на экстренный выход?
7. В чем особенности типовых алгоритмов циклической структуры: вычисления суммы n слагаемых, вычисления произведения n сомножителей, табулирования функции?
8. Что такое рекуррентное соотношение?
9. Как вычисляется сумма ряда с использованием рекуррентных соотношений?
10. Когда используются вложенные циклы?

Задания для самостоятельного выполнения

I уровень

Задание I уровня предназначено для приобретения навыков разработки программ циклической структуры при известном количестве повторений. Для предлагаемой задачи нужно определить, что является ее решением (число, таблица и пр.); выделить повторяющиеся действия; составить программу, используя оператор цикла `for`; выполнить программу вручную, после чего проверить ее работу на компьютере.

1. Вычислить $s = 2 + 5 + 8 + \dots + 35$.
2. Вычислить $s = 1 + 1/2 + 1/3 + 1/4 + \dots + 1/10$.
3. Вычислить $s = 2/3 + 4/5 + 6/7 + \dots + 112/113$.
4. Вычислить $s = \cos x + (\cos^2 x) / x + (\cos^3 x) / x^2 + \dots + (\cos^9 x) / x^8$.
5. Вычислить сумму квадратов 10 членов арифметической прогрессии

$$s = p^2 + (p + h)^2 + \dots + (p + 9h)^2.$$

6. Получить таблицу функции $y(x) = 0,5x^2 - 7x$ при изменении x от -4 до 4 с шагом $0,5$.
7. Вычислить значение факториала числа 6 ($6! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot 6$).
8. Вычислить $s = 1! + 2! + \dots + 6!$
9. Вычислить $s = (-1)^1 \cdot 5^1 / 1! + (-1)^2 \cdot 5^2 / 2! + \dots + (-1)^6 \cdot 5^6 / 6!$
10. Возвести число 3 в 7-ю степень, не используя операцию возведения в степень.
11. Напечатать заданную последовательность чисел:
а) 1 2 3 4 5 6;

6) 5 5 5 5 5 5.

12. Вычислить при заданном x сумму $s = 1 + 1/x + 1/x^2 + \dots + 1/x^{10}$.

13. Составить таблицу значений функции

$$y = \begin{cases} 1, & x \leq -1; \\ -x, & -1 < x \leq 1; \\ -1, & x > 1 \end{cases}$$

на отрезке $-1,5 \leq x \leq 1,5$ с шагом $h = 0,1$.

14. Напечатать 8 первых членов последовательности, образованной по следующему правилу: первые два числа равны 1; каждое последующее (начиная с третьего) образуется путем суммирования двух предыдущих. (Такая последовательность называется числами Фибоначчи.)

15. Вычислить 5-й член последовательности, образованной дробями $1/1, 2/1, 3/2, \dots$, т.е. числитель (знаменатель) следующего члена последовательности получается сложением числителей (знаменателей) двух предыдущих членов.

16. По древней легенде мудрец, который изобрел шахматы, потребовал от персидского шаха такое количество пшеницы, чтобы им можно было покрыть шахматную доску, положив на первую клетку одно зерно, на вторую – 2, на третью – 4 и т.д., т.е. помещая на каждую следующую клетку в два раза больше зерен, чем на предыдущую. Какое количество зерна может покрыть доску? (Считать, что в одном грамме 15 зерен.) С какой точностью получен результат?

17. Считая, что Земля – идеальная сфера с радиусом $R \approx 6350$ км, определить расстояние до линии горизонта с высоты 1, 2, ..., 10 км.

18. Одноклеточная амеба каждые 3 часа делится на 2 клетки. Определить, сколько клеток будет через 3, 6, 9, ..., 24 часа, если первоначально в замкнутом объеме находилось 10 клеток.

II уровень

Задание II уровня предназначено для приобретения навыков организации циклов по условию. Для предлагаемой задачи

нужно определить, что является ее решением; выделить повторяющиеся действия и записать их в общем виде; определить условие окончания цикла; составить программу, используя оператор цикла `while`; выполнить программу вручную, после чего проверить ее работу на компьютере.

1. Вычислить сумму $s = \cos x + (\cos^2 x) / 2^2 + \dots + (\cos nx) / n^2 + \dots$. Суммирование прекратить, когда очередной член суммы по модулю будет меньше $\varepsilon = 0,0001$.

2. Определить наибольшее значение сомножителя n , для которого произведение $p = 1 \cdot 4 \cdot 7 \cdot \dots \cdot n$ не превышает $L = 30\,000$.

3. Определить количество членов арифметической прогрессии, сумма которых $s = a + (a + h) + \dots + (a + nh)$ не превышает заданного числа p .

4. Вычислить $s = 1 + x^2 + x^4 + \dots + x^{2n}$ ($|x| < 1$). Вычисления прекратить, когда очередной член суммы будет меньше $\varepsilon = 0,0001$.

5. Определить частное и остаток от деления двух целых чисел N и M , используя операцию вычитания.

6. В задаче 18 уровня I определить, через какое время в замкнутом объеме будет находиться 10^5 клеток.

7. Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый следующий день он увеличивал дневную норму на 10 % от нормы предыдущего дня. Определить:

а) какой суммарный путь пробежит спортсмен за 7 дней;

б) через сколько дней спортсмен пробежит суммарный путь 100 км;

в) через сколько дней спортсмен будет пробегать в день больше 20 км.

8. Вкладчик положил в банк 10 000 рублей под 8 % в месяц. Определить, через какое время сумма удвоится.

9. Определить, сколько раз нужно разрезать пополам металлическую нить длиной $L = 0,1$ м, чтобы ее длина уменьшилась до атома (размер атома D_A считать равным 10^{-10} м).

10. В задаче 15 уровня I вычислить член последовательности, который отличается от предыдущего члена не более чем на 0,001.

III уровень

Задание III уровня предназначено для приобретения навыков организации вложенных циклов. Требуется использования типовых алгоритмов циклической структуры в сочетании. Выполнить также все пункты задания II уровня.

Вычислить сумму s , прекращая суммирование, когда очередной член суммы по абсолютной величине станет меньше 0,0001, при изменении аргумента x в указанном диапазоне $[a, b]$ с шагом h . Для сравнения в каждой точке вычислить также функцию $y = f(x)$, являющуюся аналитическим выражением ряда.

$$1. \quad s = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}, \quad y = \cos x, \quad a = 0,1; \quad b = 1; \quad h = 0,1.$$

$$2. \quad s = \sum_{i=1}^{\infty} x^i \sin\left(\frac{i\pi}{4}\right), \quad y = \frac{x \sin\left(\frac{\pi}{4}\right)}{1 - 2x \cos\left(\frac{\pi}{4}\right) + x^2}, \quad a = 0,1; \quad b = 0,8; \quad h = 0,1.$$

$$3. \quad s = 1 + \sum_{i=1}^{\infty} \frac{\cos(ix)}{i!}, \quad y = e^{\cos x} \cos(\sin x), \quad a = 0,1; \quad b = 1; \quad h = 0,1.$$

$$4. \quad s = \sum_{i=0}^{\infty} \frac{(2i+1)x^{2i}}{i!}, \quad y = (1+2x^2)e^{x^2}, \quad a = 0,1; \quad b = 1; \quad h = 0,1.$$

$$5. \quad s = \sum_{i=1}^{\infty} (-1)^i \frac{\cos(ix)}{i^2}, \quad y = \frac{x^2 - \frac{\pi^2}{3}}{4}, \quad a = \frac{\pi}{5}, \quad b = \pi, \quad h = \frac{\pi}{25}.$$

$$6. \quad s = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^{2i+1}}{4i^2 - 1}, \quad y = \frac{(1+x^2) \operatorname{arctg} x}{2} - \frac{x}{2}, \quad a = 0,1, \quad b = 1, \quad h = 0,1.$$

$$7. \quad s = \sum_{i=0}^{\infty} \frac{x^{2i}}{(2i)!}, \quad y = \frac{e^x + e^{-x}}{2}, \quad a = 0,1; \quad b = 1; \quad h = 0,05.$$

$$8. \quad s = \sum_{i=0}^{\infty} \frac{(2x)^i}{i!}, \quad y = e^{2x}, \quad a = 0,1; \quad b = 1; \quad h = 0,05.$$

$$9. \quad s = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{2i+1}, \quad y = \operatorname{arctg} x, \quad a = 0,1; \quad b = 0,5; \quad h = 0,05.$$

Указание. В задаче 2 третьего уровня при вычислении суммы для выхода из цикла нужно сравнивать с точностью 0,0001 не весь член суммы, а только x^i , так как второй сомножитель при $i = 4, 8, \dots$ равен 0, что приведет к прекращению суммирования при $i = 4$ и таким образом исказит результат.

ГЛАВА 3. Организация разветвлений.

Разветвления в цикле

Разветвление – это структура, содержащая две ветви, из которых в зависимости от условия будет выполнена только одна. Разветвления организуются с помощью условного оператора `if`:

```
if условие:
    операторы1
[else:
    операторы2]
...
```

Если *условие* имеет значение `True`, то выполняются *операторы1* и происходит переход к оператору, следующему за `if`. Если *условие* имеет значение `False`, то выполняются *операторы2*, расположенные после `else`, и далее выполняется оператор, следующий за условным. Если вторая ветвь отсутствует (такая структура называется «обход»), то в случае невыполнения условия никаких действий не производится и сразу выполняется оператор, следующий за условным.

Далее приводятся примеры программ с использованием разветвлений.

3.1. Организация разветвлений.

Условный оператор. If-блок

Пример 3.1. Составить программу для решения квадратного уравнения $ax^2 + bx + c = 0$. Квадратное уравнение имеет решение, если дискриминант $d = b^2 - 4ac$ не является отрицательным, т.е. после ввода коэффициентов a , b и c нужно вычислить d и проверить условие $d < 0$. Если это условие выполняется, то нужно вывести сообщение «Уравнение не имеет решения». Если условие не выполняется ($d \geq 0$), нужно вычислить и напечатать корни (возможность совпадения корней игнорируется).

```
a = float (input("Введите a"))
b = float (input("Введите b"))
c = float (input("Введите c"))
```

```
d = b * b - 4 * a * c
if d < 0:
    print("Уравнение не имеет решения")
else:
    x1 = (-b + math.sqrt(d)) / (2 * a)
    x2 = (-b - math.sqrt(d)) / (2 * a)
    print("x1 = %f, x2 = %f" % (x1, x2))
```

Пример 3.2. Вывести большее из двух чисел:

```
x = float (input("Введите первое число"))
y = float (input("Введите второе число"))
u = x
if y > u:
    u = y

print(u)
```

Пример 3.3. Найти максимальное из трех чисел:

```
x = float (input("Введите первое число"))
y = float (input("Введите второе число"))
z = float (input("Введите третье число"))
u = x
if y > u:
    u = y
if z > u:
    u = z
print(u)
```

Пример 3.4. Определить, лежит ли точка с координатами $(x; y)$ внутри или вне круга радиусом r с центром в начале координат:

```
inside = "точка внутри круга"
outside = "точка вне круга"
x = float (input("Введите x"))
y = float (input("Введите y"))
r = float (input("Введите r"))
if x * x + y * y <= r * r:
    print(inside)
else:
    print(outside)
```


Здесь использованы символьные переменные `inside` и `outside` для размещения соответствующей информации, которая выводится в качестве результата.

Второй вариант программы для решения этой задачи составлен с использованием логической переменной, в которую помещается результат (`True`, если точка внутри круга, `False` – в противоположном случае):

```
L = False
x = float (input("Введите x"))
y = float (input("Введите y"))
r = float (input("Введите r"))
if x * x + y * y <= r * r:
    L = not L
print(L)
```

В операторе `if` логической переменной `L` присваивается значение `True`: выполняется операция отрицания, если условие выполняется, т.е. точка лежит внутри круга.

Пример 3.5. Определить, лежит ли точка с координатами $(x; y)$ внутри или вне квадрата, вершинами которого являются точки с координатами $(1; 1)$, $(-1; 1)$, $(1; -1)$, $(-1; -1)$.

Точка лежит внутри квадрата, если одновременно выполняются условия $|x| \leq 1$ и $|y| \leq 1$. Результат будем получать в логической переменной:

```
L = False
x = float (input("Введите x"))
y = float (input("Введите y"))
if abs(x) <= 1 and abs(y) <= 1:
    L = not L
print(L)
```

В операторе `if` логической переменной `L` присваивается значение `True`: выполняется логическая операция «И», если условие выполняется, т.е. точка лежит внутри квадрата.

Пример 3.6. Определить, лежит ли точка с координатами $(x; y)$ внутри или вне треугольника с вершинами в точках $(0; -1)$, $(0; 1)$, $(1; 0)$.

Точка принадлежит треугольнику, если одновременно выполняются два условия $x \geq 0$ и $x + |y| \leq 1$.

```
L = False
x = float (input("Введите x"))
y = float (input("Введите y"))
if x >= 0 and abs(y) + x <= 1:
    L = not L
print(L)
```

Пример 3.7. Вводя координаты n точек $(x; y)$, определить, сколько из них попадет в круг радиусом r с центром в начале координат:

```
n = int (input("Введите количество точек"))
r = float (input("Введите радиус"))
k = 0;
for i in range(n):
    x = float (input("Введите x"))
    y = float (input("Введите y"))
    if x * x + y * y <= r * r:
        k+= 1
print("В круг попало {0} точек из {1}".format(k,n))
```

Пример 3.8. Ввести координаты точки $(x; y)$. Вычислить по выбору радиус-вектор точки, площадь прямоугольника с вершинами в точках $(0; 0)$, $(x; 0)$, $(0; y)$, $(x; y)$ или длину окружности, проходящей через точку $(x; y)$, с центром в начале координат.

Вариант вычислений будем задавать цифрой 1, 2 или 3:

```
print ("Введите x")
x = float(input())
print ("Введите y")
y = float(input())
print("Введите признак: \n 1 = радиус-вектор,\n 2 -
площадь прямоугольника,\n 3 - длина окружности")
k = int(input("Сделайте выбор:\n"))
if k is 1:
    r = pow((x * x + y * y),0.5)
    print("радиус-вектор = {0:5.3}".format(r))
elif k is 2:
```

```
r = x * y
print("площадь прямоугольника = {0:5.3}".format(r))
elif k is 3:
    r = 2 * 3.14 * pow((x * x + y * y), 0.5)
    print("длина окружности = {0:5.3}".format(r))
else: print("Неверный выбор! Выберите 1, 2 или 3")
```

3.2. Составление программ для обработки потока данных

Часто требуется обрабатывать одинаковым образом данные, поступающие последовательно друг за другом. Возможны два случая:

- количество данных известно до начала выполнения программы. Тогда это значение используется в качестве верхней границы счетчика цикла;
- количество данных заранее неизвестно. В этом случае для окончания ввода используется специальное значение того же типа, что и вводимые данные (признак конца ввода), которое заведомо не может встретиться в потоке данных.

Например, если все вводимые числа положительные, то в качестве признака конца может использоваться отрицательное число. При этом если одновременно вводятся несколько данных, то для организации цикла нужно выбрать одно из них (обычно первое), которое будет вводиться отдельно от остальных.

Следующие примеры иллюстрируют эти приемы.

Пример 3.9. Определить средний рост учеников класса (в классе n учеников). В цикле будем вводить в переменную r – рост очередного ученика и прибавлять его к сумме s . После выхода из цикла разделим суммарный рост на количество учеников (программа составлена для $n = 5$).

```
s = 0
for i in range(5):
    r = float (input("Введите рост ученика"))
    s+= r
sr = s / 5
print(sr)
```

Пример 3.10. Определить средний рост школьников, проходящих диспансеризацию (количество заранее неизвестно).

Чтобы определить средний рост, будем в цикле вводить в переменную r – рост очередного ученика и прибавлять его к сумме s , а количество учеников n увеличивать на 1. После выхода из цикла разделим суммарный рост на количество школьников.

Используем безусловный цикл `while`, для выхода из цикла будем использовать специальное значение, вводимое в r (на-пример, 0):

```
s = 0
n = 0
while True:
    print("Введите рост ученика, для окончания -0")
    r = float (input())
    if r == 0:
        break
    s+= r
    n+= 1
sr = s / n
print(sr)
```

Пример 3.11. Определить средний рост и вес школьников, проходящих диспансеризацию.

Для организации цикла будем использовать специальное значение, вводимое в переменную r (рост), и вводить его отдельно:

```
s = 0
sw = 0
n = 0
while True:
    print("Введите рост ученика, для окончания -0")
    r = float (input())
    if r == 0:
        break
    print("Введите вес ученика")
    w = float (input())
    s+= r
```

```
sw+= w
n+= 1
sr = s / n
swr = sw / n
print("Средний рост {0:.6} Средний вес {1:.5}").
format(sr, swr))
```

Пример 3.12. Вводя координаты (x ; y) произвольного числа точек, определить в процентах долю точек, попавших в круг радиусом r с центром в начале координат.

Для окончания ввода будем использовать специальное значение. Пусть для всех вводимых точек $x < 1000$. Тогда значение 1000 будем использовать для окончания ввода.

```
n = 0
k = 0
print("Введите радиус" )
r = float (input())
while True:
    print("Введите x, для окончания 1000")
    x = float (input())
    if x >= 1000:
        break
    print("Введите y")
    y = float (input())
    n = n + 1
    if x * x + y * y <= r * r:
        k = k + 1
if (n != 0):
    k = k * 100 / n
print("В круг попало k = {0:.4}% точек".format(k))
```

Вопросы для самопроверки

1. Что такое разветвление? Какой оператор используется для организации разветвлений?

2. Как реализуется типовая структура «Обход»? В чем ее особенность?

3. Что такое логические переменные, логические операторы, логические выражения? Для чего их используют?

4. Как организуется обработка потока данных, количество которых заранее не известно?

5. В чем заключаются особенности организации потокового ввода, если каждая порция данных включает несколько значений?

Задания для самостоятельного выполнения

I уровень

Задание I уровня предназначено для приобретения навыков организации разветвлений. Необходимо сформулировать задачу математически; определить, что является ее решением; составить список используемых переменных; составить программу; выполнить программу вручную, после чего проверить ее работу на компьютере.

1. На плоскости расположена окружность радиусом r с центром в начале координат. Ввести заданные координаты точки и определить, лежит ли она на окружности. Решить задачу при $r = 2$ для точек с координатами $(0; 2)$, $(1,5; 0,7)$, $(1; 1)$, $(3; 0)$.

Указание. Считать, что точка с координатами $(x; y)$ лежит на окружности радиусом r , если $|x^2 + y^2 - r^2| \leq 10^{-3}$.

2. Определить, лежит ли заданная точка внутри или вне треугольника с вершинами в точках $(-1; 0)$, $(1; 0)$, $(0; 1)$.

Указание. Уравнение прямой, ограничивающей фигуру слева: $y = 1 + x$ ($x < 0$), справа: $y = 1 - x$ ($x \geq 0$). Следовательно, точка принадлежит фигуре, если $y \geq 0$ и $y + |x| \leq 1$.

3. Для заданных a и b получить $c = \max(a, b)$, если $a > 0$ или $c = \min(a, b)$, если $a \leq 0$.

4. Для заданных a, b, c вычислить $z = \max(\min(a, b), c)$.

5. Заданы площади: круга – r и квадрата – s . Определить, поместится ли квадрат в круге. Задачу решить при: 1) $r = 70$; $s = 36,74$; 2) $r = 0,86$; $s = 0,74$.

6. Для задачи 5 определить, поместится ли круг в квадрате. Задачу решить при: 1) $r = 3,2$; $s = 3,5$; 2) $r = 3,2$; $s = 4$; 3) $r = 6$; $s = 9$.

7. Вычислить значение функции y при заданном значении аргумента x по формуле $y = 1$, если $|x| > 1$, или $y = |x|$, если $|x| \leq 1$.

8. Вычислить значение функции y при заданном значении аргумента x по формуле $y = 0$, если $|x| \geq 1$, или $y = x^2 - 1$, если $|x| < 1$.

9. Вычислить значение функции y при заданном значении аргумента x по формуле $y = 0$, если $x \geq -1$, или $y = 1 + x$, если $-1 < x \leq 0$, или $y = 1$, если $x > 0$.

10. Вычислить значение функции y при заданном значении аргумента x по формуле $y = 1$, если $x \leq -1$, или $y = -x$, если $-1 < x \leq 1$, или $y = -1$, если $x > 1$.

II уровень

Задание II уровня требует сочетания циклов и разветвлений. Предполагается, что количество вводимых исходных данных n задано. Выполнить также все пункты задания I уровня.

1. Определить средний рост девочек и мальчиков одного класса. В классе учится n учеников.

2. В компьютер вводятся по очереди координаты n точек. Определить, сколько из них попадет в круг радиусом r с центром в точке $(a; b)$.

3. Ученику 1-го класса назначается дополнительно стакан молока (200 мл), если его вес составляет меньше 30 кг. Определить, сколько литров молока потребуется ежедневно для одного класса, состоящего из n учеников. После взвешивания вес каждого ученика вводится в компьютер.

4. В компьютер вводятся по очереди координаты n точек. Определить, сколько из них попадет в кольцо с внутренним радиусом r_1 и внешним r_2 .

5. В соревнованиях по бегу принимают участие 30 спортсменов. Вводя по очереди результаты участников, определить, сколько из них выполнили заданный норматив.

6. В компьютер по очереди вводятся координаты n точек. Определить, сколько из них принадлежит фигуре, ограниченной осью абсцисс и аркой синусоиды, построенной для аргумента от 0 до π .

7. В компьютер вводятся координаты n точек, лежащих на плоскости. После ввода координат каждой точки выводится номер квадранта, в котором она находится. Определить количество точек, лежащих по отдельности в 1-м и 3-м квадрантах.

8. В компьютер вводятся координаты n точек, лежащих на плоскости. Напечатать номер точки, ближайшей к началу координат, и величину расстояния от нее до начала координат.

9. В соревнованиях по плаванию на 200 м участвуют n спортсменов. Вывести на печать лучший результат.

10. В группе учатся n студентов. Каждый получил на экзаменах по 4 оценки. Подсчитать число студентов, не имеющих «2» и «3».

11. В группе учатся n студентов. Каждый сдал 4 экзамена. Подсчитать число неуспевающих студентов и средний балл группы.

12. Вводя n значений r , вычислить по выбору площадь квадрата со стороной r , площадь круга радиусом r или площадь равностороннего треугольника со стороной r .

13. Для n пар значений A, B вычислить по выбору площадь прямоугольника со сторонами A, B ; площадь кольца, заключенного между двумя окружностями с радиусами A и B ; площадь равнобедренного треугольника со сторонами A, B, B .

III уровень

Решить задачи II уровня для случая, когда количество данных заранее неизвестно. В программе необходимо обеспечить прекращение ввода, как только входной поток иссякнет.

ГЛАВА 4. Структурированные типы данных. Типовые алгоритмы обработки и их реализация

В *Python* существуют четыре встроенных структурированных типа данных: список (`list`), кортеж (`tuple`), словарь (`dict`) и множество (`set`, `frozenset`).

Список используется для хранения однородных (и не только) объектов. Так как в *Python* отсутствуют массивы в традиционном понимании этого термина, вместо них как правило используются списки.

Кортеж – это список, который после создания нельзя изменить.

Словарь – это неупорядоченный набор пар ключ-значение.

Множество – коллекция неповторяющихся данных, хранящая эти данные в случайном порядке.

Далее рассмотрим эти структуры более подробно. При этом основное внимание уделим спискам, рассматривая их как аналог массивов.

4.1. Списки

Массивом в программировании называется структура данных, содержащая несколько значений одного типа, обозначаемых одним именем. Доступ к элементам массива осуществляется по индексу. Изменяя индексы, можно переходить от одного элемента массива к другому и таким образом обрабатывать единообразно большие наборы данных, используя циклы. Индексация массивов обычно начинается с нуля. Массивы могут быть одномерными, многомерными, вложенными. *Одномерный массив* (вектор) представляет собой линейную структуру. Положение элемента определяется одним индексом. *Двухмерный массив* можно представить себе как таблицу. Положение элемента определяется двумя индексами: номером строки и номером столбца.

В *Python* такая структура в явном виде отсутствует (хотя есть возможность подключения внешней библиотеки `array` для работы с массивами). Для отображения массивов в *Python* обычно используют списки.

Список (`list`) является одним из самых часто используемых типов в *Python*. Он представляет собой упорядоченную последовательность (коллекцию) элементов и в принципе может включать в себя объекты любого типа и использовать любые уровни вложенности. Отметим, что здесь мы будем рассматривать только списки, содержащие однородные элементы, и рассматривать их как аналог массивов.

Начнем рассмотрение с одномерных массивов (векторов).

В табл. 4.1 приведены основные операции списков.

Таблица 4.1

Основные операции списков

Операция	Результат
<code>x in s</code>	True, если какой-либо элемент <i>s</i> равен <i>x</i> , иначе False
<code>x not in s</code>	False, если какой-либо элемент <i>s</i> равен <i>x</i> , иначе True
<code>s + t</code>	Объединение (конкатенация) <i>s</i> и <i>t</i>
<code>s * n</code> (или <code>n * s</code>)	Повторение <i>s</i> <i>n</i> раз
<code>s[i]</code>	<i>i</i> -й элемент <i>s</i>
<code>s[i:j]</code>	Срез из <i>s</i> от <i>i</i> до <i>j</i>
<code>s[i:j:k]</code>	Срез из <i>s</i> от <i>i</i> до <i>j</i> с шагом <i>k</i>
<code>len(s)</code>	Длина <i>s</i>
<code>min(s)</code>	Наименьший элемент из <i>s</i>
<code>max(s)</code>	Наибольший элемент из <i>s</i>
<code>s.index(x[,i[,j]])</code>	Индекс первого вхождения <i>x</i> в <i>s</i> (или после индекса <i>i</i> и перед индексом <i>j</i>)
<code>s.count(x)</code>	Количество вхождений <i>x</i> в <i>s</i>

Замечание. Списки (как и словари) являются изменяемым типом данных. Это значит, если создать список *Y*, сделать присваивание *X = Y*, а затем изменить *X*, то *Y* тоже изменится! За этим нужно внимательно следить! Для того чтобы этого не происходило, необходимо создать новый объект, эквивалентный исходному, например, так: *X = Y[:]*. Например:

```
M = L = [i for i in range(2,12,3)]
print (M, L)
[2, 5, 8, 11] [2, 5, 8, 11]
L[2] = 112
print (M, L)
[2, 5, 112, 11] [2, 5, 112, 11]
```

```
M = L[:]  
L[2] = 0  
print (M, L)  
[2, 5, 112, 11] [2, 5, 0, 11]
```

Или, если требуется обеспечить неизменность созданного объекта, можно использовать вместо списков кортежи.

4.1.1. Одномерные массивы

Списки могут быть сконструированы несколькими способами.

Для обозначения пустого списка используется пара квадратных скобок (`[]`):

```
y = []
```

Инициализацию можно выполнить и при объявлении переменной списка, используя квадратные скобки и разделяя элементы запятыми:

```
x = [1, 6, 2]
```

Также можно использовать генератор списков [`x for x in iterable`]:

```
z = [a**2+1 for a in (19, 3, 5, 0)]
```

и конструктор типа: `list()` или `list(iterable)`.

Индексация элементов массива по умолчанию начинается с 0. Таким образом, массив размером n содержит элементы с индексами от 0 до $n - 1$. Описанный выше массив z будет содержать элементы с $z[0]$ по $z[3]$.

Доступ к отдельному элементу массива осуществляется заданием индекса, указываемого в квадратных скобках после имени массива. Например:

```
w = [6, 14, 2, 5]  
w[3] = 8
```

Здесь описан массив w , содержащий 4 элемента от $w[0]$ до $w[3]$; 4-му (последнему) элементу этого массива присваивается значение 8.

Вводить данные в массив можно и с клавиатуры. Например, для ввода массива a размером 5 с элементами типа `int` необходимо использовать цикл

```
a = [int(input()) for i in range(5)]
```

Здесь, как и при вводе значений простых переменных, нужно вводить одно число (значение очередного элемента массива), после чего нажимать клавишу [Enter].

Можно набирать значения элементов массива на клавиатуре в одной строке, разделяя их пробелами (или любым другим разделителем). Например:

```
v = input("Введите элементы массива, разделяя значения элементов пробелом ")
L = [int(i) for i in v.split(" ")]
print(L)
```

Здесь в строке последовательно набираются значения элементов массива, разделяемые пробелами. Далее производится *разбор* этой строки с помощью метода `Split()`. Определяется положение первого пробела в строке `v`, символы (цифры), расположенные перед ним, преобразуются в тип `int` и пересылаются в первый элемент массива. Далее определяется положение следующего пробела в строке `v`, символы, расположенные между текущим и предыдущим пробелами, преобразуются в тип `int` и пересылаются в следующий элемент списка, и так продолжается до тех пор, пока не будет закончен разбор всей строки.

Также можно использовать в качестве исходных данных случайные числа.

Модуль `random` включает в себя функцию `random()`, которая возвращает действительное число в диапазоне от 0,0 до 1,0; для целых чисел – функцию `randint(начало, конец)`, для вещественных – функцию `uniform(начало, конец)`:

```
import random
L1 = [random.random() for i in range(5)]
L2 = [random.randint(1,100) for i in range(5)]
L3 = [random.uniform(10,80) for i in range(5)]
```

Для вывода массива также можно воспользоваться несколькими способами. Можно напечатать массив целиком:

```
x = [6.0, 14.11, 2.3456]
print (x)
[6.0, 14.11, 2.3456]
```

А также поэлементно, в строку или в столбец:

```
for i in range (3):
```

```

    print (x[i], end=' ')
print ()
6.0 14.11 2.3456

```

```

for i in range (3):
    print (x[i])
6.0
14.11
2.3456

```

К такому же результату приводит

```

for I in x:
    print (I)

```

При выводе массива необходимо обеспечить наглядность и удобство восприятия выводимых данных. Как правило, вывод одномерного массива целесообразно осуществлять в строку с использованием формата

```

for i in range (3):
    print ("{:<8.2f}".format(x[i]), end='')
print ()
6.00  14.11 2.35

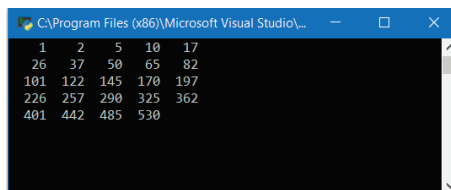
```

Рассмотрим вывод одномерного массива размером n по m элементов в строке. Код приведен для $n = 24$, $m = 5$:

```

n = 24
m = 5
z=[i**2+1 for i in range(n)]
for i in range (n):
    print ("{:>5d}".format(z[i]), end='')
    if ((i+1) % m == 0):
        print ()
print ()

```



Если номер очередного выведенного элемента $(i + 1)$ кратен 5 (результат его деления нацело на 5 равен 0), то выполняется оператор пустой `print()` и происходит перевод строки.

4.1.2. Алгоритмы обработки одномерных массивов

Рассмотрим типовые алгоритмы обработки массивов, используя для их представления списки.

Заметим, что хотя в языке *Python* для структурированных типов данных существует набор операций, реализующих стандартные алгоритмы (основные операции, которые поддерживаются списками, приведены в табл. 4.1 и 4.2), их модификация для решения конкретных задач бывает затруднена (либо невозможна), также их использование часто приводит к неоправданному увеличению времени работы программы, особенно при больших объемах данных. Поэтому знание типовых алгоритмов обработки массивов, умение их модифицировать и грамотно использовать в каждом конкретном случае является одним из важнейших навыков при разработке алгоритмов и составлении программ.

Таблица 4.2

Основные методы списков

Метод	Результат
<code>.remove()</code>	Удаление элемента по значению (удаляет только первый элемент с таким значением)
<code>.pop()</code>	Удаление и возвращение элемента по индексу
<code>.clear()</code>	Удаление всех элементов из списка
<code>del</code>	Удаление элемента по индексу
<code>.append()</code>	Добавление элемента в конец списка
<code>.extend()</code>	Включение добавления в список из переданного списка
<code>.insert()</code>	Включение элемента в список перед элементом с заданным индексом
<code>.sort()</code>	Сортировка списка
<code>.reverse()</code>	Инвертирование списка

Далее приводятся фрагменты кода программ с необходимыми пояснениями.

1. *Вычислить сумму элементов массива:*

```
x = [1, 6, 2, -4, 1]
```

```
print("исходный список")
for i in range (5):
    print(x[i], end=' ')
print ()
S = 0
for i in range (5):
    S+=x[i]
print("S = ",S)
print("проверка: sum = ",sum(x))
исходный список
1 6 2 -4 1
S = 6
проверка: sum = 6
```

Здесь на каждом шаге к сумме добавляется очередной элемент массива. Доступ к элементу осуществляется по индексу. Для перебора индексов используется генератор `range()`.

Для решения этой задачи в принципе можно использовать цикл `while`, перебирая индексы «вручную»:

```
i = 0
S = 0
while i < 5:
    S+=x[i]
    i+=1
print("S = ",S)
```

Однако найти сумму элементов этого массива можно проще, используя `for` для перебора элементов массива без использования индексов. Например, приведенный выше фрагмент будет выглядеть так:

```
x = [1,6,2,-4,1]
S = 0
for el in x:
    S+=el
print("S= ",S)
```

Здесь после ключевого слова `for` используем переменную, которой при каждой итерации будет последовательно присва-

иваться значение очередного элемента массива, и ее можно будет использовать в цикле вместо переменной с индексом.

Заметим, что для указания длины массива можно использовать функцию `len()`, например:

```
i = 0
S = 0
while i < len(x):
    S+=x[i]
    i+=1
print("S = ",S)
```

2. Вычислить среднее арифметическое положительных элементов массива размером 5:

```
a = [4.0, 3.1, 4.2, -5.0, -7.9]
print ("исходный список")
for el in a:
    print("{0:<5.1f}".format(el), end=' ')
print ()
s = 0
m = 0
for el in a:
    if el > 0:
        s = s + el
        m += 1
sr = s / m;
print("sr = {0:5.2f}".format(sr))
исходный список
4.0 3.1 4.2 -5.0 -7.9
sr = 3.77
```

Здесь в переменной *s* накапливается сумма положительных элементов, а в переменной *m* – их количество. После выхода из цикла остается разделить сумму на количество.

3. Найти сумму элементов, расположенных до первого отрицательного элемента массива.

```
a = [4.0, 3.1, -4.2, 5.0, -7.9]
print ("исходный список")
```



```
print (a)
s = 0
for el in a:
    if el > 0:
        s = s + el
    else:
        break
print("s = {0:4.2f}".format(s))
исходный список
[4.0, 3.1, -4.2, 5.0, -7.9]
s = 7.10
```

Оператор `break` завершает цикл, если не будет выполнено условие `el > 0` (т.е. встретится первый отрицательный элемент). Произойдет выход из цикла и суммирование закончится.

Возможен другой вариант программы, в котором вместо разветвления в цикле реализуется обход, что является более предпочтительным с точки зрения структурного подхода к разработке алгоритма:

```
a = [4.0, 3.1, -4.2, 5.0, -7.9]
print ("исходный список")
print (a)
s = 0
for el in a:
    if el < 0:
        break
    s = s + el
print("s = {0:4.2f}".format(s))
исходный список
[4.0, 3.1, -4.2, 5.0, -7.9]
s = 7.10
```

4. Перестановка элементов вектора. В примере переставлены нулевой и первый элементы.

```
a = [4.0, 3.1, -4.2, 5.0, -7.9]
print ("исходный список ", a)
p = a[0]
a[0] = a[1]
```

```
a[1] = p
print("результат ",a)
исходный список    [4.0, 3.1, -4.2, 5.0, -7.9]
результат          [3.1, 4.0, -4.2, 5.0, -7.9]
```

При перестановке используется вспомогательная переменная (в данном случае *p*).

В общем виде перестановка *i*-го и *j*-го элементов осуществляется следующим образом:

```
p = a[i]
a[i] = a[j]
a[j] = p
```

Заметим, что обмен значений переменных (в том числе элементов списка) можно выполнить в одну строчку, используя так называемое *множественное присваивание*:

```
a, b = b, a
```

При выполнении этого оператора интерпретатор *Python* сначала получает значения, связанные с переменными *b* и *a* (правая часть), и помещает их в кортеж: (*b*, *a*) (подробнее о кортежах см. ниже). После этого он связывает каждый элемент кортежа в определенной позиции с переменной в той же позиции, но в кортеже слева (*a*, *b*). Таким образом можно поменять значения не только двух переменных, но и трех, четырех и т.д., в том числе можно обменять значения переменных разных типов.

С использованием множественного присваивания программа будет иметь вид

```
a = [4.0, 3.1, -4.2, 5.0, -7.9]
print ("исходный список ",a)
a[0],a[1] = a[1],a[0]
print("результат ",a)
исходный список    [4.0, 3.1, -4.2, 5.0, -7.9]
результат          [3.1, 4.0, -4.2, 5.0, -7.9]
```

5. Последний отрицательный элемент массива поменять с первым элементом массива:

```
a = [4, -3, 2, -5, 9]
print ("исходный список ",a)
```

```
j = 0
for i in range (5):
    if a[i] < 0:
        j = i
p = a[j]
a[j] = a[0]
a[0] = p
print("результат ",a)
```

Здесь массив просматривается с начала, и номер очередного встретившегося отрицательного элемента помещается в переменную *j*. Таким образом, по завершении цикла в ней останется номер последнего отрицательного элемента массива. Перестановка элементов выполняется с использованием вспомогательной переменной *p*.

Однако этот вариант не является оптимальным с точки зрения времени выполнения программы, поскольку массив в любом случае будет просмотрен целиком. Рассмотрим еще один вариант:

```
a = [4, -3, 2, -5, 9]
print ("исходный список ",a)
j = 0
for i in range (4, -1, -1):
    if a[i] < 0:
        j = i
        break
a[0],a[j] = a[j],a[0]
print("результат ",a)
исходный список      [4, -3, 2, -5, 9]
результат             [-5, -3, 2, 4, 9]
```

Здесь массив просматривается с конца (начиная с последнего элемента, с шагом -1), и первый встретившийся отрицательный элемент будет последним отрицательным элементом в массиве. Перестановка элементов выполняется с использованием множественного присваивания. Предложите самостоятельно вариант кода (с использованием оператора `for`) без явного использования индексов элементов.

6. Удалить элемент с заданным индексом из массива.

Удалить элемент с заданным индексом k означает сдвинуть все следующие за ним элементы на одну позицию влево, т.е. выполнить операцию $a_i = a_{i+1}$ для $i = k, k + 1, \dots, n - 1$:

```
a = [4, -3, 2, -5, 9]
print ("исходный список ",end='')
n = len(a)
for i in range (n):
    print ("{:0:4d}".format(a[i]), end='')
print ()
k = 1
n = n - 1
for i in range(k,n):
    a[i] = a[i + 1];
print("результат ",end='')
for i in range (n):
    print ("{:0:4d}".format(a[i]), end='')
print ()
print ("полученный список ",a)
исходный список      4 -3 2 -5 9
результат             4 2 -5 9
полученный список [4, 2, -5, 9, 9]
```

Размер массива уменьшился на 1. При этом на месте последнего элемента исходного массива остается старое значение, но оно не будет нас интересовать, так как формально не входит в состав результирующего массива (ср. поэлементный вывод результата и вывод всего списка целиком).

С использованием операции удаления элемента с заданным индексом программа будет выглядеть следующим образом:

```
a = [4, -3, 2, -5, 9]
del a[k]
print ("проверка ",a, end='')
проверка      [4, 2, -5, 9]
```

Поэтому для удаления элемента с заданным индексом использование функции `del` практически всегда предпочтительнее.

7. Включить заданное значение P после k -го элемента массива.

Размер массива при этом должен увеличиться на 1. Это необходимо предусмотреть при задании массива, добавив в конце пустую ячейку (`None`).

Чтобы не потерять $(k + 1)$ -й элемент исходного массива, необходимо освободить место для нового значения, передвинув вправо на одну позицию все элементы, начиная с $(k + 1)$ -го, т.е. выполнить операцию $a_{i+1} = a_i$ для $i = n, n - 1, \dots, k + 1$:

```
n = 5
a = [4, -3, 2, -5, 9, None]
print ("исходный список ",end='')
for i in range (n):
    print ("{:0:4d}".format(a[i]), end='')
print ()
k = 1
P = 10
for i in range(n,k + 1, - 1):
    a[i] = a[i - 1]
n = n + 1
a[k + 1] = P
print("результат ",end='')
for i in range (n):
    print ("{:0:4d}".format(a[i]), end='')
print ()
исходный список      4 -3 2 -5 9
результат            4 -3 10 2 -5 9
```

Перемещать элементы нужно начиная с последнего. В противном случае элементы массива, расположенные после $(k + 1)$ -го элемента, будут иметь одинаковое значение, равное значению $(k + 1)$ -го элемента. Убедитесь в этом самостоятельно.

С использованием операции включения элемента в список на заданную позицию программа будет выглядеть следующим образом:

```
a = [4, -3, 2, -5, 9]
```

```

k = 1
P = 10
a.insert(k + 1, P)
print ("проверка ", a, end='')
проверка      [4, -3, 10, 2, -5, 9]

```

8. Сформировать массив из элементов другого массива, удовлетворяющих заданному условию.

Пусть требуется переслать в массив *b* положительные элементы массива *a*. Сразу заметим, что индексы одинаковых по значению элементов массивов *a* и *b* не будут совпадать. Цикл организуем по индексам элементов массива *a*, индекс пересылаемого элемента в массиве *b* будем каждый раз увеличивать на 1:

```

a = [4, -3, 2, -5, 9]
print "исходный список ", end=''
for i in range (len(a)):
    print ("{:0:4d}".format(a[i]), end='')
print ()
print("результат ", end='')
b = [None] * len(a)
j = 0
for i in range(len(a)):
    if a[i]>0:
        b[j]=a[i]
        j+=1

```

После выхода из цикла в переменной *j* будет содержаться количество значимых элементов массива *b* (значение индекса последнего элемента будет при этом на 1 меньше), и для вывода массива *b* необходимо организовать следующий цикл:

```

for i in range (j):
    print ("{:0:4d}".format(b[i]), end='')
print ()
исходный список      4   -3  2   -5  9
результат             4   2   9

```

Замечание. Так как массив *b* содержит лишние – пустые – элементы, можно для их удаления воспользоваться функцией `del` для удаления всех элементов, начиная с *j*-го:

```
print (b)
del b[-(len(a) - j):]
print (b)
[4, 2, 9, None, None]
[4, 2, 9]
```

Чтобы этого избежать, можно до создания нового списка подсчитать количество элементов, удовлетворяющих условию, и сразу создать список нужного размера:

```
a = [4, -3, 2, -5, 9]
j = 0
for el in a:
    if el > 0:
        j+=1
b = [None] * j
```

Но это приводит к необходимости проходить по исходному списку «лишний» раз, что может быть неэффективно в случае списка большого размера.

Для проверки воспользуемся генератором с условием для включения элемента в новый список:

```
a = [4, -3, 2, -5, 9]
b = [i for i in a if i > 0 ]
print ("проверка ",end='')
print (b)
проверка [4, 2, 9]
```

Замечание. Условие, расположенное после итератора – это фильтр, при выполнении которого элемент пойдет в финальное выражение; если элемент ему не удовлетворяет, он будет пропущен!

9. Найти максимальный элемент массива и его индекс.

Алгоритм поиска массива заключается в последовательном сравнении всех элементов массива с переменной *атах*. Если очередной элемент окажется больше, то поместим его значение в *атах*. Таким образом, после завершения цикла в *атах* окажется значение наибольшего элемента массива.

Этот алгоритм будет работать корректно, если в качестве начального значения *amax* будет выбрано значение, не превосходящее по величине максимальный элемент массива. Обычно для конкретной задачи не трудно подобрать начальное значение. Например, если в массиве *a* хранятся оценки студентов одной группы, то в качестве начального значения *amax* можно использовать любое отрицательное число, так как все оценки имеют положительные значения. Однако в общем случае (если про значения, хранящиеся в массиве, заранее ничего неизвестно) в качестве начального присваивания можно выбрать любой элемент массива, поскольку никакой из них не может превосходить значение максимального элемента. Обычно для начального присваивания выбирают начальный элемент ($amax = a_0$):

```
a = [3, 5, -1, 7, 9, 6]
amax = a[0]
for el in a:
    if (el > amax):
        amax = el
print ("значение =", amax)
print ("проверка ", max(a))
значение = 9
проверка    9
```

Поиск минимального элемента массива осуществляется аналогично с той лишь разницей, что в операторе *if* нужно использовать оператор отношения «меньше» (<), а не «больше» (>).

Чтобы найти индекс максимального элемента, в переменную *imax* параллельно помещают номер того элемента, который замещает *amax*:

```
a = [3, 5, -1, 7, 9, 6]
amax = a[0]
imax = 0
for i in range(1, len(a)):
    if a[i] > amax:
        amax = a[i]
        imax = i
print ("значение =", amax, ", индекс =", imax)
print ("проверка")
```



```
z = max(a)
print ("значение =", z, ", индекс =", a.index(z))
значение = 9 , индекс = 4
проверка
значение = 9 , индекс = 4
```

Если в массиве имеется *несколько максимальных элементов*, у которых значения одинаковые, а индексы разные, то в переменной, в которой сохраняется индекс, будет сохранен индекс первого из них. Чтобы получить индекс последнего, достаточно в операторе `if` вместо строгого (`>`) проверять нестрогое (`>=`) условие.

Решите задачу нахождения индекса последнего максимального элемента с помощью операций обработки списков самостоятельно. Изменится ли количество проходов списка?

Если нужно запомнить индексы всех максимальных элементов, то необходимо предусмотреть массив для хранения их индексов, например *im*, в котором будут запоминаться индексы одинаковых максимальных элементов. При изменении значения *amax* массив *im* начнет заполняться сначала. Если очередной элемент a_i равен максимальному, то в следующий элемент массива *im* помещается текущий индекс *i*:

```
a = [9, 9, -1, 7, 9, 6]
im = [None]*len(a)
k=0
amax = a[0]
for i in range(len(a)):
    if a[i] > amax:
        amax = a[i]
        k = 0
        im[k] = i
        k+= 1
    elif a[i] == amax:
        im[k] = i
        k+= 1
for j in range(k):
    print (im[j], " ", end="")
print()
```

```
print (im)
del im[-(len(a) -k):]
print (im)
0 1 4
[0 1 4 None None None]
[0 1 4]
```

После выхода из цикла количество элементов массива *im* будет находиться в переменной *k*. Поэтому оставшиеся пустые элементы удаляются (это удобнее, чем изменять размер списка на каждом шаге алгоритма!).

10. Упорядочить массив.

Требуется расположить элементы массива *a* в определенном порядке, например по убыванию.

Рассмотрим алгоритм упорядочения, базирующийся на двух уже рассмотренных алгоритмах, а именно на алгоритмах поиска максимального элемента и перестановки элементов.

В цикле при $i = 0, 1, 2, \dots, n - 2$ будем устанавливать на *i*-е место нужный элемент. Для этого найдем максимальный элемент в части массива, начиная с *i*-го элемента, и далее поменяем местами найденный элемент с *i*-м. После окончания цикла элементы массива будут расположены в порядке убывания, при этом на последнем ($n - 1$) - м месте окажется самый маленький элемент:

```
a = [3, 5, 1, 7, 9, 6]
a1 = a
print ("исходный массив ",a)
for i in range(len(a)-1):
    amax = a[i]
    imax = i
    for j in range(i+1,len(a)):
        if (a[j] > amax):
            amax = a[j]
            imax = j
    a[imax] = a[i]
    a[i] = amax
print ("упорядоченный массив",a)
```

```
a1.sort() # упорядочение по возрастанию
a1.sort(reverse=True) # инвертирование
print ("проверка ",a1)
исходный массив      [3, 5, 1, 7, 9, 6]
упорядоченный массив [9, 7, 6, 5, 3, 1]
проверка              [9, 7, 6, 5, 3, 1]
```

Обратите внимание, что в отличие от `sort()`, функция `sorted()` не меняет исходную коллекцию, а возвращает новый список из ее элементов:

```
x = [1,6,-8,0,3]
print(x)
x1 = sorted(x)
print(x1)
print(x)
x.sort()
print (x)
[1, 6, -8, 0, 3]
[-8, 0, 1, 3, 6]
[1, 6, -8, 0, 3]
[-8, 0, 1, 3, 6]
```

Типовые алгоритмы 11–15 приводятся без соответствующих кодов. Рекомендуется разработать их самостоятельно, используя приведенные схемы алгоритмов, и проверить их на компьютере.

11. Поиск заданного элемента в упорядоченном массиве (бинарный поиск).

Поиск в упорядоченном массиве осуществляется существенно быстрее, чем в неупорядоченном. Поэтому часто целесообразно предварительно произвести упорядочение массива, особенно в случае необходимости многократного поиска и больших массивов.

Требуется в массиве a размером n , упорядоченном по возрастанию, определить наличие заданного элемента x и его индекс, если такой элемент найден.

Схема алгоритма

- x сравнивается со средним элементом массива.

- Индекс среднего элемента $i = (i_1 + i_2) / 2$, где $i_1 = 0$, $i_2 = n - 1$.
- Если $x = a_i$, задача решена.
- Если $x < a_i$, то поиск продолжается в левой половине: $i_2 = i - 1$, i_1 не изменяется.
- Если $x > a_i$, то поиск продолжается в правой половине: $i_1 = i + 1$, i_2 не изменяется.
- Искомый элемент отсутствует в массиве, если выполнится условие $i_2 < i_1$.

12. Объединение двух массивов с чередованием элементов.

Требуется объединить два массива одинакового размера $A = [a_0, a_1, \dots, a_{n-1}]$ и $B = [b_0, b_1, \dots, b_{n-1}]$ в один массив $C = [a_0, b_0, a_1, b_1, \dots, a_{n-1}, b_{n-1}]$.

Элементами массива C с четными индексами являются элементы массива A :

$$c_0 = a_0; c_2 = a_1; \dots; c_{2i} = a_i, \dots,$$

элементами с нечетными индексами – элементы массива B :

$$c_1 = b_0, c_3 = b_1, \dots, c_{2i-1} = b_i, \dots$$

Таким образом, требуется организовать цикл и выполнить операции

$$c_{2i} = a_i, c_{2i-1} = b_{i-1}, \text{ для } i = 0, 1, 2, \dots, n - 1.$$

Если массивы имеют разные размеры, то больший массив обрезается по размеру меньшего, и меньший массив и урезанный большой объединяются в соответствии с предложенным алгоритмом. Далее оставшиеся элементы большего массива пересылаются подряд.

13. Объединение двух упорядоченных массивов в один с сохранением упорядоченности.

Требуется объединить два упорядоченных по убыванию массива A размером n и B размером m в один массив C размером $(n + m)$, также упорядоченный.

Схема алгоритма

- Начиная с первых элементов массивов A и B , сравниваем элементы a_i и b_j . В массив C пересылаем больший из них, например a_i (если выполняется условие $a_i \geq b_j$).

- Далее продолжаем пересылать в массив C элементы массива A (индекс j при этом не меняется), пока очередной элемент массива A не будет меньше b_j (условие $a_i \geq b_j$ не выполняется).

- Тогда начинаем пересылать в массив C элементы массива B (индекс i при этом не меняется), пока для какого-либо элемента массива B не будет снова выполнено условие ($a_i \geq b_j$).

- Если один из массивов исчерпан (выполнено условие $i \geq n$ или $j \geq m$), то оставшиеся элементы другого массива пересылаются друг за другом без всяких проверок.

Реализовать алгоритм самостоятельно.

14. Инвертирование массива.

Требуется изменить порядок следования элементов массива размером n на обратный, т.е. поменять местами нулевой элемент массива с $(n - 1)$ -м, первый с $(n - 2)$ -м, i -й с $(n - (i + 1))$ -м. Последними нужно поменять местами средние элементы, т.е. $(n / 2)$ -й и $(n / 2 + 1)$ -й.

15. Циклический сдвиг.

Требуется переместить элементы массива A размером n на t позиций вправо. При этом t элементов из конца массива перемещаются в начало:

$A = [a_0, a_1, a_2, a_3, a_4]$ – исходный массив;

$A = [a_3, a_4, a_0, a_1, a_2]$ – после циклической перестановки на 2 позиции вправо.

Вариант 1. Используется вспомогательный массив для временного хранения t последних элементов массива A . Далее оставшиеся элементы с нулевого по $(n - t - 1)$ -й смещаются вправо на t позиций. (Заметим, что перемещение нужно начинать с последнего из перемещаемых элементов, чтобы не испортить элементы массива A .) После этого в первые t элементов пересылаются элементы вспомогательного массива.

Вариант 2. Циклический сдвиг осуществляется с использованием одной вспомогательной переменной, в которую каждый раз пересылается последний элемент массива A , после чего все элементы сдвигаются вправо на 1 позицию (начиная с конца), и на место первого элемента помещается значение вспомогательной переменной. Эта процедура повторяется t раз.

В заключение рассмотрим несколько примеров решения задач.

Пример 4.1. Задан массив x , содержащий 12 элементов. Вставить после максимального элемента новый элемент, значение которого равно среднему арифметическому положительных элементов, расположенных перед максимальным элементом:

```
n = 12
x = [5.0, -7.0, -9.0, 6.0, 4.0, 12.0, 2.0, 8.0, 7.0,
6.0, 5.0, 3.0, 1.0, 2.0, None]
for i in range(n):
    print(x[i], end=" ")
print()
# Поиск максимального элемента и его индекса
xmax = x[0]
imax = 0
for i in range(n):
    if (x[i] > xmax):
        xmax = x[i]
        imax = i
# Нахождение среднего арифметического
s = 0
k = 0
for i in range(imax):
    if (x[i] > 0):
        s+= x[i]
        k+= 1;
sr = s / k
# включение нового элемента
for i in range(n,imax+1,-1):
    x[i] = x[i-1]
x[imax + 1] = sr
n+= 1
for i in range(n):
    print(x[i], end=" ")
5.0 -7.0 -9.0 6.0 4.0 12.0 2.0 8.0 7.0 6.0 5.0 3.0
5.0 -7.0 -9.0 6.0 4.0 12.0 5.0 2.0 8.0 7.0 6.0 5.0 3.0
```

Пример 4.2. Упорядочить положительные элементы массива *a*, содержащего 7 элементов, по убыванию, оставив отрицательные элементы массива на прежних местах:

```
a = [4, 7, 9, -10, 3, -12, 8]
print ("исходный массив ",a)
for i in range(len(a)-1):
    if a[i] > 0:
        amax = a[i]
        imax = i
        for j in range(i + 1,len(a)):
            if (a[j] > amax):
                amax = a[j]
                imax = j
        a[imax] = a[i]
        a[i] = amax
print ("упорядоченный массив",a)
исходный массив      [4, 7, 9, -10, 3, -12, 8]
упорядоченный массив [9, 8, 7, -10, 4, -12, 3]
```

Пример 4.3. Включить заданное значение *p* в качестве элемента в массив *x*, содержащий 12 элементов, с сохранением упорядоченности (массив упорядочен по убыванию):

```
n = 12
a = [18, 15, 14, 13, 12, 9, 8, 7, 6, 5, 4, 3, None]
print ("исходный массив")
for i in range(n):
    print (a[i], " ", end="")
print()
P = 10
for i in range(n):
    if a[i] < P:
        break
for k in range(n, i, -1):
    a[k] = a[k - 1]
a[i] = P
print(a)
18, 15, 14, 13, 12, 9, 8, 7, 6, 5, 4, 3
[18, 15, 14, 13, 12, 10, 9, 8, 7, 6, 5, 4, 3]
```

Пример 4.4. Массив M содержит оценки 25 студентов по курсу «Информатика». Определить, сколько студентов получили оценку «5», «4», «3», «2» или «0» (в случае неявки на экзамен).

Исходные данные содержатся в массиве M размером 25. Результат получим в массиве K размером 5, где каждый элемент содержит количество соответствующих оценок. Сформируем также вспомогательный массив MK размером 5, в который поместим сами оценки:

```
M = [4, 4, 5, 0, 2,
      3, 0, 3, 3, 3,
      4, 3, 0, 4, 4,
      5, 2, 4, 3, 4,
      3, 3, 5, 4, 4]
MK = [5, 4, 3, 2, 0]
K = [0]*5
for oc in M:
    if oc == 0:
        K[4] += 1
    else:
        K[5 - oc] += 1
print («оценка количество»)
for i in range(len(MK)):
    print (" ", MK[i], " ", K[i])
```

Здесь само значение оценки в массиве M (т.е. M_i) связано с индексом j соответствующего элемента в массиве MK : $j = 5 - M_i$, т.е. количество оценок «5» будет помещаться в 1-й элемент массива MK , количество оценок «4» – во 2-й элемент массива MK и т.д. Исключение составляют только значения «0», количество которых необходимо поместить в последний элемент массива MK , что и сделано в программе отдельно.

4.13. Матрицы. Типовые алгоритмы обработки матриц

Для работы с матрицами в *Python* также используются списки. При этом каждый элемент списка-матрицы содержит вложенный список.

Матрица (двухмерный массив) реализуется в *Python* как список списков, таким образом, получается структура из вло-

женных списков, количество которых определяет количество строк матрицы, а число элементов внутри каждого вложенного списка указывает на количество столбцов в исходной матрице.

Положение элемента в массиве определяется двумя индексами: номером строки и номером столбца. Нумерация, как и для одномерных массивов, начинается с нуля.

Доступ к элементу массива осуществляется указанием двух индексов. Рассмотрим пример матрицы размером 3×4:

```
matrix = [[0, 1, 2, 3],
          [10, 11, 12, 13],
          [20, 21, 22, 23]]
print(matrix)
print(matrix[1][2])
[[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23]]
12
```

Каждая строка матрицы представляет собой одномерный массив. Поэтому при обработке матриц в основном используются типовые алгоритмы обработки одномерных массивов.

При работе с матрицами, как правило, используются вложенные циклы, например в цикле по строкам необходимо получить доступ к каждому элементу строки, т.е. организовать цикл по столбцам.

Ввод матриц можно осуществлять поэлементно с использованием вложенных циклов. Элементы матрицы вводятся, как правило, по строкам. При этом для ввода строки используются те же способы, что и для одномерных массивов.

Вывод матриц должен осуществляться в наглядной форме (в виде таблицы), т.е. каждая строка матрицы должна выводиться в новую строку экрана с использованием подходящего формата:

```
matrix = [[0, 1, 2, 3],
          [10, 11, 12, 13],
          [20, 21, 22, 23]]
for i in range(len(matrix)):
    for j in range(len(matrix[i])):
        print("{:4d}".format(matrix[i][j]), end = " ")
    print()
```

```
0  1  2  3
10 11 12 13
20 21 22 23
```

Замечание. `len(matrix)` – количество строк в матрице.

В заключение отметим, что хотя для обработки матриц существует библиотека `Numpy`, содержащая большое количество функций для работы с матрицами, в рамках данного учебника основное внимание уделяется использованию типовых алгоритмов обработки массивов.

1. Просуммировать элементы строк матрицы *a*. Результат получить в виде вектора (одномерного массива) *b*:

```
a = [[0, 1, 2, 3],
      [10, 11, 12, 13],
      [20, 21, 22, 23]]
b = [0] * len(a)
for i in range(len(a)):
    for j in range (len(a[i])):
        print("{:4d}".format(a[i][j]), end = " ")
    print()
print()
for i in range(len(a)):
    S = 0
    for j in range (len(a[i])):
        S+= a[i][j]
    b[i] = S
print ("результат:")
for i in range(len(b)):
    print("{:4d}".format(b[i]), end = " ")
print()
0  1  2  3
10 11 12 13
20 21 22 23
результат:
6  26 46
```

Аналогично осуществляется формирование одномерных массивов из значений максимальных (минимальных) элемен-

тов строк, индексов максимальных (минимальных) элементов строк и т.п. (см. соответствующие алгоритмы для одномерных массивов).

Если необходимо осуществить обработку столбцов матрицы, то внешний цикл необходимо организовать по номеру столбца, а внутренний – по номеру строки. В остальном алгоритмы аналогичны.

Например, нахождение суммы по столбцам:

```
n, m = 4, 6
a = [[10, 12, 26, 28, 5, 31],
      [7, 54, 41, 79, 44, 5],
      [92, 36, 75, 100, 25, 28],
      [32, 6, 54, 54, 13, 60]]
b = [0] * m
for i in range(n):
    for j in range(m):
        print("{:4d}".format(a[i][j]), end = " ")
    print()
for j in range(m):
    S = 0
    for i in range(n):
        S = S + a[i][j]
    b[j] = S
print(«результат:»)
for i in range(m):
    print("{:4d}".format(b[i]), end = " ")
print()
10   1  26  28   5  31
 7  54  41  79  44   5
92  36  75 100  25  28
32   6  54  54  13  60
результат:
112 230 356 219  87 124
```

2. Поменять местами ii -ю и jj -ю строки матрицы.

Поменять местами строки означает поменять местами каждую пару соответствующих элементов этих строк, т.е. необходим цикл по столбцам:

```

n, m = 4, 6
a = [[10, 12, 26, 28, 5, 31],
      [7, 54, 41, 79, 44, 5],
      [92, 36, 75, 100, 25, 28],
      [32, 6, 54, 54, 13, 60]]
for i in range(n):
    for j in range(m):
        print("{:4d}".format(a[i][j]), end = " ")
    print()
print("результат:")
ii, jj = 0, 1
for j in range(m):
    p = a[ii][j]
    a[ii][j] = a[jj][j]
    a[jj][j] = p
for i in range(n):
    for j in range(m):
        print("{:4d}".format(a[i][j]), end = " ")
    print()
print()
10  1  26  28   5  31
 7  54  41  79  44   5
92  36  75 100  25  28
32   6  54  54  13  60
результат:
 7  54  41  79  44   5
10  1  26  28   5  31
92  36  75 100  25  28
32   6  54  54  13  60

```

Столбцы меняются местами аналогично (цикл организуется по строкам).

3. Удалить k -ю строку матрицы.

Чтобы удалить строку, необходимо переместить все строки, расположенные после k -й, на одну позицию вверх. Для перемещения строки нужно организовать цикл по элементам строки, т.е. по столбцам (здесь приведен вариант с поэлементной обработкой строки):

```

n, m = 4, 6
a = [[10, 12, 26, 28, 5, 31],
      [7, 54, 41, 79, 44, 5],
      [92, 36, 75, 100, 25, 28],
      [32, 6, 54, 54, 13, 60]]
for i in range(n):
    for j in range(m):
        print("{:4d}".format(a[i][j]), end = " ")
    print()
k = 1
n -= 1
for i in range(k, n):
    for j in range(m):
        a[i][j] = a[i + 1][j]
print("результат:")
for i in range(n):
    for j in range(m):
        print("{:4d}".format(a[i][j]), end = " ")
    print()
10  1  26  28   5  31
 7  54  41  79  44   5
92  36  75 100  25  28
32   6  54  54  13  60
результат:
10  1  26  28   5  31
92  36  75 100  25  28
32   6  54  54  13  60

```

Удаление столбца осуществляется аналогично (внутренний цикл по элементам столбца, т.е. по строкам).

4. Вставить новую строку, заданную вектором b размером t , после k -й строки матрицы.

Вначале нужно освободить место для новой строки, переместив строки, расположенные после k -й, на одну позицию вниз. (При объявлении матрицы предусмотреть необходимость соответствующего увеличения ее размера.)

Перемещение строк нужно начинать с последней строки (см. аналогичный алгоритм для одномерных массивов). В этом

варианте программы каждая строка рассматривается как единое целое (элемент исходного списка):

```
n, m = 4, 6
k = 1
a = [[10, 12, 26, 28, 5, 31], [7, 54, 41, 79, 44,
5], [92, 36, 75, 100, 25, 28], [32, 6, 54, 54, 13,
60], [None]*m]
for i in range(n):
    for j in range(m):
        print("{:4d}".format(a[i][j]), end = " ")
    print()
print()
b = [1, 5, 3, 8, 7, 0]
for j in range(m):
    print("{:4d}".format(b[j]), end = " ")
print()
for i in range(n, k-1, -1):
    a[i] = a[i-1]
n+= 1
a[k] = b
print()
print(«результат»)
for i in range(n):
    for j in range(m):
        print("{:4d}".format(a[i][j]), end = " ")
    print()
```

Заметим, что тот же результат можно получить, используя операцию вставки:

```
a1.insert(k,b)
```

При решении задач, где требуется иметь дело не со строками, а со столбцами матрицы, используется поэлементная обработка.

Для вставки вектора b в качестве k -го столбца в матрицу a можно использовать, например:

```
for i in range(n):
    a[i].insert(k, b[i])
```

5. Найти сумму элементов матрицы.

Здесь нужно обратиться к каждому элементу матрицы и добавить его к сумме:

```
n, m = 3, 4
a = [[random.randrange(0,10) for x in range(m)] for y
in range(n)]
for i in range(n):
    for j in range(m):
        print ("{:4d}".format(a[i][j]), end = " ")
    print()
S = 0
for i in range(3):
    for j in range(3):
        S+= a[i][j]
print("сумма =", S)
```

Аналогично осуществляется поиск максимального элемента во всей матрице.

Далее будут рассмотрены типовые алгоритмы для квадратной матрицы a размером $n \times n$.

6. Найти сумму элементов, расположенных на главной диагонали (след матрицы).

Элементы, расположенные на главной диагонали, имеют одинаковые индексы (номера строк и столбцов совпадают), поэтому для их обработки используется один цикл:

```
S = 0
for i in range(n):
    S+= a[i][i]
```

Аналогично можно организовывать и другие алгоритмы для работы с диагональными элементами (нахождение максимального элемента и т.п.).

7. Найти сумму элементов, расположенных ниже главной диагонали (включая диагональ), т.е. просуммировать элементы нижнего треугольника матрицы.

Здесь во внешнем цикле (по строкам) номера строк изменяются от 0 до $n - 1$, но в каждой i -й строке суммируются только элементы, расположенные до диагонального элемента этой строки, т.е. до i -го:

```
S = 0
for i in range(n):
    for j in range(i+1):
        S+= a[i][j]
```

8. Транспонирование матрицы с получением результата в том же массиве.

Для квадратной матрицы размером $n \times n$ требуется переставлять элементы, расположенные симметрично относительно главной диагонали:

```
for i in range(n):
    for j in range(i,n):
        p = a[i][j]
        a[i][j] = a[j][i]
        a[j][i] = p
```

Заметим, что формально тот же результат можно получить и с использованием генератора выражений, но при этом будет создана другая матрица:

```
t = [[row[i] for row in a] for i in range(n)]
```

Для прямоугольной матрицы размером $n \times m$ транспонированная матрица может быть получена на месте исходной, если последняя размещена в массиве размером не менее чем $n \times n$ (предполагается, что $n > m$). При этом можно использовать приведенный выше алгоритм. Фиктивные столбцы, дополняющие исходную матрицу до квадратной, помещаются в этом случае в фиктивные строки транспонированной матрицы.

9. Умножение матрицы на вектор.

Требуется умножить матрицу a размером $n \times m$ на вектор b размером m . Для этого необходимо вычислить

$$c[i] = \sum_{j=0}^{m-1} a[i,j]b[j], \quad i=0, \dots, n-1.$$

```
n = 3
m = 3
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
b = [3, 4, 1]
c = [0] * m
```



```

for i in range(n):
    s = 0
    for j in range(m):
        s+= a[i][j] * b[j]
    c[i] = s

```

10. Умножение матрицы на матрицу.

Требуется умножить матрицу a размером $n \times k$ на матрицу b размером $k \times m$. Для этого необходимо вычислить

$$c[i, j] = \sum_{l=0}^{k-1} a[i, l] b[l, j]; i=0, \dots, n-1; j=0, \dots, m-1.$$

```

n = m = k = 3
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
b = [[3, 4, 1], [1, 2, 4], [2, 4, 3]]
c = [[0 for j in range(m)] for i in range(n)]
for i in range(n):
    for j in range(m):
        s = 0
        for l in range(k):
            s+= a[i][l]*b[l][j]
        c[i][j] = s

```

11. Определение номера k (нумерация начинается с 0) элемента $a[i, j]$ матрицы размером $n \times m$ ($i = 0, 1, \dots, n-1; j = 0, 1, \dots, m-1$), заданной в виде одномерного массива по строкам:
 $k = im + j$.

12. Определение номера k (нумерация начинается с 0) элемента $a[i, j]$ симметрической матрицы размером $n \times n$ ($i = 0, 1, \dots, n-1; j = 0, 1, \dots, n-1$), заданной своим верхним треугольником в одномерном массиве b размером $(n+1)n/2$:

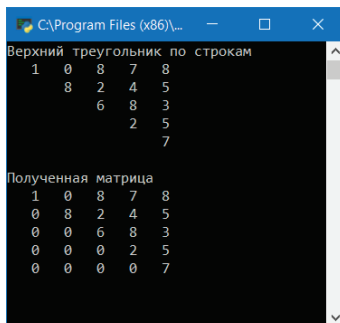
$$k = (2(n-1)-i+1)i/2 + j; i = 0, 1, \dots, n-1; j = i, i+1, \dots, n-1.$$

Элементы нижнего треугольника определяются как $a[i, j] = a[j, i]; i = 1, \dots, n-1; j = 0, \dots, i-1$.

Пример 4.5. В одномерном массиве b хранится по строкам верхний треугольник квадратной матрицы (элементы, расположенные выше главной диагонали, включая главную диаго-

наль). Напечатать его по строкам. Восстановить исходную матрицу, заполнив ее нижний треугольник нулями. Напечатать по строкам:

```
n = 5
k = 0
m = int((n * (n + 1)) / 2)
a = [[0 for x in range(n)] for i in range(n)]
b = [random.randrange(0,10) for i in range(m)]
print ("Верхний треугольник по строкам")
for i in range(n):
    for l in range(i):
        print (' '*4,end = "")
    for j in range(i, n):
        print ("{:4d}".format(b[k]), end = " ")
        k+= 1
    print()
print ()
k = 0
for i in range(n):
    for j in range(i,n):
        a[i][j] = b[k]
        k+= 1
print ("Полученная матрица")
for i in range(n):
    for j in range(n):
        print ("{:4d}".format(a[i][j]), end = " ")
    print()
print()
```



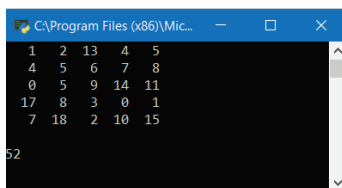
```
C:\Program Files (x86)\Python\Python27\python.exe
Верхний треугольник по строкам
1  0  8  7  8
   8  2  4  5
    6  8  3
     2  5
      7

Полученная матрица
1  0  8  7  8
0  8  2  4  5
0  0  6  8  3
0  0  0  2  5
0  0  0  0  7
```

Пример 4.6. Просуммировать элементы квадратной матрицы размером $n \times n$, расположенные в ее верхней четверти, ограниченной главной и побочной диагоналями, включая элементы, расположенные на диагоналях.

Номера строк в верхней половине матрицы изменяются от 0 до $n/2$. Индексы суммируемых элементов в строке изменяются от i до $n - i$:

```
n = 5
m = int((n + 1) / 2)
s = 0
a = [[1, 2, 13, 4, 5],
      [4, 5, 6, 7, 8],
      [0, 5, 9, 14, 11],
      [17, 8, 3, 0, 1],
      [7, 18, 2, 10, 15]]
for i in range(n):
    for j in range(n):
        print ("{:4d}".format(a[i][j]), end = " ")
    print()
print()
for i in range(m):
    for j in range(i, n-i):
        s+= a[i][j]
print(s)
```



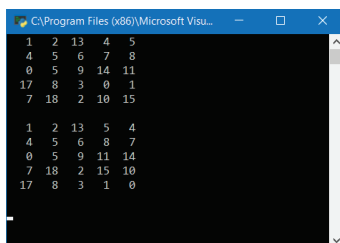
Пример 4.7. Для квадратной матрицы поменять местами минимальный и максимальный элементы главной диагонали, используя перестановку строк и столбцов:

```
n = 5
a = [[1, 2, 13, 4, 5],
      [4, 5, 6, 7, 8],
      [0, 5, 9, 14, 11],
```

```

        [17, 8, 3, 0, 1],
        [7, 18, 2, 10, 15]]
for i in range(n):
    for j in range(n):
        print ("{:4d}".format(a[i][j]), end = " " )
    print()
print()
amax = a[0][0]
imax = 0
amin = a[0][0]
imin = 0
for i in range(n):
    if a[i][i] > amax:
        amax = a[i][i]
        imax = i
    if a[i][i] < amin:
        amin = a[i][i]
        imin = i
for j in range(n):
    p = a[imax][j]
    a[imax][j] = a[imin][j]
    a[imin][j] = p
for i in range(n):
    p = a[i][imax]
    a[i][imax] = a[i][imin]
    a[i][imin] = p
for i in range(n):
    for j in range(n):
        print ("{:4d}".format(a[i][j]), end = " " )
    print()
print()

```



Пример 4.8. Поменять местами максимальный элемент нижнего треугольника (включая главную диагональ) квадратной матрицы x размером $n \times n$ с максимальным элементом верхнего треугольника:

```
n = 6
a = [[random.randrange(0,100) for x in range(n)] for
y in range(n)]
for i in range(n):
    for j in range(n):
        print ("{:4d}".format(a[i][j]), end = " ")
    print()
print()
ad = a[0][0]
id = 0
jd = 0
for i in range(n):
    for j in range(i+1):
        if a[i][j] > ad:
            ad = a[i][j]
            id = i
            jd = j
au = a[0][1]
iu = 0
ju = 1
for i in range(n):
    for j in range(i+1, n):
        if a[i][j] > au:
            au = a[i][j]
            iu = i
            ju = j
a[id][jd] = au
a[iu][ju] = ad
for i in range(n):
    for j in range(n):
        print ("{:4d}".format(a[i][j]), end = " ")
    print()
print()
```

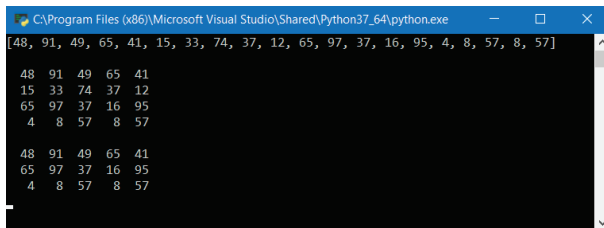
17	3	70	2	96	46
19	1	31	93	1	56
48	4	40	68	62	17
66	52	29	98	65	51
92	74	45	64	1	42
94	69	3	62	25	20
17	3	70	2	98	46
19	1	31	93	1	56
48	4	40	68	62	17
66	52	29	96	65	51
92	74	45	64	1	42
94	69	3	62	25	20

Пример 4.9. В матрице a размером $n \times m$, представленной в виде одномерного массива по строкам, удалить строку, содержащую максимальный элемент в k -м столбце:

```

n, m = 4, 5
N = n * m
a = [random.randrange(0,100) for x in range(N)]
print(a)
print()
for i in range(N):
    print ("{:>4d}".format(a[i]), end = " " )
    if (i + 1) % m == 0:
        print()
print ()
k = 2
amax = a[k]
imax = 0
for i in range (n):
    if a[i * m + k] > amax:
        amax = a[i * m + k]
        imax = i
for i in range (imax, n - 1):
    for j in range (m):
        a[i * m + j] = a[(i + 1) * m + j]
K = 0
N -= m
for i in range(N):
    print ("{:>4d}".format(a[i]), end = " " )
    if (i + 1) % m == 0:
        print()
print ()

```



```

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
[48, 91, 49, 65, 41, 15, 33, 74, 37, 12, 65, 97, 37, 16, 95, 4, 8, 57, 8, 57]

48 91 49 65 41
15 33 74 37 12
65 97 37 16 95
4 8 57 8 57

48 91 49 65 41
65 97 37 16 95
4 8 57 8 57

```

4.2. Словари

Словарь (`dict`), в отличие от списка, представляет собой неиндексированный набор элементов, при этом каждый из них представляет собой пару ключ-значение.

Для обозначения словаря используется пара фигурных скобок (`{}`):

```

y = {}
print (type(D))
<class 'dict'>

```

При этом сам словарь изменяем (можно добавлять / удалять новые пары ключ-значение); значения элементов словаря – изменяемые и не уникальные; а ключи – не изменяемые и уникальные, поэтому, например, мы не можем сделать ключом словаря список (но можем, например, кортеж). Из уникальности ключей следует и уникальность элементов словаря – пар ключ-значение.

Для генерации множества и словаря используются одинаковые скобки, разница в том, что у словаря указывается двойной элемент «ключ: значение».

```

DA = {'A': 1, 'B': 2, 'C': 3, 'D': 3}
print(DA)
D1 = {v: k for k, v in DA.items()}
print(D1)
{'A': 1, 'B': 2, 'C': 3, 'D': 3}
{1: 'A', 2: 'B', 3: 'D'}

```

Обратите внимание, мы «потеряли» C: так как значения C и D были одинаковы, то когда они стали ключами, сохранилось только последнее значение.

Словарь также можно создать из списка:

```
List1 = [-2, 0, 2, 3, 4]
Dict1 = {z: z**2 for z in List1}
print(Dict1)
{-2: 4, 0: 0, 2: 4, 3: 9, 4: 16}
```

Такой синтаксис создания словаря работает только в фигурных скобках, выражение-генератор так создать нельзя (для этого используется другой синтаксис).

К словарям применимы многие операции, аналогичные операциям со списками (основные методы словарей приведены в табл. 4.3), например:

```
D = {'B': 2, 'A': 1, 'E': 5, 'C': 3, 'D': 4}
# печать словаря
print (D)
# подсчет количества членов словаря
# (пара ключ-значение считаются одним элементом)
print (len(D))
# проверка принадлежности элемента словарю
print('a' in D)
print('a' in D.keys())
print('a' in D.values())
print(1 in D.values())
# проверка принадлежности пары словарю
print(('a',1) in D.items())
print(('a',2) in D.items())
# поиск минимального значения
print(min(D.values()))
# суммирование элементов, если они все числовые
print(sum(D.values()))
{'B': 2, 'A': 1, 'E': 5, 'C': 3, 'D': 4}
5
False
False
False
True
False
False
1
15
```


А вот методы, связанные с индексацией, например `.count()`, `.index()` и др. (см. табл. 4.1), к словарям, очевидно, не применимы.

Таблица 4.3

Основные методы словарей

Метод	Результат
<code>.pop()</code>	Удаление элемента и возвращение значения по ключу
<code>.popitem()</code>	Удаление и возвращение случайного элемента
<code>.clear()</code>	Удаление всех элементов из списка
<code>.update()</code>	Для существующих ключей обновление значений элементов, для несуществующих – добавление новых элементов
<code>.sorted()</code>	Создание нового отсортированного словаря
<code>.reverse()</code>	Возвращает элементы в обратном порядке
<code>del</code>	Удаление элемента по ключу
<code>len()</code>	Длина словаря
<code>min()</code>	Наименьшее значение элемента из словаря
<code>max()</code>	Наибольшее значение элемента из словаря
<code>sum()</code>	Суммирование значений элементов, если они все числовые

Есть свои особенности и при поэлементной обработке словарей.

Перебор элементов словаря осуществляется по ключам

```
for el in D:
    print (el, " ", end = " ")
В      А      Е      С      D
```

и приводит к тому же результату, что и

```
for el in D.keys():
    print (el, " ", end = " ")
В      А      Е      С      D
```

Можно перебирать по значениям:

```
for el in D.values():
    print (el, " ", end = " ")
2      1      5      3      4
```

Но чаще всего нужно перебирать пары ключ-значение:

```
for key, value in D.items():
    print(key, value, " ", end = " ")
В 2    А 1    Е 5    С 3    D 4
```

```
k = ('A', 'B', 'C') # ключи
v = [1, 2] # значения
```

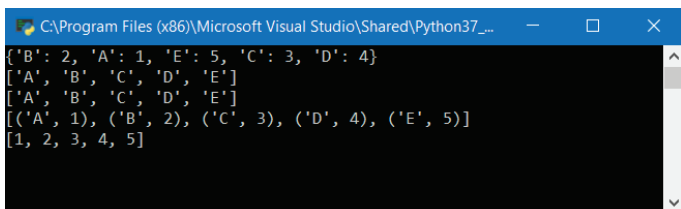
Если количество элементов разное, будут созданы возможные пары, лишние элементы будут отброшены:

```
d = dict(zip(k,v)) # получение словаря из пар
print(d)
print(type(k), type(v), type(d))
{'A': 1, 'B': 2}
<class 'tuple'> <class 'list'> <class 'dict'>
```

Особенности сортировки словаря. В сортировке словаря есть свои особенности, связанные с тем, что элементом словаря является пара ключ-значение.

Говоря о сортировке словаря, мы имеем в виду сортировку полученных из словаря данных для вывода или сохранения в индексированную коллекцию. Сохранить данные сортированными в самом словаре не получится, они в нем, как и других неиндексированных коллекциях, располагаются в произвольном порядке.

```
D = {'B': 2, 'A': 1, 'E': 5, 'C': 3, 'D': 4}
print(D)
# отсортированный список ключей
print(sorted(D))
# тот же результат
print(sorted(D.keys()))
# отсортированный по ключу список элементов
print(sorted(D.items()))
# отсортированный список значений
print(sorted(D.values()))
```



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_...
{'B': 2, 'A': 1, 'E': 5, 'C': 3, 'D': 4}
['A', 'B', 'C', 'D', 'E']
['A', 'B', 'C', 'D', 'E']
[('A', 1), ('B', 2), ('C', 3), ('D', 4), ('E', 5)]
[1, 2, 3, 4, 5]
```

Замечание. Отдельные сложности представляет сортировка словаря в случае, когда нужно получить пары, отсортированные значению. Для решения этой задачи можно в качестве специальной функции сортировки передавать лямбда-функцию `lambda x: x[1]`, которая из получаемых на каждом этапе кортежей (ключ, значение) будет брать для сортировки второй элемент кортежа, например:

```
Ds = sorted(D.items(), key=lambda x: x[1])
```

Для изменения словаря с добавлением элементов другого словаря используется метод `.update()`. Обратите внимание, что для совпадающих ключей словаря при этом обновляются значения:

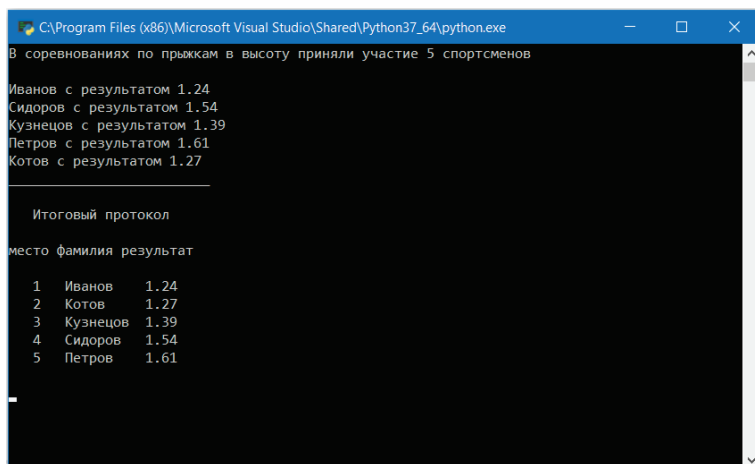
```
dict1 = {'a': 1, 'b': 2}
dict2 = {'a': 100, 'c': 3, 'd': 4}
dict1.update(dict2)
print(dict1)
{'a': 100, 'c': 3, 'b': 2, 'd': 4}
```

Пример 4.10. Протокол соревнований по прыжкам в высоту содержит список фамилий и результатов (одна попытка) в порядке стартовых номеров. Получить итоговую таблицу, содержащую фамилии и результаты в порядке занятых мест.

```
D = {'Иванов': 1.24,
      'Сидоров': 1.54,
      'Кузнецов': 1.39,
      'Петров': 1.61,
      'Котов': 1.27}

print('В соревнованиях по прыжкам в высоту приняли
участие {0} спортсменов\n'.format(len(D)))
for name, res in D.items():
    print('{0} с результатом {1}'.format(name, res))
Ds = sorted(D.items(), key=lambda x: x[1])
i = 1
print('_'*25)
print('\n Итоговый протокол')
print('\nместо фамилия результат\n'.format(len(D)))
for el in Ds:
    print('{0:^7}{1:8}{2:^8}'.format(i, el[0], el[1]))
    i+=1
```

```
print()
```



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
В соревнованиях по прыжкам в высоту приняли участие 5 спортсменов

Иванов с результатом 1.24
Сидоров с результатом 1.54
Кузнецов с результатом 1.39
Петров с результатом 1.61
Котов с результатом 1.27

Итоговый протокол

место фамилия результат

1 Иванов 1.24
2 Котов 1.27
3 Кузнецов 1.39
4 Сидоров 1.54
5 Петров 1.61
```

4.3. Множества

Множество (`set`) и неизменяемое множество (`frozenset`) – это еще два типа коллекций, изменяемый и неизменяемый соответственно. Порядок элементов в множествах не важен!

```
b = {1, 2, 3}
print(type(b))
<class 'set'>
```

Пример генератора множества:

```
listA = [-2, -1, 0, 1, 2, 3, 4, 5]
setA = {i for i in list_a}
print(setA)
{0, 1, 2, 3, 4, 5, -1, -2}
```

Также возможно получение множества преобразованием другой коллекции, например:

```
listA = [2, 5, 7, 2, 5, 1]
setA = set(listA)
print(setA)
{1, 2, 5, 7}
```

Для множеств определены особые методы сравнения:

- `a.isdisjoint(b)` – истина, если `a` и `b` не имеют общих элементов;

- `b.issubset(a)` – истина, если все элементы множества `b` принадлежат множеству `a` (т.е. `b` является подмножеством `a`);
- `a.issuperset(b)` – истина, если условие выше выполнено (т.е. `a` является надмножеством `b`).

```
a = {1, 2, 3}
b = {2, 1}
c = {4}
d = {1, 2, 3}
print(a.isdisjoint(c)) #True (у множеств нет общих
элементов)
print(b.issubset(a)) #True (b – подмножество a)
print(a.issuperset(b)) #True (a – надмножество b)
```

При равенстве множеств они одновременно и подмножество, и надмножество друг для друга.

```
print(a.issuperset(d)) #True
print(a.issubset(d)) #True
```

Также для обоих типов множеств возможны различные варианты комбинации множеств (исходные множества при этом не меняются – возвращается новое множество).

```
a = {'A', 'B'}
b = {'B', 'D'}
#объединение
c1 = a | b
#пересечение
c2 = a & b
#разность
c3 = a - b
#симметрическая разность
c4 = a ^ b
print(c1)
print(c2)
print(c3)
print(c4)
{'A', 'B', 'D'}
{'B'}
{'A'}
{'A', 'D'}
```

Замечание. Для изменяемого множества (`set`) существуют аналогичные операции с изменением исходного множества.

В табл. 4.4 приведены основные методы множеств.

Таблица 4.4

Основные операции и методы множеств

Метод	Результат
<code>.remove()</code>	Удаление элемента
<code>.discard()</code>	Аналог <code>.remove()</code> , не дает ошибки в случае отсутствия элемента
<code>.pop()</code>	Удаление и возвращение случайного элемента
<code>.clear()</code>	Удаление всех элементов
<code>.update()</code>	Замена множества на его объединение с переданным множеством
<code>.add()</code>	Добавление элемента
<code>len()</code>	Длина (количество элементов) множества
<code>min()</code>	Наименьшее значение элемента множества
<code>max()</code>	Наибольшее значение элемента множества
<code>sum()</code>	Суммирование значений элементов, если они все числовые

Замечание. Добавление и удаление элементов возможно только для изменяемого множества (`set`).

4.4. Кортежи

Кортежи (`tuple`) служат для хранения нескольких объектов вместе. Их можно рассматривать как аналог списков, однако они неизменяемы, т.е. модифицировать кортежи невозможно.

```
tuple1 = (1, 2, 3, 4, 5)
print(tuple1[0]) # 1
tuple1[0] = 6 #TypeError: 'tuple' object does not support item assignment
```

Вопросы для самопроверки

1. Что такое массив? Элемент массива? Индекс элемента массива? Как можно инициализировать массив? Как организуется вывод массива в строку, в столбец?

2. Каков алгоритм суммирования элементов массива? Суммирования элементов массива, удовлетворяющих условию?

3. Как сформировать другой массива из элементов заданного массива, удовлетворяющих условию? Как осуществляется перестановка элементов массива?

4. В чем заключается удаление элемента массива? Включение элемента в массив?

5. Как найти максимальный (минимальный) элемент массива?

6. Как осуществляется упорядочение элементов массива?

7. Что такое матрица? Как задается матрица? В чем особенности ввода матрицы по строкам, по столбцам? Как осуществить вывод матрицы в наглядной форме?

8. Каков алгоритм суммирования элементов матрицы? Суммирования элементов матрицы, удовлетворяющих условию? Нахождения максимального (минимального) элемента матрицы?

9. Как осуществляются операции со строками (столбцами) матрицы (поиск максимального элемента, включение, удаление элемента, перестановка элементов и т.п.)?

10. Как реализовать удаление строки (столбца) матрицы? Включение одномерного массива в качестве строки (столбца) в матрицу?

11. В чем особенности операций с главной диагональю, с побочной диагональю?

12. Как осуществляется обработка фрагмента матрицы (верхнего, нижнего треугольника; верхней, нижней, правой, левой четверти; периметра и т.п.)?

13. В чем особенности обработки матрицы, заданной в виде одномерной последовательности?

14. Как осуществляется вывод матрицы, заданной в виде одномерной последовательности?

Задания для самостоятельного выполнения

I уровень

Задание I уровня предназначено для приобретения навыков работы с одномерными массивами с использованием типовых алгоритмов обработки массивов и их сочетаний. Для решения задачи нужно составить программу, самостоятельно запро-

граммировав алгоритм; использовать явное задание значений элементов массивов. Для отображения массивов следует использовать списки. В программе нужно предусмотреть печать исходных данных и печать результата с поясняющим текстом. Для проверки – использовать методы списков.

1. Найти сумму элементов одномерного массива размером 6. Разделить каждый элемент исходного массива на полученное значение. Результат получить в том же массиве.

2. Положительные элементы массива размером 8 заменить средним арифметическим среди положительных элементов.

3. Вычислить сумму и разность двух одномерных массивов размером 4. (Суммой (разностью) двух массивов одинакового размера называется третий массив такого же размера, каждый элемент которого равен сумме (разности) соответствующих элементов исходных массивов.)

4. Найти среднее значение элементов массива размером 5. Преобразовать исходный массив, вычитая из каждого элемента полученное значение.

5. Вычислить скалярное произведение двух векторов размером 4. (Скалярным произведением называется сумма попарных произведений соответствующих элементов массивов.)

6. Вычислить длину вектора размером 5. (Длина вектора вычисляется по формуле $L = \sqrt{x_1^2 + \dots + x_5^2}$.)

7. Элементы одномерного массива размером 7, которые больше среднего значения элементов массива, заменить на 0.

8. Подсчитать количество отрицательных элементов заданного одномерного массива размером 6.

9. Определить, сколько элементов заданного одномерного массива размером 8 больше среднего значения элементов этого массива.

10. Дан одномерный массив размером 10 и два числа P и Q ($P < Q$). Определить, сколько элементов массива заключено между P и Q .

11. Сформировать массив из положительных элементов заданного массива размером 10.

12. Определить значение и номер последнего отрицательного элемента массива размером 8.

13. Дан массив размером 10. Сформировать два массива размером 5, включая в первый массив элементы исходного массива с четными индексами, во второй – с нечетными.

14. Найти сумму квадратов элементов, расположенных до первого отрицательного элемента массива размером 11.

15. Задан массив x размером 10. Вычислить значения функции $y = 0,5 \ln x$ при значениях аргумента, заданных в массиве x . Вычисленные значения поместить в массив y . Вывести массивы x и y в виде двух столбцов.

16. Минимальный элемент заданного одномерного массива увеличить в два раза.

17. В одномерном массиве найти сумму элементов, расположенных до максимального элемента массива.

18. Все элементы одномерного массива, расположенные перед минимальным, увеличить в 2 раза.

19. В одномерном массиве все элементы, расположенные после максимального, заменить средним значением элементов массива.

20. Задан одномерный массив. Сформировать другой одномерный массив из отрицательных элементов, расположенных между максимальным и минимальным элементами исходного массива.

21. Задан одномерный массив и число P . Включить элемент, равный P , после того элемента массива, который наиболее близок к среднему значению его элементов.

22. Увеличить в 2 раза элемент, расположенный непосредственно после максимального элемента массива.

23. Поменять местами максимальный элемент массива и минимальный элемент части массива, расположенной после максимального.

24. Найти среднее арифметическое значение элементов массива, расположенных между минимальным и максимальным элементами массива.

25. Удалить минимальный среди положительных элементов массива.

26. Включить заданный элемент P после последнего положительного элемента массива.

27. Первый отрицательный элемент массива заменить суммой элементов, расположенных после максимального.

28. Максимальный элемент массива среди элементов с четными индексами заменить значением его индекса.

29. Поменять местами максимальный и первый отрицательный элементы массива.

30. Заданы массивы A и B , содержащие n и m элементов соответственно. Вставить массив B между k -м и $(k + 1)$ -м элементами массива A (k задано).

31. Определить индексы элементов массива, меньших среднего. Результат получить в виде массива.

32. Если максимальный элемент расположен до минимального, то найти среднее арифметическое положительных элементов массива, иначе – среднее арифметическое отрицательных элементов массива.

33. Если максимальный среди элементов с четными индексами больше максимального среди элементов с нечетными индексами, то заменить нулями элементы первой половины массива, иначе – элементы второй половины.

34. Если максимальный элемент массива больше суммы элементов массива, заменить его нулем, иначе – удвоить.

35. Если 1-й отрицательный элемент массива расположен до минимального, то найти сумму элементов с четными индексами, иначе – с нечетными индексами.

II уровень

Задание II уровня предназначено для приобретения навыков работы с матрицами на основе алгоритмов обработки одномерных массивов. Программу составить так, чтобы она могла быть использована для обработки массивов произвольного размера. Выполнить также все пункты задания I уровня.

1. Найти сумму элементов матрицы A размером 5×7 .

2. Найти среднее среди положительных элементов матрицы A размером 5×7 .

3. Найти след (сумму диагональных элементов) квадратной матрицы A размером 4×4 .

4. Определить номер строки и столбца, содержащих минимальный элемент матрицы A размером 3×6 .

5. Определить значение и номер первого отрицательного элемента заданного столбца матрицы A размером 5×4 .

6. Сформировать одномерный массив из индексов минимальных элементов строк матрицы A размером 4×7 .

7. Сформировать одномерный массив из значений максимальных элементов столбцов матрицы A размером 3×5 .

8. Сформировать одномерный массив из средних значений среди положительных элементов строк матрицы A размером 4×6 .

9. Поменять местами максимальный и 1-й элементы строк матрицы A размером 5×7 .

10. В матрице A размером 5×7 поменять местами строку, содержащую максимальный элемент в 3-м столбце, с 4-й строкой.

11. Удалить строку матрицы A размером 5×7 , содержащую минимальный элемент в 1-м столбце.

12. В матрице A размером 6×7 удалить столбец и строку, на пересечении которых находится максимальный элемент матрицы.

13. В матрице A размером 5×5 поменять местами 4-й столбец со столбцом, содержащим максимальный элемент на диагонали.

14. Сформировать одномерный массив из количеств отрицательных элементов столбцов матрицы A размером 4×3 .

15. Преобразовать матрицу A размером 5×7 , умножив максимальный элемент каждой строки на номер этой строки.

16. В каждой строке матрицы A размером $n \times m$ максимальный элемент поместить в конец строки, сохранив порядок остальных элементов.

17. В каждой строке матрицы B размером $n \times m$ минимальный элемент поместить в начало строки, сохранив порядок остальных элементов.

18. В каждой строке матрицы D размером $n \times m$ максимальный среди элементов, расположенных до первого отрицательного, поменять местами с последним отрицательным в этой строке.

19. В каждой строке матрицы C размером $n \times m$ все отрицательные элементы, расположенные перед максимальным, разделить на максимальный элемент.

20. В каждой строке матрицы F размером $n \times m$ максимальный элемент заменить на полусумму первого и последнего отрицательного в строке.

21. В матрице H размером 5×7 заполнены первые 6 столбцов. Поместить в качестве предпоследнего столбца столбец, состоящий из максимальных элементов строк.

22. В матрице Z размером 6×8 максимальный элемент матрицы заменить средним арифметическим положительных элементов матрицы.

23. В матрице G размером 5×7 заполнены первые 6 столбцов. В каждой строке продублировать максимальный элемент, расположив новый элемент, равный максимальному, сразу после максимального.

24. В матрице Y размером 6×5 в каждой строке заменить отрицательные элементы, расположенные перед максимальным, на среднее среди положительных элементов, расположенных после максимального.

25. В матрице X размером 6×5 поменять местами строки, содержащие минимальное и максимальное число отрицательных элементов.

26. В матрице A размером 5×7 строку, содержащую максимальный элемент в 6-м столбце, заменить заданным вектором B размером 7.

27. В матрице B размером 5×7 4-й столбец матрицы заменить одномерным массивом, состоящим из максимальных элементов строк, расположенных в обратном порядке (т.е. 1-й элемент 4-го столбца – это максимальный элемент 5-й строки и т.д.).

28. В матрице A размером 7×5 удалить строку с максимальной суммой положительных элементов строки.

29. В матрице F размером 5×7 удалить столбец, расположенный после столбца, содержащего минимальный по модулю элемент во 2-й строке.

30. Строку, содержащую максимальный элемент главной диагонали матрицы B размером 5×5 , поменять местами со строкой, содержащей 1-й (от начала столбца) отрицательный элемент в 3-м столбце.

31. В матрице A размером 5×8 заполнены первые 7 столбцов. Поместить вектор B размером 5 после столбца, содержащего минимальный элемент в 5-й строке.

32. В матрице A размером 5×7 в каждой строке заменить максимальный элемент средним среди положительных элементов строки.

33. Сформировать одномерный массив из отрицательных элементов матрицы A размером 5×7 .

34. Дана матрица A размером 5×7 . Для каждой строки сравнить элементы, расположенные непосредственно перед и после максимального элемента этой строки, и меньший из них увеличить в 2 раза. Если максимальный элемент является первым или последним в строке, то увеличить в 2 раза только один соседний с максимальным элемент.

35. Дана матрица A размером 7×5 . Если количество положительных элементов столбца больше количества отрицательных, то максимальный элемент этого столбца заменить на 0, в противном случае максимальный элемент заменить на номер максимального элемента этого столбца.

36. Дана матрица A размером 10×5 . Преобразовать матрицу следующим образом: заменить первый элемент столбца суммой элементов столбца, расположенных после максимального элемента, если максимальный элемент находится в первой половине столбца. В противном случае оставить столбец без изменения.

37. Дана матрица A размером 7×5 и массив B размером 5. Заменить максимальный элемент столбца соответствующим элементом массива B , если этот элемент больше найденного максимального элемента столбца. В противном случае замены не производить.

38. Дана матрица A размером 7×5 . Максимальный элемент столбца заменить полусуммой 1-го и последнего элементов столбца, если максимальный элемент меньше этой полусуммы, в противном случае максимальный элемент заменить его номером в столбце.

39. Сформировать матрицу размером $n \times 3n$, составленную из трех единичных квадратных матриц размером $n \times n$.

40. В матрице A размером 6×6 найти максимальный элемент на главной диагонали. Заменить нулями элементы матрицы, расположенные правее главной диагонали в строках, расположенных выше строки, содержащей максимальный элемент на главной диагонали.

41. В матрице B размером 6×6 поменять местами максимальные элементы 1-й и 2-й строк, 3-й и 4-й, 5-й и 6-й.

42. В матрице A размером 6×7 расположить элементы строк в обратном порядке.

III уровень

Задание III уровня предназначено для приобретения навыков решения задач с использованием типовых алгоритмов обработки массивов и их модификаций, а также некоторого творческого подхода. Выполнить также все пункты задания I уровня.

Программу составить в двух вариантах, представляя матрицу: а) в виде двухмерного массива; б) в виде одномерной последовательности.

1. Найти все максимальные элементы одномерного массива (предполагается, что в массиве несколько одинаковых максимальных элементов) за один проход исходного массива. Сформировать массив из их индексов.

2. В одномерном массиве увеличить максимальные элементы на их порядковые номера (1-й максимальный – на 1; 2-й – на 2; 3-й – на 3 и т.д.).

3. В одномерном массиве поменять местами соседние элементы (1-й со 2-м, 3-й с 4-м и т.д.), расположенные перед максимальным элементом массива.

4. Задан массив B . Максимальный элемент (или максимальные элементы, если их несколько) заменить суммой элементов массива, расположенных до него (до каждого из них, если их несколько).

5. Упорядочить по возрастанию элементы массива с четными индексами (остальные элементы оставить на своих местах).

6. В массиве A найти максимальное количество следующих подряд упорядоченных по убыванию элементов.

7. Все отрицательные элементы переставить в конец массива с сохранением порядка их следования.

8. Упорядочить по убыванию отрицательные элементы массива, сохраняя остальные элементы на прежних местах.

9. В заданном массиве определить длину самой большой упорядоченной (по возрастанию или по убыванию) последовательности.

10. В массиве, заполненном наполовину, продублировать все элементы с сохранением порядка следования. (Например, задан массив $X = [3, 8, \dots]$, получить массив $X = [3, 3, 8, 8, \dots]$).

11. Вычислить значения функции $y = \cos x + x \sin x$ в n точках отрезка $[a, b]$. Результат получить в двух массивах X (аргумент), Y (функция). Используя сформированные массивы, определить все значения аргумента, при которых функция имеет максимум или минимум (включая локальные). Для точек, в которых достигается экстремум, вывести значения аргумента, функции и вид экстремума (максимум или минимум). Определить глобальные экстремумы.

Указание. Функция имеет локальный максимум со значением y_i (из массива Y) при значении аргумента x_i (из массива X), если значения функции y_{i-1} и y_{i+1} в двух соседних точках меньше y_i . Аналогично для минимума.

12. Из массива размером 12 удалить все отрицательные элементы.

13. Из массива удалить повторяющиеся элементы.

14. Заданный массив преобразовать таким образом, чтобы все его элементы принадлежали отрезку $[-1; 1]$. Предусмотреть возможность обратного преобразования.

15. В матрице размером 7×5 переставить строки таким образом, чтобы минимальные элементы строк следовали в порядке убывания.

16. Заполнить нулями элементы квадратной матрицы, расположенные по ее периметру (использовать один цикл).

17. Для квадратной матрицы размером $n \times n$ просуммировать элементы, расположенные на диагоналях, параллельных главной, включая главную диагональ. Результат получить в виде вектора размером $2n - 1$.

18. Для матрицы размером $n \times n$ заполнить единицами нижнюю половину (включая среднюю строку, если n нечетное) за исключением элементов, расположенных справа от главной диагонали.

19. В матрице размером $n \times n$ найти максимальный по модулю элемент матрицы. Перестановкой строк и столбцов перевести максимальный по модулю элемент на пересечение k -й строки и k -го столбца ($1 \leq k \leq n$).

20. В матрице размером $n \times n$ сформировать два одномерных массива: в один переслать по строкам верхний треугольник матрицы, включая элементы главной диагонали, в другой –

нижний треугольник. Вывести верхний и нижний треугольники по строкам.

21. Перемножить две симметрические матрицы, заданные в одномерных массивах верхними треугольниками по строкам. Результат получить в одномерном массиве. Вывести в привычном виде исходные матрицы и матрицу-результат.

22. В матрице размером 7×5 переставить строки таким образом, чтобы количества положительных элементов в строках следовали в порядке убывания.

23. В матрице размером 5×7 переставить столбцы таким образом, чтобы количества отрицательных элементов в столбцах следовали в порядке возрастания.

24. В заданной матрице упорядочить элементы строк с четными индексами по убыванию, с нечетными – по возрастанию.

25. В заданной матрице удалить все строки, содержащие нулевые элементы.

26. Привести заданную квадратную матрицу a размером $n \times n$ к такому виду, чтобы все элементы ниже главной диагонали были нулевыми. Использовать линейные преобразования (умножение строки на число, сложение строк).

Указание. В цикле по номеру столбца ($j = 0, 1, \dots, n - 2$). В каждом столбце обнулить элементы с $(j + 1)$ – го до последнего, для этого из каждой k -й строки ($k = j + 1, \dots, n - 1$) вычесть j -ю строку, умноженную на коэффициент $p = a_{kj} / a_{jj}$; т.е. в цикле по номеру строки k ($k = j + 1, \dots, n - 1$) организовать цикл по номеру столбца m ($m = j, \dots, n - 1$), в котором выполнять $a_{km} = a_{km} - a_{jm} p$.

27. В задаче 26 преобразовать матрицу таким образом, чтобы все элементы выше главной диагонали были нулевыми.

28. В задаче 26 преобразовать матрицу таким образом, чтобы все элементы ниже и выше главной диагонали были нулевыми.

ГЛАВА 5. Функции

В упрощении разработки программ важную роль играет принцип *модульного программирования*. Суть его состоит в том, что программа конструируется из небольших частей, называемых модулями. Преимущества такого подхода:

- маленький фрагмент кода легче написать и отладить;
- модули можно использовать повторно в разных местах программы, это уменьшает объем кода и время разработки;
- модули можно объединить в библиотеки и предоставить возможность их использования другим программистам;
- в целом, модульное проектирование повышает качество кода.

В языке *Python* модули реализуются с помощью *функций*.

Функция – именованный фрагмент кода, который может быть вызван многократно.

Мы уже использовали стандартные функции (например, функции ввода / вывода `input()` и `print()`, математические функции `abs()`, `sin()` и т.д.).

Однако можно написать и свою функцию, а затем вызывать ее столько раз, сколько потребуется.

Функции позволяют существенно упростить жизнь программиста. С помощью функций мы можем структурировать наши программы, разбить ее на логически завершенные части. Каждый раз, когда в программе встречается часть, описывающая какое-то действие, мы можем выделить ее в отдельную функцию. В дальнейшем мы сможем эту функцию повторно использовать в других местах этой или другой программы. Даже если повторного использования не будет, нам будет проще понять нашу программу. С помощью функций мы можем создавать новые примитивы, операторы внутри нашей программы. При этом после написания функции можно абстрагироваться от того, как она реализована. Достаточно помнить, какие аргументы она принимает и что делает, а не держать полностью в голове алгоритм ее работы. Приведем пример – функция `sorted()`. Она принимает на вход последовательность, а возвращает отсортированный список. Это все, что нам необходимо знать, чтобы использовать ее в своей программе.

Общий синтаксис описания функции выглядит следующим образом.

1. Используется ключевое слово `def`, за ним следует название функции (может быть произвольным, действуют те же правила, что и для переменных).

2. В круглых скобках описываются параметры (аргументы) функции. После этого – двоеточие.

3. На последующих строках с отступом пишется тело функции.

4. Ключевое слово `return` завершает работу функции, возвращает значение, указанное после него, в то место программы, откуда была вызвана функция.

В качестве примера рассмотрим функцию, которая находит минимум среди двух чисел:

```
def min2(a, b):  
    if a <= b:  
        return a  
    else:  
        return b  
print(min2(42, 30))  
30
```

Для того чтобы вызвать функцию, нам нужно указать ее имя и в круглых скобках передать аргументы:

```
def min2(a, b):  
    if a <= b:  
        return a  
    else:  
        return b  
print(min2(min2(42, 30), 25))  
25
```

Функция `min2()` принимает на вход два параметра, поэтому в вызове ее мы передаем ей два целых числа: 42 и 30.

В качестве аргументов мы можем передавать как конкретные значения (числа, строки), так и переменные, произвольные выражения (например, $a * b$) или даже другие функции. После того как функция `min2()` будет вызвана, внутри нее в переменную `a` будет подставлено значение 42, а в переменную

b – значение 30. Далее эти значения будут использоваться внутри функции.

В качестве примера рассмотрим случай, когда в качестве аргумента передается результат вызова другой функции. Вначале сравним два числа 42 и 30. Результат сравнения передадим в функцию `min2()`. Это позволит, не изменяя тело функции, решить более сложную задачу, а именно – найти наименьшее из трех чисел:

```
def min2(a, b):
    if a <= b:
        return a
    else:
        return b
print(min2(min2(42, 30), 25))
25
```

Замечание: функция должна быть определена до ее первого вызова!

В рассмотренном выше примере мы имели дело с функцией, *возвращающей значение*. Но в *Python* используются и другие разновидности функций.

Функция может *не возвращать никакого значения*. Мы выделяем логически заверченный фрагмент программы и оформляем его в виде функции. Эта функция выполняется просто как некий набор действий. Например, это может быть функция, которая выводит справочное сообщение на экран. В таком случае функция может либо не содержать `return`, либо содержать пустой `return` (без параметра).

Также могут быть функции, которые *не принимают никаких параметров*. Если мы хотим создать функцию без параметров, то после имени функции ставятся пустые круглые скобки, например:

```
def foo():
    print("Функция без параметров")
foo()
Функция без параметров
```

Функция может также *иметь произвольное число параметров*. Мы уже сталкивались с такими функциями, например

функция `print()`. Мы можем передать ей один параметр, и она выведет его значение. Точно также можем передать ей два, три или вообще произвольное количество параметров, например:

```
a = 'первый параметр'
b = 'второй параметр'
c = 'третий параметр'
print(a)
print(a, b, c)
первый параметр
первый параметр второй параметр третий параметр
```

При задании собственной (пользовательской) функции с произвольным количеством параметров используется следующая реализация.

1. В заголовке функции (в скобках, следующих за именем функции) мы вместо перечисления аргументов записываем конструкцию **a*. Это специальный способ указать, что наша функция принимает на вход произвольное число аргументов.

2. Все аргументы, переданные функции, будут храниться в последовательности с именем *a*. К этой последовательности мы можем обращаться с помощью индексов или итерироваться по ней с помощью цикла `for`. После завершения цикла вернем *m* в качестве результата функции.

Данную функцию мы можем вызывать с произвольным количеством аргументов:

```
def min(*a):
    m=a[0]
    for x in a:
        if m > x:
            m = x
    return m
print(min(5))
print(min(5, 3))
print(min(5, 3, 6, 10))
```

Функции могут иметь *значение параметров по умолчанию*. С такими функциями мы уже встречались.

Например, функция `print()`, которая имеет параметр `end`. В случае, когда мы не указываем его при вызове, по заверше-

нии вывода функция `print()` переводит строку. Если же мы вызовем функцию, передав значение `end`, равное пустой строке, курсор останется в строке вывода.

```
for i in range(3):
    print(i)
print()
for i in range(3):
    print(i, end = '')
0
1
2
012
```

Еще один пример – функция `range()`, которая позволяет генерировать последовательность чисел. При этом значение, которое мы указываем в скобках, обозначает количество итераций цикла.

```
for i in range(3):
    print(i, end = ' ')
0 1 2
```

Если мы хотим начинать последовательность не с 0, а с какого-то другого числа, то мы можем указывать в скобках два значения.

Например, вызов функции `range(1, 5)` сгенерирует последовательность чисел 1, 2, 3, 4.

Таким образом, передавая один или два параметра в функцию `range()`, мы можем генерировать любую последовательность целых чисел с шагом 1.

Для изменения шага последовательности необходимо передать в функцию `range()` три параметра.

Например, при вызове функции `range(1, 10, 2)` создаст последовательность чисел 1, 3, 5, 7, 9, а вызов функции `range(5, 30, 5)` сгенерирует последовательность 5, 10, 15, 20, 25.

При создании собственной функции значения аргументов по умолчанию задаются явно при перечислении аргументов в заголовке функции:

```
def my_function(start, stop, step = 1)
```

В данном примере параметром по умолчанию является параметр `step`.

```
def my_function(start, stop, step = 1):
    res = []
    if step > 0:
        x = start
        while x < stop:
            res+= [x]
            x+= step
    elif step < 0:
        x = start
        while x > stop:
            res+= [x]
            x+= step
    return res
```

В случае, когда значение параметра `step` явно не указано при вызове функции, он принимает значение 1:

```
print(my_function(2, 5))
[2, 3, 4]
```

В противном случае `step` принимает то числовое значение, которое стоит в списке аргументов на третьем месте:

```
print(my_function(2, 5, 3))
print(my_function(12, 5, - 3))
[2]
[12, 9, 6]
```

В этом примере передача параметра осуществляется по позиции в списке аргументов. При вызове функции мы также можем *явно* указывать, какое значение имеет та или иная переменная внутри функции, например:

```
print(my_function(stop = 15, start = 10))
print(my_function(start = 10, stop = 15))
[10,11,12,13,14]
[10,11,12,13,14]
```

Как видно из примера, при явном указании значений параметров порядок их перечисления не важен.

Переменные, которые объявлены внутри функции, называются *локальными*. Их невозможно использовать вне функции. Например, если у нас есть функция `init_values()`, в которой мы инициализируем значения переменных *a* и *b*, то мы можем использовать эти значения переменных только внутри функции. Снаружи функции (даже после вызова этой функции) значения переменных *a* и *b* считаются неизвестными. Если мы попробуем вывести значения *a* и *b*, мы увидим сообщение об ошибке:

```
def init_values():
    a = 8
    b = -6
init_values()
print(a + b)
NameError: name 'a' is not defined
```

Рассмотрим случай, когда в нашей программе есть две переменных с одинаковым именем *a*, одна инициализирована вне определения функции, вторая – внутри:

```
a=0
def init_values():
    a = 8
    b = -6
init_values()
print('значение a =', a)
значение a = 0
```

Переменные, которые объявлены вне всяких функции, называются *глобальными*. Их можно использовать во всей программе. Например, у нас есть функция `print_value()`, внутри которой мы никак не инициализировали значение переменной *a*, но, тем не менее, используем его для вывода. Такой код будет работать, если перед вызовом функции мы инициализируем глобальную переменную *a*.

```
a = 5
def init_values():
    print('значение a =', a)
init_values()
значение a = 5
```

При вызове функции компилятор, обнаружив переменную *a*, которая не связана ни с каким значением внутри функции, будет смотреть глобальный контекст: если переменная *a* определена глобально, он подставит ее значение в функцию, в противном случае сообщит об ошибке.

```
def init_values():  
    print('значение a =', a)  
init_values()  
NameError: name 'a' is not defined
```

Если переменную *a* внутри функции сначала использовать как глобальную, а потом изменить ее и попытаться вывести новое значение, также получим сообщение об ошибке, но уже другого содержания:

```
a = 5  
def init_values():  
    print('a =', a)  
    a=10  
    print('a =', a)  
init_values()  
UnboundLocalError: local variable 'a' referenced before  
assignment
```

Для того чтобы разобраться в причине ошибки, надо понимать, что происходит при вызове функции.

Если внутри функции мы пытаемся изменить значение переменной (присвоить ей другое значение), то в таком случае переменная считается локальной внутри всей функции. Поскольку переменная считается локальной, при первом вызове функции `print()` поиск по глобальным переменным не будет происходить, а локальная переменная *a* к этому моменту еще не инициализирована. Сообщение об ошибке будет выведено, даже несмотря на то, что в глобальной области определена переменная с именем *a*.

Вопросы для самопроверки

1. В чем заключается принцип модульного программирования?

2. Что такое функция? Как выглядит общий синтаксис описания функции?
3. Каковы разновидности функций по характеру возвращаемого значения? В чем особенности их оформления?
4. Как осуществляется вызов функции? В чем специфика обращения к функции в зависимости от количества ее параметров: функция без параметров, с фиксированным количеством параметров, с произвольным количеством параметров?
5. В чем заключаются особенности задания собственных функций?
6. Как происходит использование параметров по умолчанию?
7. Что такое локальные и глобальные переменные? В чем особенности работы с указанными видами переменных?

Задания для самостоятельного выполнения

Задание предназначено для приобретения начальных навыков создания функций. Необходимо сформулировать, если нужно, задачу математически; определить входные параметры функции и возвращаемое значение; написать функцию таким образом, чтобы не портились входные параметры; составить программу; разработать тесты и проверить работу программы на компьютере.

1. Напишите программу, которая обеспечивает ввод пользователем двух чисел и выполняет суммирование всех чисел в указанном диапазоне, включая граничные значения. Если пользователь задаст первое число больше, чем второе, программа должна поменять их местами. Процесс суммирования реализуйте с использованием функции.

2. Напишите программу, на вход которой подается три числа. Результатом работы программы является число, максимальное из трех введенных. Поиск максимума реализуйте с использованием функции.

3. Напишите программу, на вход которой подается список чисел. Результат работы программы – произведение всех чисел в списке. Вычисление произведения реализуйте с использованием функции.

4. Напишите программу, на вход которой подается натуральное число. Результатом работы программы является значение факториала этого числа. Вычисление факториала реализуйте с использованием функции, которая принимает введенное число в качестве аргумента.

5. Напишите программу, на вход которой подается три числа, два из которых упорядочены по возрастанию и имеют смысл границ числового промежутка. Результатом работы программы является ответ на вопрос, находится ли третье введенное число в заданном диапазоне. Проверку реализуйте с использованием функции.

6. Напишите программу, на вход которой подается список. Результатом работы программы является новый список с уникальными элементами первого списка. Формирование нового списка реализуйте с использованием функции, которая принимает исходный список в качестве аргумента.

7. Напишите программу, на вход которой подается натуральное число. Результатом работы программы является ответ на вопрос, является введенное число простым или нет. Процесс отнесения введенного числа к определенной группе реализуйте с использованием функции, которая принимает это число в качестве аргумента.

8. Напишите программу, на вход которой подается список чисел. Результатом работы программы является новый список, содержащий все четные числа из исходного списка. Поиск четных чисел реализуйте с использованием функции.

9. Напишите программу, на вход которой подается натуральное число. В результате работы программа печатает первые n строк треугольника Паскаля. Вычисление значений для каждой строки реализуйте с использованием функции.

Указание. Треугольник Паскаля (арифметический треугольник) – бесконечная таблица биномиальных коэффициентов, имеющая треугольную форму. В этом треугольнике на вершине и по бокам стоят единицы. Каждое число равно сумме двух расположенных над ним чисел.

10. Напишите программу, которая находит полное число метров по заданному числу сантиметров. Вычисления организовать с использованием функции.

11. Напишите программу для пересчета величины временного интервала, заданного в минутах, в величину, выраженную в часах и минутах. Вычисления организовать с использованием функции.

12. Напишите программу, в которой рассчитывается сумма и произведение цифр положительного трехзначного числа. Вычисления организовать с использованием функции.

13. Напишите программу для нахождения цифр четырехзначного числа. Вычисления организовать с использованием функции.

14. Напишите программу, которая определяет, является число четным или нечетным. Процесс отнесения введенного числа к определенной группе реализуйте с использованием функции, которая принимает это число в качестве аргумента.

15. Напишите программу, которая определяет, являются ли три заданных числа (в указанном порядке) последовательными членами арифметической прогрессии. Процесс отнесения введенных чисел к определенной группе реализуйте с использованием функции, которая принимает эти числа в качестве аргументов.

16. На вход программе подается четыре числа. В результате работы программа печатает эти числа по возрастанию. Поиск минимального значения реализовать с использованием функции.

17. Напишите программу, которая считывает три числа и вычисляет сумму только положительных чисел. Суммирование реализовать с использованием функции.

18. Напишите программу, которая определяет, является ли год с данным номером високосным. Если год является високосным, то выведите «YES», иначе выведите «NO». Вычисления организовать с использованием функции.

Указание. Год является високосным, если его номер кратен 4, но не кратен 100, или если он кратен 400.

19. Напишите программу, которая принимает три положительных числа и определяет вид треугольника, длины сторон которого равны введенным числам. Вычисления организовать с использованием функции.

20. Красный, синий и желтый называются основными цветами, потому что их нельзя получить путем смешения других

цветов. При смешивании двух основных цветов получается вторичный цвет:

- если смешать красный и синий, то получится фиолетовый;
- если смешать красный и желтый, то получится оранжевый;
- если смешать синий и желтый, то получится зеленый.

Напишите программу, которая считывает названия двух основных цветов для смешивания. Если пользователь вводит что-нибудь, помимо названий «красный», «синий» или «желтый», то программа должна вывести сообщение об ошибке. В противном случае программа должна вывести название вторичного цвета, который получится в результате. Процесс определения результирующего цвета организовать с использованием функции.

20. Напишите программу, которая принимает на вход два отрезка: $[a1; b1]$ и $[a2; b2]$. В результате работы программа печатает результат – пересечение этих отрезков. Пересечением двух отрезков может быть:

- отрезок;
- точка;
- пустое множество.

Вычисления организовать с использованием функции.

21. Определить, сколькими способами можно отобрать команду в составе 5 человек из 8 кандидатов; из 10 кандидатов; из 11 кандидатов. Использовать метод для подсчета количества способов отбора по формуле

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

22. Два треугольника заданы длинами своих сторон a , b и c . Определить треугольник с большей площадью, вычисляя площади треугольников по формуле Герона

$$S = \sqrt{p(p-a)(p-b)(p-c)}.$$

где $p = (a + b + c) / 2$.

23. Два велосипедиста одновременно начинают движение из одной точки. Первый начинает движение со скоростью 10 км/ч

и равномерно увеличивает скорость на 1 км/ч. Второй начинает движение со скоростью 9 км/ч и равномерно увеличивает скорость на 1,6 км/ч. Определить:

- а) какой спортсмен преодолеет большее расстояние через 1 ч, через 4 ч;
- б) когда второй спортсмен догонит первого.

Использовать функцию для вычисления пути в зависимости от времени по формуле

$$S = vt + at^2 / 2,$$

где v – начальная скорость;

a – ускорение.

ЗАКЛЮЧЕНИЕ

В результате изучения материалов учебника студенты освоят базовые приемы и методы алгоритмизации задач, узнают, какие существуют типовые структуры алгоритмов и какие средства имеются в языке *Python* для их программной реализации; как типовые структуры могут соединяться в программе в рамках структурного подхода; как осуществляется ввод данных в переменные программы и вывод результатов на экран; как выбрать подходящий способ представления данных для решаемой задачи. Также познакомятся с основными типовыми алгоритмами обработки массивов, научатся модифицировать эти алгоритмы и использовать их в сочетании для создания эффективных программ.

Содержание учебника создает необходимый фундамент для понимания основ и принципов программирования и возможность дальнейшего освоения языка для решения более сложных задач.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. A Byte of Python (Russian). Версия 2.01 Swaroop С Н / Translated by Vladimir Smolyar.
2. Широков А.И., Пышняк М.О. Информатика : Основы разработки программ на языке программирования Питон : Часть 1 : учебник. М. : Изд. Дом НИТУ «МИСиС», 2020. 146 с.
3. Самоучитель Python – <https://pythonworld.ru/samo-uchitel-python>
4. Справочник по языку Python – <https://docs.python.org/3/reference/index.html>
5. Стандартная библиотека Python – <https://docs.python.org/3/library/index.html>
6. Список документации Python – <https://docs.python.org/3/>

ПРИЛОЖЕНИЕ.

Основные операторы

Таблица П.1

Арифметические операторы

Символ	Описание	Пример
+	Оператор суммы	<code>print(5 + 8)</code> 13
-	Оператор вычитания	<code>print(31 - 2)</code> 29
*	Оператор произведения	<code>print(12 * 9)</code> 108
/	Оператор деления	<code>print(6 / 4)</code> 1.5
%	Оператор получения остатка от деления	<code>print(6 % 4)</code> 2
**	Оператор возведения в степень	<code>print(9 ** 2)</code> 81
//	Оператор целочисленного деления	<code>print(6 // 4)</code> 1

Таблица П.2

Операторы сравнения (реляционные)

Символ	Описание	Пример
==	Проверяет, равны ли операнды между собой. Если они равны, то выражение становится истинным	<code>print(5 == 5)</code> True <code>print(6 == 44)</code> False
!=	Проверяет, не равны ли операнды между собой. Если они не равны, то выражение становится истинным	<code>print(12 != 12)</code> False <code>print(1231 != 0.4)</code> True
>	Проверяет, больше ли левый операнд, чем правый. Если больше, то выражение становится истинным	<code>print(53 > 23)</code> True <code>print(432 > 500)</code> False
<	Проверяет, меньше ли левый операнд, чем правый. Если меньше, то выражение становится истинным	<code>print(5 < 51)</code> True <code>print(6 < 4)</code> False

Окончание табл. П.2

Символ	Описание	Пример
<>	Проверяет, не равны ли операнды между собой. Если они не равны, то выражение становится истинным	print(132 <> 11) True print(44 <> 44) False
>=	Проверяет, больше (или равен) левый операнд, чем правый. Если больше, то выражение становится истинным	print(5 >= 5) True print(6 >= 44) False
<=	Проверяет меньше (или равен) левый операнд, чем правый. Если меньше, то выражение становится истинным	print(32 <= 232) True print(65 <= 9) False

Таблица П.3**Операторы присваивания**

Символ	Описание	Пример
=	Присваивает значение правого операнда левому	var = 5 print(var) 5
+=	Прибавляет значение правого операнда к левому и присваивает левому. a+=b эквивалентно записи a=a+b	var = 5 var += 4 print(var) 9
-=	Отнимает значение правого операнда у левого и присваивает левому. a-=b эквивалентно записи a=a-b	var = 5 var -= 2 print(var) 3
=	Умножает значение левого операнда на правое и присваивает левому. a=b эквивалентно записи a=a*b	var = 5 var *= 10 print(var) 50
/=	Делит значение левого операнда на правое и присваивает левому. a/=b эквивалентно записи a=a/b	var = 5 var /= 4 print(var) 1.25
%=	Вычисляет остаток от деления значения левого операнда на правое и присваивает левому. a%=b эквивалентно записи a=a%b	var = 5 var %= 10 print(var) 5

Окончание табл. П.3

Символ	Описание	Пример
=	Возводит значение левого операнда в степень правого и присваивает левому. $a=b$ эквивалентно записи $a=a**b$	<pre>var = 5 var **= 8 print(var) 390625</pre>
//=	Целочисленно делит значение левого операнда на правое и присваивает левому. $a//=b$ эквивалентно записи $a=a//b$	<pre>var = 5 var //= 30 print(var) 0</pre>

Таблица П.4

Побитовые операторы*

Символ	Описание	Пример (в двоичной системе)
&	Бинарный «И» оператор, копирует бит в результат, только если бит присутствует в обоих операндах	<pre>0&0=0 1&0=0 0&1=0 1&1=1 101 & 011 = 001</pre>
 	Бинарный «ИЛИ» оператор копирует бит, если тот присутствует в хотя бы в одном операнде	<pre>0 0=0 1 0=1 0 1=1 1 1=1 101 011 = 111</pre>
^	Бинарный «Исключительное ИЛИ» оператор копирует бит, только если бит присутствует в одном из операндов, но не в обоих сразу	<pre>0^0=0 1^0=1 0^1=1 1^1=0 101 ^ 011 = 110</pre>
~	Побитовая операция «НЕ». Для числа a соответствует $-(a + 1)$	<pre>~1 = -2 ~0 = -1 ~101 = -110</pre>
>>	Побитовый сдвиг вправо. Значение левого операнда «сдвигается» вправо на количество бит, указанных в правом операнде	<pre>100 >> 2 = 001</pre>
<<	Побитовый сдвиг влево. Значение левого операнда «сдвигается» влево на количество бит, указанных в правом операнде	<pre>100 << 2 = 10000</pre>

***Примечание.** Данные операторы работают с данными в двоичной системе числения. Например, число 13 в двоичной системе будет равно 1101.

Таблица П.5

Логические операторы

Символ	Описание	Пример
and	Логический оператор «И». Условие будет истинным, если оба операнда истина	True and True = True True and False = False False and True = False False and False = False
or	Логический оператор «ИЛИ». Если хотя бы один из операндов истинный, то и все выражение будет истинным	True or True = True True or False = True False or True = True False or False = False
not	Логический оператор «НЕ». Изменяет логическое значение операнда на противоположное	not True = False not False = True

Таблица П.6

Операторы членства*

Символ	Описание	Пример
in	Возвращает истину, если элемент присутствует в последовательности, иначе возвращает ложь	print('he' in 'hello') True print(5 in [1,2,3,4,5]) True print(12 in [1,2,4,5,6]) False
not in	Возвращает истину если элемента нет в последовательности	Результаты противоположны примерам выше

***Примечание.** Данные операторы участвуют в поиске данных в некоторой последовательности.

Таблица П.7

Операторы тождественности*

Символ	Описание	Пример
is	Возвращает истину, если оба операнда указывают на один объект	a = 12 b = 12 a is b True c = 22 a is c False

Окончание табл. П.7

Символ	Описание	Пример
is not	Возвращает ложь, если оба операнда указывают на один объект	результаты противо- положны примерам выше

**Примечание.* Данные операторы помогают сравнить размещение двух объектов в памяти компьютера.

Таблица П.8

Приоритет операторов

or	and	not	in, not in	is, not is	<, <=, >, >=, !=, ==		^	&	<<, >>	+, *	*, /, //, %	~	**
Логические операторы	Операторы членства	Операторы тождественности	Операторы сравнения	Побитовые операторы	Арифметические операторы	Побитовый оператор	Арифметический оператор						

Наименьший приоритет

Наибольший приоритет

Учебное издание

Андреева Ольга Владимировна
Ремизова Ольга Игоревна

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

Учебник

Научный редактор *Т.А. Кравченко*
Корректор *Е.Н. Леонова*
Верстальщик *А.М. Маркин*

Подписано в печать 26.05.22 Уч.-изд. л. 9,31

Формат 60 × 90¹/₁₆

Национальный исследовательский
технологический университет «МИСиС»,
119049, Москва, Ленинский пр-т, 4А

Издательский Дом НИТУ «МИСиС»,
119049, Москва, Ленинский пр-т, 2А
Тел. 8 (495) 638-44-06

Отпечатано в типографии
Издательского Дома НИТУ «МИСиС»,
119049, Москва, Ленинский пр-т, 4А
Тел. 8 (495) 638-44-16, 8 (495) 638-44-43