

А. Н. Маслобоев

**ЯЗЫКИ И МЕТОДЫ
ПРОГРАММИРОВАНИЯ**

**ОСНОВЫ ПРОГРАММИРОВАНИЯ
НА ЯЗЫКЕ PYTHON**

Учебное пособие

**Санкт-Петербург
2024**

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«Санкт-Петербургский государственный университет
промышленных технологий и дизайна»
Высшая школа технологии и энергетики**

А. Н. Маслобоев

**ЯЗЫКИ И МЕТОДЫ
ПРОГРАММИРОВАНИЯ**

**ОСНОВЫ ПРОГРАММИРОВАНИЯ
НА ЯЗЫКЕ PYTHON**

Учебное пособие

Утверждено Редакционно-издательским советом ВШТЭ СПбГУПТД

Санкт-Петербург
2024

УДК 681.3 (075)
ББК 22.18я7
М 316

Рецензенты:

доктор технических наук, профессор, заведующий кафедрой радиотехники и информационных технологий Санкт-Петербургского государственного университета кино и телевидения,

В. М. Пестриков;

кандидат технических наук, профессор, заведующий кафедрой информационно-измерительных технологий и систем управления Высшей школы технологии и энергетики Санкт-Петербургского государственного университета промышленных технологий и дизайна

В. И. Сидельников

Маслобоев, А. Н.

М 316 Языки и методы программирования. Основы программирования на языке Python: учеб. пособие / А. Н. Маслобоев. — СПб.: ВШТЭ СПбГУПТД, 2024. — 62 с.

ISBN 978-5-91646-388-0

Учебное пособие соответствует программам и учебным планам дисциплины «Языки и методы программирования» для студентов, обучающихся по направлению подготовки 01.03.02 «Прикладная математика и информатика». В учебном пособии рассматриваются основы высокоуровневого языка программирования общего назначения Python, а также приводятся необходимые теоретические сведения по языку Python и примеры программ на данном языке. Рассматривается создание вычислительных программ линейной структуры, применение управляющих конструкций и описание и вызов простых функций. С целью лучшего усвоения материала студентам предлагаются контрольные вопросы.

Пособие предназначено для подготовки бакалавров очной и формы обучения.

УДК 681.3 (075)
ББК 22.18я7

ISBN 978-5-91646-388-0

© ВШТЭ СПбГУПТД, 2024
© Маслобоев А. Н., 2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
ГЛАВА 1. ЛИНЕЙНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОГРАММЫ.....	9
Контрольные вопросы.....	19
ГЛАВА 2. ОПЕРАТОРЫ ВЫБОРА.....	20
Контрольные вопросы.....	31
ГЛАВА 3. ОПЕРАТОРЫ ЦИКЛА	32
Контрольные вопросы.....	43
ГЛАВА 4. ФУНКЦИИ	44
Контрольные вопросы.....	51
ГЛАВА 5. СПИСКИ.....	53
Контрольные вопросы.....	59
ЗАКЛЮЧЕНИЕ	60
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	61

ВВЕДЕНИЕ

В XXI веке информационные технологии проникли во все сферы человеческой деятельности: от гуманитарных наук до управления полетами космических кораблей. Неотъемлемой частью и необходимым условием этого процесса является всеобщая компьютеризация. При этом широкое применение компьютеров подразумевает наличие соответствующего программного обеспечения, без которого даже самая совершенная техника, созданная человеческим разумом и трудом, остается только грудой пластика и металла. Для разработки программного обеспечения необходимо наличие специалистов в области информатики и информационных технологий, обладающих необходимыми знаниями и практическими навыками. Одним из перспективных направлений в этом отношении является изучение высокоуровневого языка программирования общего назначения Python.

Несмотря на то, что язык Python изначально создавался не для учебных целей, он нашел свою нишу в сфере обучения студентов, в особенности будущих специалистов по информатике, основам программирования. Этот язык, получивший название в честь популярного телешоу, задумывался его разработчиком нидерландским программистом Гвидо Ван Россумом в первую очередь как сценарный язык. Но Python «перерос» свои исходные рамки и стал применяться для многих других целей. Сейчас Python используется в таких областях информационных технологий, как сетевое программирование, приложения для работы с электронной почтой, приложения для обслуживания баз данных, программы для научных расчетов, средства поддержки технологий HTML и XML, приложения с графическим интерфейсом и в целом ряде других. Также данный язык можно с успехом применять не только для решения вышеперечисленных профессиональных задач, но и как удобное и доступное средство для обучения студентов современным технологиям программирования.

Простота синтаксиса, интуитивная понятность и наглядность данного языка позволила использовать его в качестве первого языка для обучения основам программирования, и, в то же время, овладев этими основами, обучающийся получает возможность далее совершенствоваться в этой сфере, используя уже знакомый ему синтаксис Python.

Оригинальной и крайне полезной чертой этого языка является то, что, он с самого начала приучает разработчика к определенным принципам, составляющим основу культуры программирования. Так, для создания достаточно хорошо структурируемого и читаемого исходного кода программы существуют общепринятые рекомендации о необходимости использования системы отступов в тексте. Но зачастую эти рекомендации так и остаются пожеланиями, поскольку отсутствие данного элемента, то есть отступов, на синтаксическую правильность программы не влияет, а текст этой программы становится неряшливым и трудно читаемым. Однако в языке Python отступы – это не просто элементы оформления. Они выполняют существенную синтаксическую роль, выделяя в тексте программы такие блоки как тело

функции, тело цикла, различные ветви условного оператора и ряд других. Таким образом, приступая к изучению основ программирования, будущий специалист сразу приучается не только к соблюдению синтаксических правил языка, но и к правильному и грамотному оформлению исходного кода, так как в противном случае программа на языке Python просто не будет работать.

Следует отметить еще одну полезную черту языка Python (хотя в данном случае это относится скорее к популярной среде разработки и обучения на данном языке, которая называется Idle Python). Эта черта заключается в том, что разработчик программы не имеет возможность запустить ее на выполнение, не сохранив исходный код программы в виде файла, хранящегося под определенным именем на жестком диске компьютера. Начинающим программистам можно достаточно долго и, к сожалению, зачастую безрезультатно объяснять, что перед запуском готовой программы на выполнение следует сохранить ее на каком-либо носителе, а не оставлять только в оперативной памяти компьютера. Это необходимо, поскольку любой сбой в работе компьютерной системы, который может быть вызван как неквалифицированными действиями самого пользователя, так и не зависящими от него техногенными факторами, приведет к потере исходного кода, который придется набирать заново. Конечно, особенно досадно, когда такая ситуация происходит не с программой сложения трех чисел, а с достаточно большим и серьезным приложением, потребовавшим многих часов упорного труда. При работе же с Idle Python подобная возможность исключается с самого начала, поскольку попытка запустить не сохраненный нигде исходный код сразу же блокируется этой средой.

Важным преимуществом данного языка с учетом современных тенденций в развитии информационных технологий является то, что Python с самого начала создавался как язык, в котором инкорпорирована парадигма объектно-ориентированного программирования. Такие понятия как класс, объекты и экземпляры классов, конструктор и деструктор экземпляра класса, поля и методы класса, наследование, перегрузка операторов, декораторы являются органической частью языка и поэтому в достаточной степени легко осваиваются обучающимися.

Безусловно, подобного рода конструкции есть и в других языках программирования, например, в языках C++ и C#. Но для того, чтобы освоить все возможности языков семейства C, надо, как и в любом другом языке, сперва освоить его азы, а это может вызывать определенные затруднения. Уже при написании первых самых простых программ обучающимся приходится сталкиваться с такими понятиями, как «препроцессор», «пространство имен» и других, понимание и корректное использование которых на этом, начальном этапе может быть достаточно сложным.

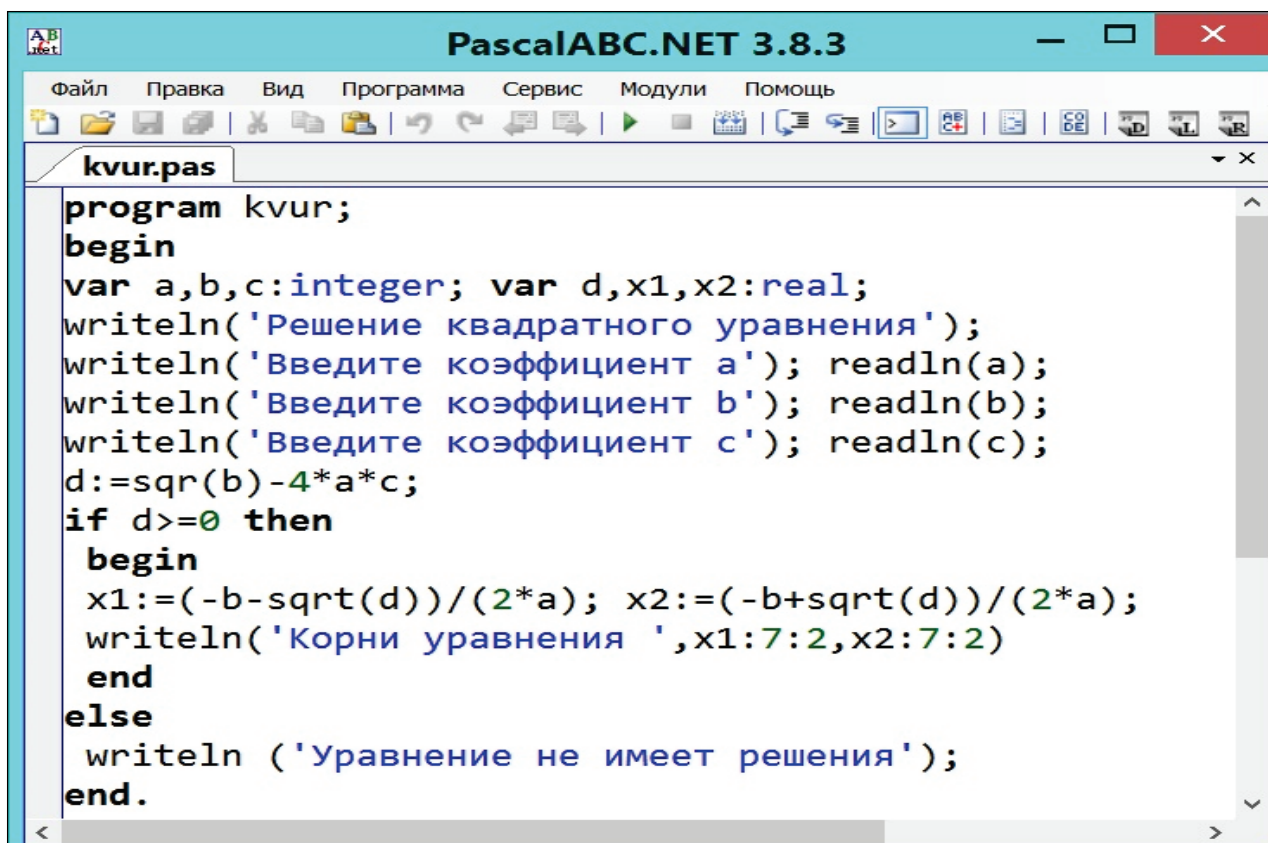
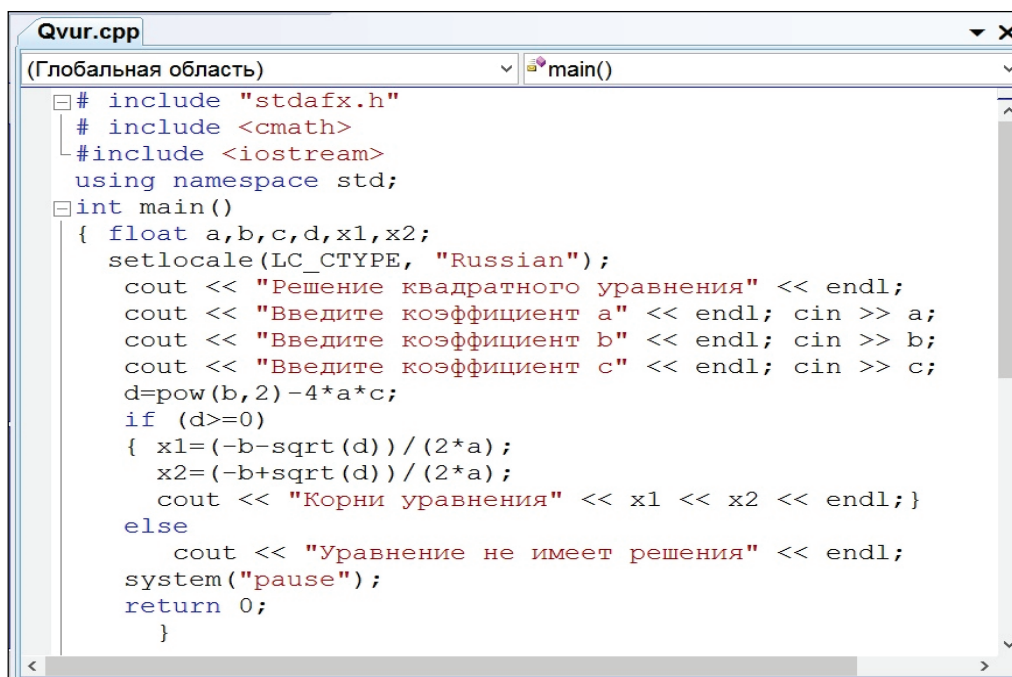


Рисунок В.1 – Программа решения квадратного уравнения на языке Pascal ABC

Что же касается языка Python, то с ним таких проблем, как правило, не возникает, поскольку пользователь практически сразу может начать написание небольших, несложных программ, а уже от них двигаться к вершинам программирования. Для того, чтобы наглядно оценить сравнительную простоту и сложность синтаксиса различных языков программирования здесь приведены тексты аналогичных по содержанию программ, созданных на разных языках программирования. На рисунке В.1 приведена программа решения квадратного уравнения, написанная на языке Паскаль (использована версия языка Pascal.ABC, созданная специалистами института математики, механики и точных наук Южного Федерального Университета).

На рисунке В.2 показан пример решения той же задачи на языке C++ (использована версия языка, применяемая в программном пакете Microsoft Visual Studio, разработанного фирмой Microsoft).

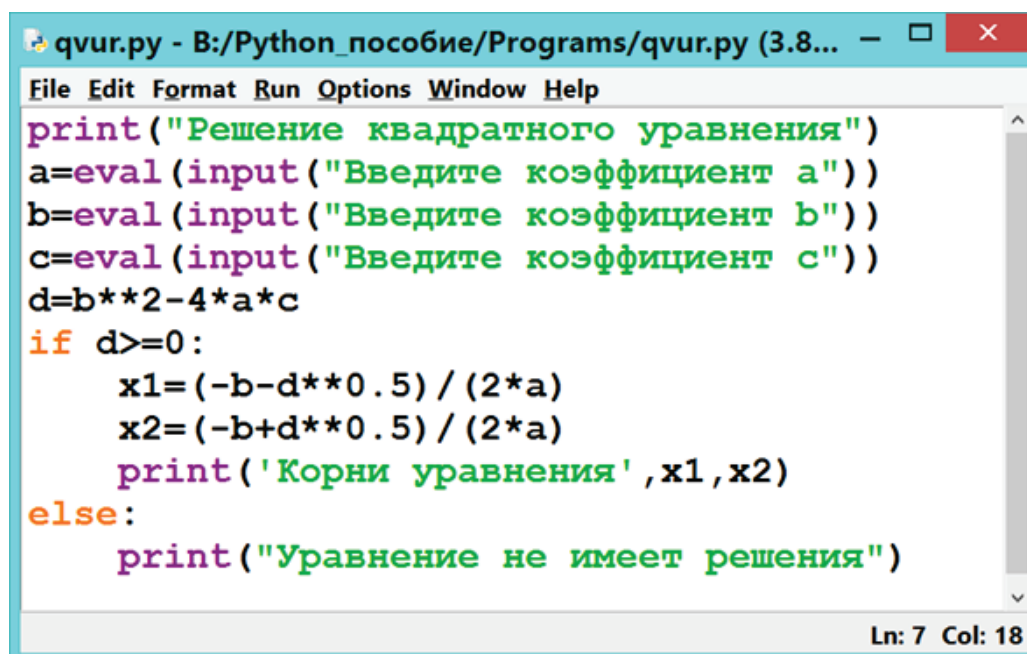


```
#include "stdafx.h"
#include <cmath>
#include <iostream>
using namespace std;

int main()
{
    float a, b, c, d, x1, x2;
    setlocale(LC_CTYPE, "Russian");
    cout << "Решение квадратного уравнения" << endl;
    cout << "Введите коэффициент a" << endl; cin >> a;
    cout << "Введите коэффициент b" << endl; cin >> b;
    cout << "Введите коэффициент c" << endl; cin >> c;
    d = pow(b, 2) - 4 * a * c;
    if (d >= 0)
    {
        x1 = (-b - sqrt(d)) / (2 * a);
        x2 = (-b + sqrt(d)) / (2 * a);
        cout << "Корни уравнения" << x1 << x2 << endl;
    }
    else
    {
        cout << "Уравнение не имеет решения" << endl;
        system("pause");
        return 0;
    }
}
```

Рисунок В.2 – Программа решения квадратного уравнения на языке C++

Наконец на рисунке В.3 продемонстрировано решение все той же задачи, выполненное на языке Python в среде IDLE. Даже чисто визуально можно оценить уровень сложности решения однотипной задачи на разных языках и в разных средах программирования. Наиболее “объемной” является программа на языке C++, решение задачи на Паскале по сложности занимает промежуточное положение, а наименьший объем исходного программного кода имеет решение задачи на языке Python.



```
print("Решение квадратного уравнения")
a=eval(input("Введите коэффициент a"))
b=eval(input("Введите коэффициент b"))
c=eval(input("Введите коэффициент c"))
d=b**2-4*a*c
if d>=0:
    x1=(-b-d**0.5)/(2*a)
    x2=(-b+d**0.5)/(2*a)
    print('Корни уравнения', x1, x2)
else:
    print("Уравнение не имеет решения")
```

Рисунок В.3 – Программа решения квадратного уравнения на языке Python

Необходимо отметить, что максимальная простота синтаксиса языка Python имеет и свою обратную сторону. Так, отсутствие в этом языке требования об обязательном предварительном описании типов переменных, используемых в программе, создает определенные трудности для обучающихся при переходе к программированию на других языках (языки семейства C, Java). Использование отступов приучает к правильному стилю написания программного кода, но при этом является единственным способом выделения блочных элементов в программе (в Python нельзя использовать ключевые слова, как в Паскале или фигурные скобки как в языках семейства C).

Но в целом можно сказать, что достоинства языка Python перевешивают его недостатки и данный язык можно признать вполне достойным средством для обучения основам программирования студентов, специализирующихся в области прикладной математики и информатики.

ГЛАВА 1. ЛИНЕЙНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОГРАММЫ

Естественно, для того, чтобы пользователь мог начать работу с языком программирования, ему сначала необходимо установить на свой компьютер среду программирования, поддерживающую этот язык. Python является свободно распространяемым программным продуктом, и самый простой, надежный и безопасный способ его установки – это загрузка его дистрибутива с официального сайта разработчиков, который находится по адресу [python.org](https://www.python.org).

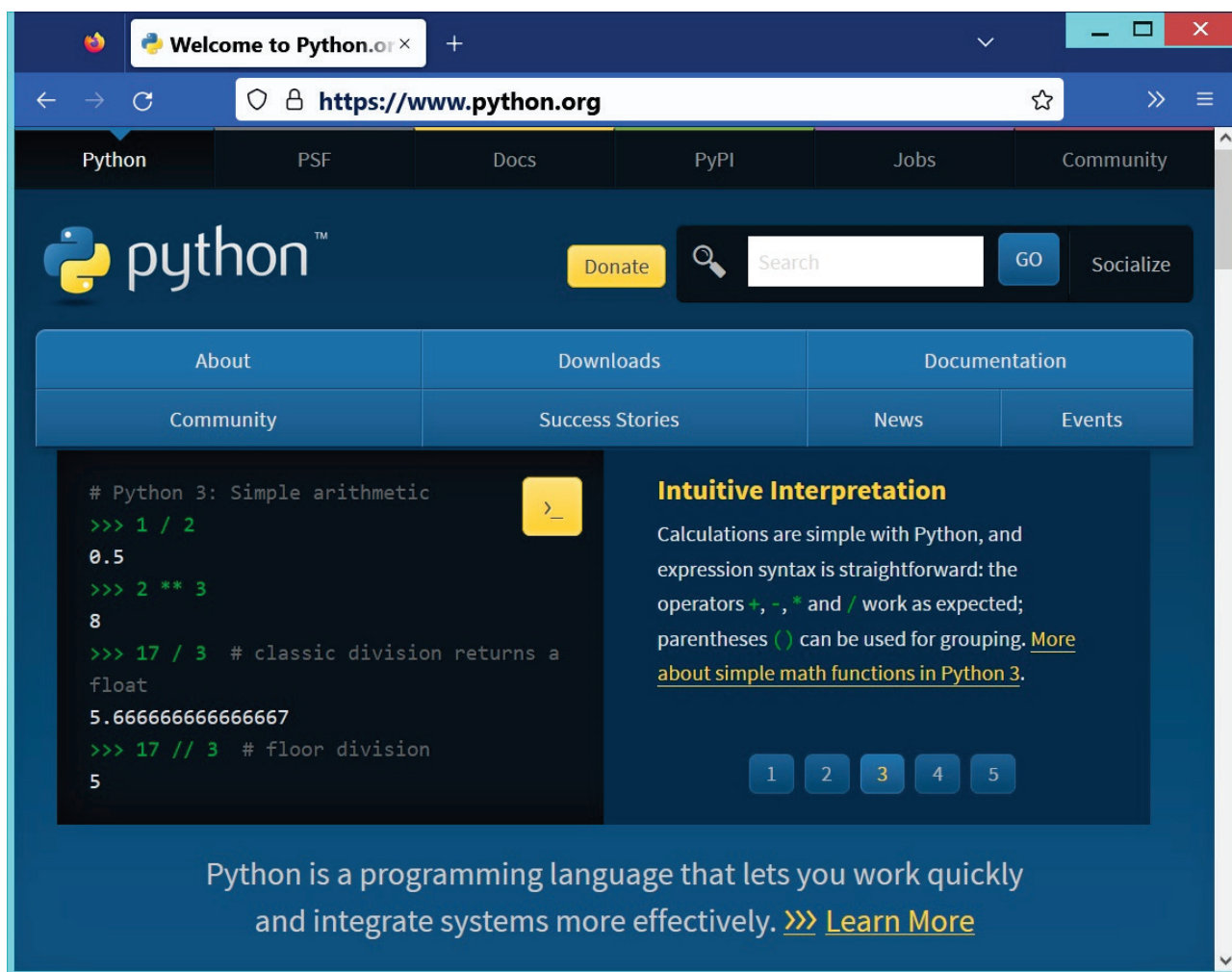


Рисунок 1.1 – Официальный сайт Python. Главная страница

Следующим шагом является выбор необходимого дистрибутива, для чего следует с основной страницы сайта перейти на страницу Downloads (Загрузки). На этой странице нужно выбрать подходящую версию дистрибутива в зависимости от установленной на компьютере пользователя операционной системы и ее конкретной версии. После скачивания дистрибутива и установки среды программирования на рабочем столе компьютера появляется ярлык, а в меню «Пуск» – соответствующий раздел (здесь рассматривается вариант установки применительно к операционной системе Windows).

После запуска среды программирования IDLE Python любым удобным для пользователя способом на экране компьютера появляется начальный экран этой среды. В верхней строке можно прочитать название текущей версии среды программирования, а ниже находится строка меню.

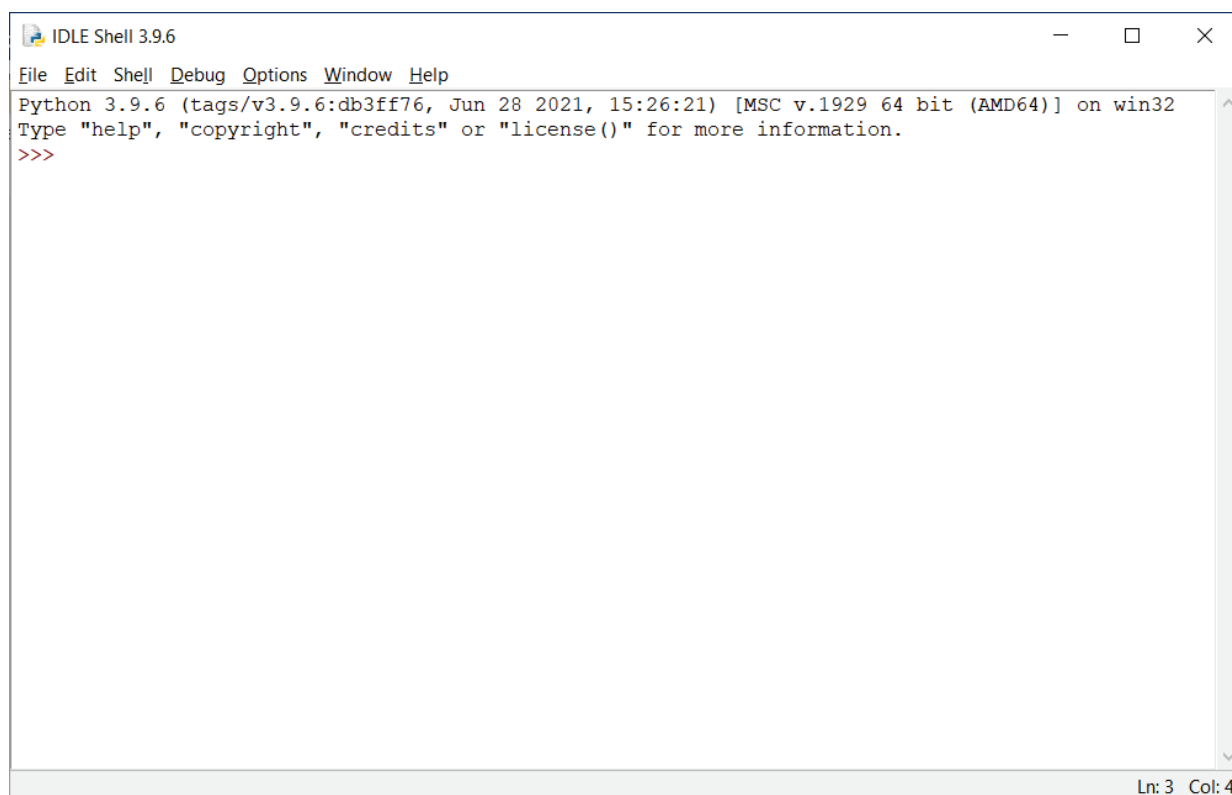


Рисунок 1.2 – Начальный экран среды программирования IDLE Python

Теперь необходимо создать еще одно окно, в котором будет отображаться исходный код (текст) создаваемой программы. Для этого следует в строке меню выбрать раздел «File», а в этом разделе пункт «New File». Появляется новое окно, в котором и нужно будет набрать, и затем сохранить текст будущей программы. Обратите внимание, что в строке заголовка этого окна написано слово «Untitled», то есть пока создаваемая программа является безымянной.

Далее, перед тем как приступить непосредственно к программированию, следует рассмотреть алфавит языка Python, основные операции, используемые в этом языке, и некоторые особенности его синтаксиса.

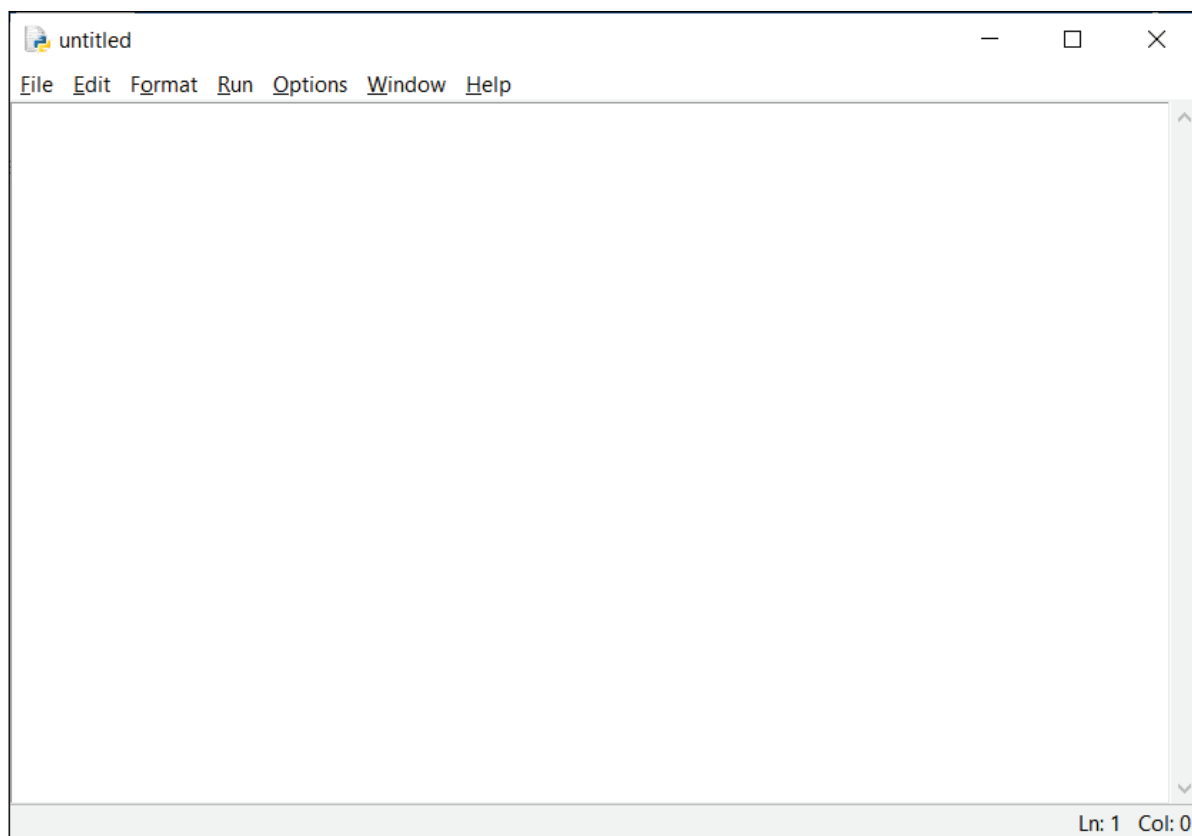


Рисунок 1.3 – Окно для написания исходного кода программы на языке Python в среде IDLE Python

Алфавит языка Python включает в себя следующие элементы:

- а) прописные буквы латинского алфавита от А до Z и строчные буквы от а до z (Python является языком, чувствительным к регистру букв, то есть имена переменных `sum` и `SUM` будут трактоваться языком как разные);
- б) цифры от 0 до 9;
- в) различные специальные символы, например, символы математических операций `+` `-` `*` `/` и другие.

Буквы кириллицы частью алфавита языка не являются, и их можно использовать только в двух ситуациях:

- а) в комментариях к тексту программы, где допустимы любые символы, имеющиеся на клавиатуре компьютера (комментарии всегда начинаются с символа `#`);
- б) в сообщениях, которые отображаются на экране компьютера с помощью операторов вывода (эти сообщения заключаются в двойные кавычки).

Кроме перечисленных элементов алфавита в языке Python существует ряд ключевых слов, имеющих фиксированное значение. Это такие слова как **`break`**, **`continue`**, **`elif`**, **`else`**, **`for`**, **`import`**, **`in`**, **`return`**, **`while`** и целый ряд других.

Программа состоит из последовательности строк, каждая из которых включает в себя определенный оператор языка Python. Операторы (команды языка) содержат ключевые слова языка Python, имена переменных, константы и знаки математических, логических и других операций, выполняемых над

переменными и константами. В отличие от других языков программирования (например, Паскаля и языков семейства С, где в конце оператора обязательно нужно ставить точку с запятой), в Python никаких специальных символов ставить в конце оператора не надо. Для того чтобы завершить ввод очередного оператора, нужно просто нажать клавишу «Enter» и перейти тем самым на следующую строку.

Переменными являются величины, которые могут изменяться в процессе выполнения программы. Каждая переменная должна иметь свое собственное уникальное имя, называемое также идентификатором переменной. Для наименования переменных в языке Python существуют следующие правила:

а) имена переменных могут включать латинские буквы, цифры и символ подчеркивания;

б) имена переменных могут начинаться только с латинской буквы или символа подчеркивания;

в) в именах переменных недопустимы пробелы и символы кириллицы (если нужно дать переменной осмысленное имя, которое состоит из двух или более слов, то пробел нужно заменять символом подчеркивания);

г) как уже было сказано ранее, прописные и строчные буквы в именах переменных различаются;

д) имена переменных не должны совпадать с ключевыми словами.

В языке Python есть следующие основные математические операции: + (сложение), – (вычитание), *(умножение), / (обычное деление), ** (возведение в степень). Помимо обычного деления в Python существуют еще две специальные операции деления, которые также будут рассмотрены в этой главе, но несколько позднее.

Теперь, вооружившись этим начальными знаниями о языке Python, мы можем приступить к написанию на этом языке первой простой программы. Эта программа должна по заданной длине и ширине прямоугольника вычислять его площадь. Исходный код программы приведен в листинге 1.1.

Листинг 1.1. Вычисление площади прямоугольника

```
# Вычисление площади прямоугольника  
# Ввод исходных данных  
a=int(input("Введите длину прямоугольника "))  
b=int(input("Введите ширину прямоугольника "))  
# Вычисление результата  
s=a*b  
# Вывод результата  
print("Площадь прямоугольника равна ",s)
```

Разберем основные элементы этой программы. Перед собственно программой в двух строках написаны комментарии к тексту этой программы. Комментарием может быть любой текст, расположенный после символа # (октогорт). Комментарии никак не влияют на работу программы, а только поясняют ее назначение.

Затем идут непосредственно строки программного кода. В этих строках переменным `a` и `b` присваиваются числовые значения, введенные пользователем с клавиатуры. В ряде языков программирования, таких как Pascal, C, C++ и других, перед тем как использовать переменную в программе, нужно обязательно описать тип ее значения (целый, вещественный или какой-либо иной). В Python такого жесткого правила нет, и имя переменной, как правило, указывают только тогда, когда ей присваивается начальное значение (эту операцию часто называют инициализацией переменной).

Оператор присваивания в Python записывается следующим образом. Слева указывается имя переменной, а справа от нее – знак равенства, который, собственно, и обозначает присваивание. В нашей программе переменной `a` присваивается значение длины прямоугольника, `b` – ширины прямоугольника. Присваиваемое значение располагается правее знака равенства. Это может быть числовая или иная константа либо имя переменной. Также это может быть выражение, содержащее константы, переменные и знаки математических или логических операций. Значение этого выражения вычисляется, и оно также присваивается находящейся в левой части оператора переменной.

Если же значение присваивается непосредственно с клавиатуры, как в данном примере, то справа от знака равенства помещается оператор `input`. После ключевого слова `input` обязательно ставится пара круглых скобок. Внутри них может находиться приглашение на ввод данных, которое обязательно заключается в двойные кавычки. Приглашение на ввод не является обязательным элементом, но, тем не менее, рекомендуется всегда его указывать для того, чтобы пользователь понимал, какую именно информацию он должен ввести с клавиатуры компьютера.

Оператор `input` позволяет ввести данные только в строковом формате, но нам для вычисления площади прямоугольника нужны числа. Поэтому для преобразования вводимых данных в начале указывается тот тип, к которому они преобразуются (в данном случае это целочисленный тип, обозначаемый ключевым словом `int`), а уже затем после него в круглых скобках указывается оператор `input` с приглашением на ввод данных. Естественно, что для правильного преобразования типов вводимые с клавиатуры данные должны представлять собой число, то есть только последовательность цифр, не содержащую каких-либо других символов.

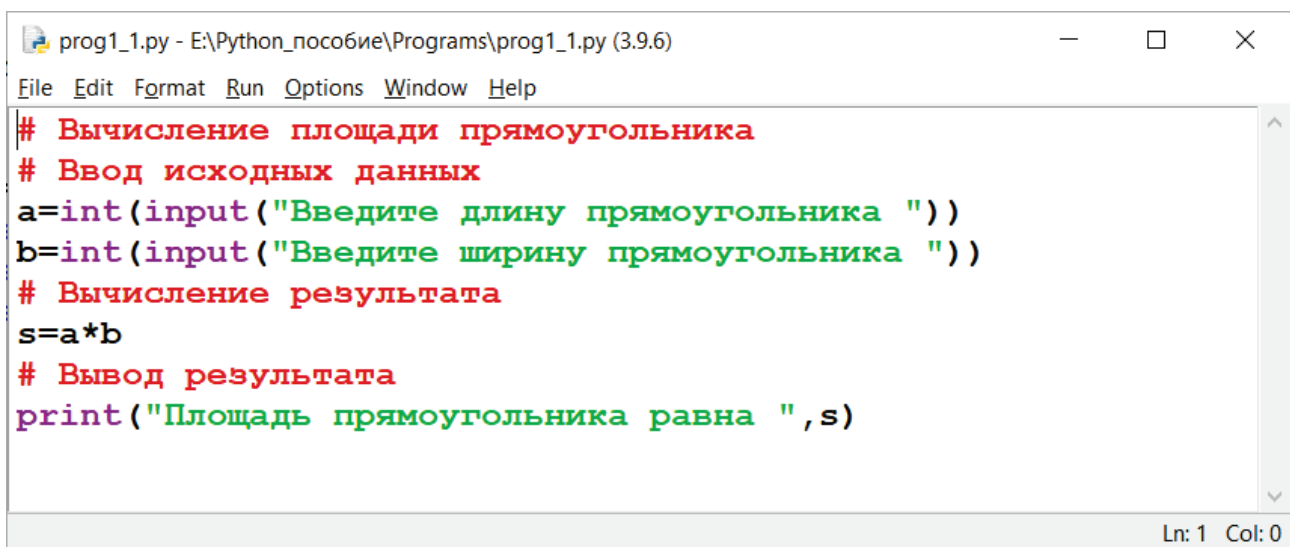
Можно также преобразовывать строковые данные к вещественному типу. Для этого перед круглыми скобками, в которые заключен оператор `input`, ставится ключевое слово `float`. В этом случае в числе, вводимом с клавиатуры, нужно отделить целую часть от дробной точкой. Кроме того, возможен еще один вариант, когда нужно преобразовать строковые данные в числовую форму, но точный формат вводимого числа (целый или вещественный) заранее неизвестен. Тогда рекомендуется использовать универсальное ключевое слово `eval`, которое преобразует данные в соответствующее числовое значение.

После ввода исходных данных в программе идет еще один комментарий, а вслед за ним оператор, который производит присваивание переменной `s`. Это

присваивание производится непосредственно в самой программе. Справа от знака равенства находится выражение, в котором перемножаются значения переменных *a* и *b*. Полученный при умножении результат присваивается переменной *s*. Поскольку в данном случае, и переменная, и присваиваемый ей результат относятся к одному и тому же числовому типу, в этом операторе никаких преобразований данных производить не нужно.

Затем в программе после очередного комментария производится вывод полученного значения переменной *s* на экран компьютера. Для этого используется оператор вывода `print`. В данном операторе после ключевого слова `print` в круглых скобках находится список вывода. Список содержит элементы, которые перечисляются через запятую. Элементами списка могут быть текстовые сообщения, которые заключаются в кавычки, и имена переменных, которые кавычек не требуют. Когда программа запускается на выполнение, то текст выводится точно так, как он представлен в операторе вывода (только без кавычек), а вместо имени переменной выводится ее значение. В нашем примере в операторе вывода содержится один текстовый элемент и одно имя переменной – *s*.

На рисунке 1.4. представлено окно среды программирования IDLE Python, в котором записан этот программный код. В данном пособии иллюстрации – черно-белые, но если набрать приведенный в листинге код на компьютере в среде IDLE Python, то будет видно, что разные части программы отображаются разными цветами. Так ключевые слова языка Python `int`, `input` и `print` будут выделены фиолетовым цветом, текст, выводимый на экран компьютера операторами `print` и `input` – зеленым, имена переменных, обозначения операций и знаки препинания (в данном примере это запятые и скобки) – черным, а комментарии к тексту программы – красным. Такая цветовая дифференциация текста облегчает разработчику программы контроль над правильностью ввода ее исходного кода.



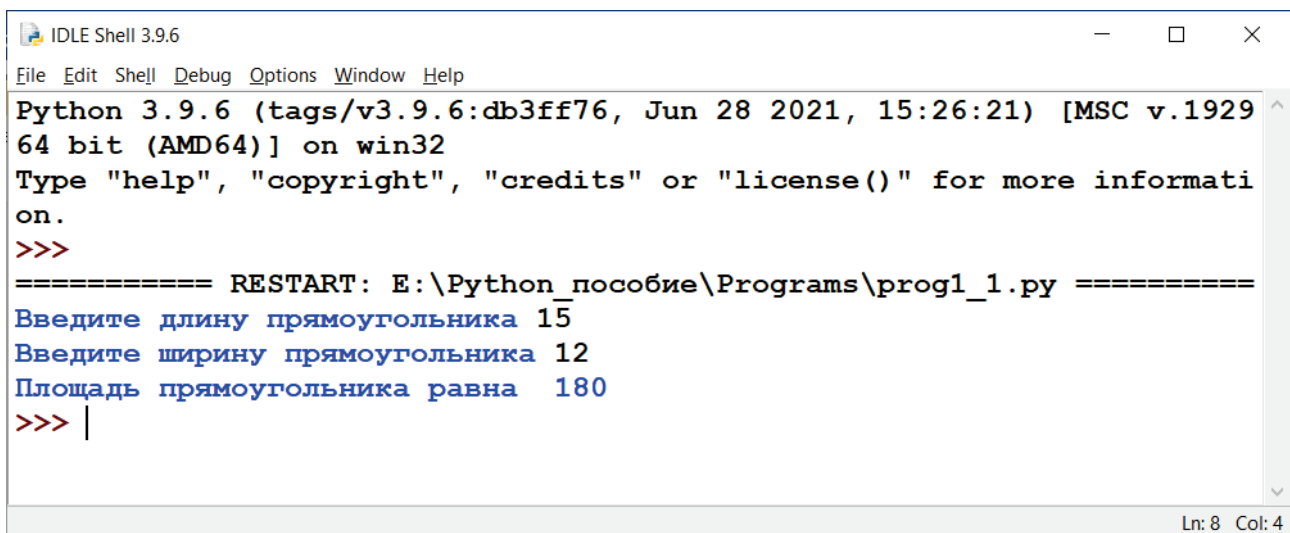
```
prog1_1.py - E:\Python_пособие\Programs\prog1_1.py (3.9.6)
File Edit Format Run Options Window Help
# Вычисление площади прямоугольника
# Ввод исходных данных
a=int(input("Введите длину прямоугольника "))
b=int(input("Введите ширину прямоугольника "))
# Вычисление результата
s=a*b
# Вывод результата
print("Площадь прямоугольника равна ",s)
Ln: 1 Col: 0
```

Рисунок 1.4 – Исходный код программы «Вычисление площади прямоугольника»

Затем нужно обязательно сохранить исходный текст программы на каком-либо носителе информации. Это может быть жесткий диск компьютера или портативный носитель, такой как флэш-память. Если пользователь этого не сделает, то программа просто не запустится на выполнение и на экран компьютера будет выведено диалоговое окно, в котором будет настоятельно предложено сохранить файл программы.

После сохранения исходного кода программы для того, чтобы проверить ее правильность и работоспособность, программу следует запустить на выполнение. Это можно сделать двумя способами. Можно использовать меню, в котором выбрать раздел «Run», а в этом разделе – пункт «Run Module». Либо же можно просто нажать на клавиатуре компьютера клавишу F5, что даст тот же результат. Если в исходном коде программы имеются ошибки, то в окне вывода появится соответствующее сообщение, причем, как правило, выделяется так строка программы, в которой эта ошибка допущена. После исправления всех обнаруженных ошибок и повторного сохранения ее текста программу нужно снова запустить на выполнение.

Если же с исходным кодом, все в порядке, то на экране компьютера появится окно, подобное тому, которое представлено на рисунке 1.5. В этом окне осуществляется ввод исходных данных и затем отображаются результаты работы программы. Здесь также используется цветовое выделение. Значения вводимых переменных отображаются черным цветом, а приглашения на ввод данных и результаты работы программы отображаются синим цветом.



```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more informati
on.
>>>
===== RESTART: E:\Python_пособие\Programs\prog1_1.py =====
Введите длину прямоугольника 15
Введите ширину прямоугольника 12
Площадь прямоугольника равна 180
>>> |
Ln: 8 Col: 4
```

Рисунок 1.5 – Результат работы программы «Вычисление площади прямоугольника»

В следующей программе производится расчет объема цилиндра и общей площади его поверхности по известной высоте цилиндра и радиусу его основания. Для расчета используются формулы, известные из стереометрии, согласно которым объем цилиндра равен $V=\pi R^2h$, а площадь его поверхности $S=2\pi R(R+h)$.

Из приведенных формул видно, что в расчетах используется число π , представляющее собой бесконечную дробь. Можно, конечно, взять из справочника значение этого числа и использовать его в расчетах в качестве константы, но более грамотным и корректным будет следующий подход: использовать в программе стандартную функцию языка Python под названием `pi`, что и сделано в приведенном ниже листинге 1.2.

Листинг 1.2. Вычисление площади поверхности и объема цилиндра

```
# Вычисление объема цилиндра и площади его поверхности  
# Подключение математического модуля  
import math  
# Ввод исходных данных  
h=eval(input("Введите высоту цилиндра"))  
r=eval (input("Введите радиус основания цилиндра"))  
# Вычисление искомых величин  
v=math.pi*r**2*h  
s=2*math.pi*r*(r+h)  
print("Объем цилиндра равен %.3f" %v)  
print ("Площадь цилиндра равна %.3f" %s)
```

Просмотрев этот листинг, отметим, что нового здесь появилось по сравнению с предыдущей программой.

Во-первых, программа начинается с подключения математического модуля оператором `import math`. Этот оператор содержит ключевое слово `import`, означающее в данном случае команду «подключить» и собственно название модуля `math`. Для чего это нужно?

Дело в том, что язык Python имеет модульную структуру. Существует основное ядро языка, которое содержит важнейшие операторы и команды, но для выполнения каких-либо специфических задач требуется подключение дополнительных программных модулей. Такие модули содержат различные дополнительные функции, расширяющие возможности языка.

Одной из таких функций и является функция `pi`, возвращающая значение числа π , вычисленного с большой точностью. Эта функция входит в состав модуля `math` и не будет работать без его подключения. Используя эту функцию в программе, мы должны указывать не только ее имя, но и модуль к которому она относится. Причем сначала указывается имя модуля, а затем, после точки имя самой функции, то есть: `math.pi`.

Во-вторых, в операторах ввода для преобразования вводимых с клавиатуры данных из строковой формы в числовую используется команда `eval`. Это расширяет возможности программы, поскольку с клавиатуры можно вводить не только целочисленные значения, как в предыдущей программе, но и вещественные.

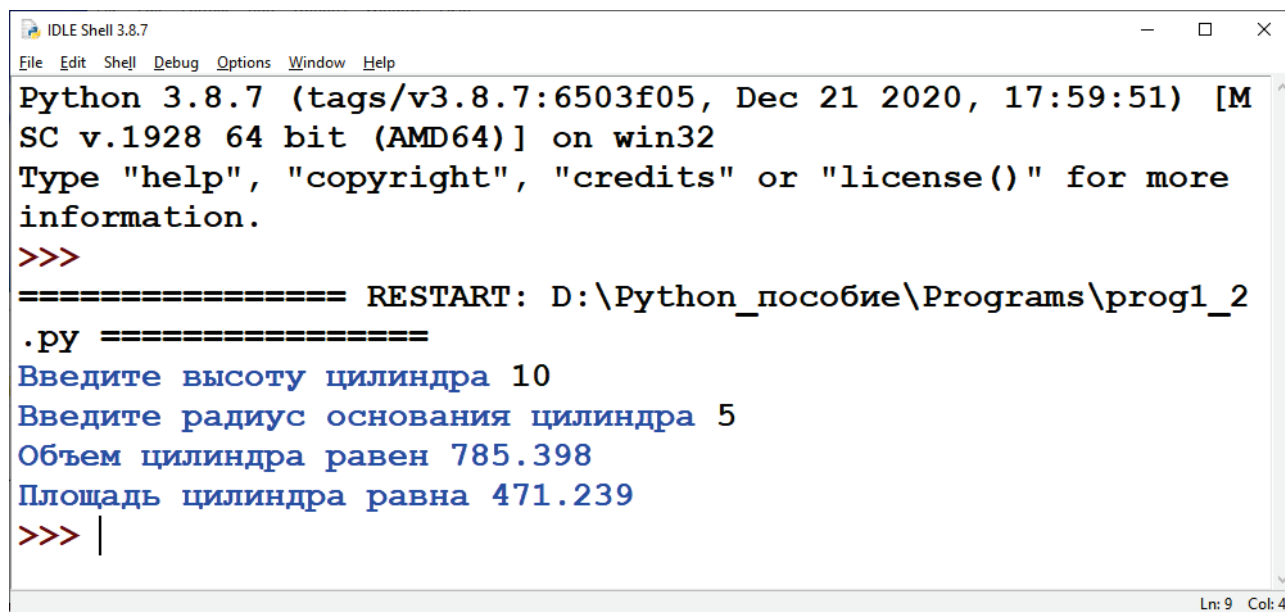
В-третьих, по сравнению с предыдущей программой изменен формат вывода данных. Вычисленные в программе значения объема и площади цилиндра выводятся на экран компьютера с большим количеством разрядов в

дробной части, но для большинства практических расчетов такая точность не требуется и достаточно указать 3-4 разряда после точки, отделяющей целую часть от дробной.

Для того чтобы задать требуемый формат вывода данных необходимо сделать следующее. В операторе `print` в списке вывода, заключенном в круглые скобки, внутри двойных кавычек (там же, где выводится текстовое сообщение) нужно указать шаблон для вывода числовой величины.

Этот шаблон начинается со знака `%`, а затем указывается, собственно, формат. Для вещественного числа он должен содержать точку, количество знаков, которое должно быть в дробной части числа, и букву `f`, показывающую, что данное число относится к вещественному типу `float`. Например, шаблон `%.2f` обозначает, что выводимое вещественное число будет содержать два разряда в дробной части. Затем после закрывающих кавычек и пробела указывается имя переменной, которая должна быть отформатирована по заданному слева от нее шаблону. Перед именем переменной также ставится знак `%`. Следует обратить внимание на то, что при форматировании вышеуказанным способом не просто отбрасываются «лишние» разряды, а вещественное число округляется по принятым в математике правилам.

На рисунке 1.6 представлены результаты работы программы, содержащейся в листинге 1.2.

A screenshot of a Python IDLE Shell window. The title bar reads "IDLE Shell 3.8.7". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The shell displays the following text:

```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
===== RESTART: D:\Python_пособие\Programs\prog1_2
.py =====
Введите высоту цилиндра 10
Введите радиус основания цилиндра 5
Объем цилиндра равен 785.398
Площадь цилиндра равна 471.239
>>> |
```

The status bar at the bottom right shows "Ln: 9 Col: 4".

Рисунок 1.6 – Результат работы программы вычисления объема и площади поверхности цилиндра

Как уже отмечалось ранее, помимо обычных математических операций, в языке Python существуют две операции деления, которые могут применяться только по отношению к целочисленным величинам. Это операции целочисленного деления и нахождения остатка при делении. Целочисленное деление обозначается символами `//` (двойной косой чертой), а нахождение

остатка при делении % (знаком процента). При целочисленном делении в начале выполняется обычная операция деления, а затем из получившегося частного берется только целая часть, а дробная часть отбрасывается.

Практическое применение этих операций мы рассмотрим на следующем примере. Требуется составить программу, которая находит сумму цифр целого положительного трехзначного числа, введенного пользователем с клавиатуры. Текст программы, решающей данную задачу, приведен в листинге 1.3.

Листинг 1.3. Вычисление суммы цифр трехзначного числа

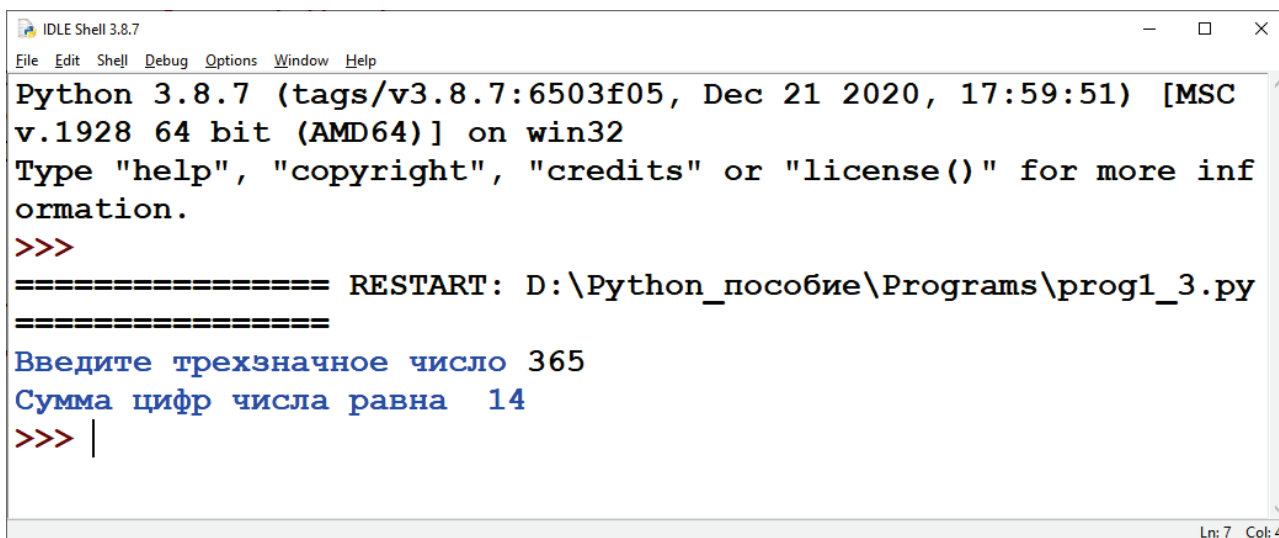
```
# Вычисление суммы цифр трехзначного числа  
# Ввод исходных данных  
numb=int(input("Введите трехзначное число"))  
# Вычисление искомого результата  
r1=numb//100  
p=numb%100  
r2=p//10  
r3=p%10  
sum=r1+r2+r3  
# Вывод результата вычислений  
print("Сумма цифр числа равна ",sum)
```

Рассмотрим основные элементы данной программы. Сначала происходит ввод данных с клавиатуры с помощью оператора input. Поскольку по условиям задачи вводимое число должно быть целым, то преобразуем его к данному типу с помощью ключевого слова int. Это целочисленное значение присваивается переменной numb.

Затем находим старший разряд числа numb (количество сотен). Для этого выполняем целочисленное деление исходного числа на 100 (операция //). Результат деления присваивается переменной r1. Это и будет первая цифра числа. Далее с помощью операции % находим остаток от деления числа на 100. Это значение присваивается вспомогательной переменной p. В данной переменной находится количество десятков и единиц числа. На следующем этапе необходимо найти вторую цифру числа, то есть количество десятков. Для этого вновь необходимо прибегнуть к целочисленному делению, на этот раз на 10. Полученный результат присваивается переменной r2. Для нахождения младшего разряда числа, то есть количества единиц, выполняем операцию нахождения остатка от деления вспомогательной переменной p на 10. Это и будет последняя искомая цифра данного трехзначного числа, которая будет присвоена переменной r3.

После нахождения всех цифр, содержащихся в переменных r1, r2 и r3, остается только сложить их и присвоить полученное значение переменной sum, которая выводится на экран компьютера с помощью уже известного нам оператора print.

На рисунке 1.7. продемонстрированы результаты работы программы, содержащейся в листинге 1.3.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC
v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>>
===== RESTART: D:\Python_пособие\Programs\prog1_3.py =====
Введите трехзначное число 365
Сумма цифр числа равна 14
>>> |
```

Рисунок 1.7 – Результат работы программы «Вычисление суммы цифр трехзначного числа»

Все приведенные в данной главе примеры относятся к программам, имеющим линейную структуру. Это означает, что все содержащиеся в программе операторы выполняются последовательно друг за другом, что и позволяет получить требуемый результат. Но подобная структура является только одной из трех базовых структур программирования. Другие структуры, которые необходимо знать (и уметь ими пользоваться) для составления полноценной программы на языке Python, будут рассмотрены в следующих главах этого пособия.

Контрольные вопросы

1. Какой способ загрузки дистрибутива среды программирования на языке Python является наиболее безопасным и оптимальным?
2. Из чего нужно исходить при выборе версии дистрибутива для установки среды программирования Python?
3. Как в среде IDLE Python создать новый файл для записи в нем исходного текста программы?
4. Что в среде программирования IDLE Python необходимо сделать в обязательном порядке перед запуском программы на выполнение?
5. Как в среде Python IDLE запустить готовую программу на выполнение?
6. Каким образом в языке Python определяется тип значения переменной?
7. Каким образом в языке Python можно подключить стандартную библиотеку для математических вычислений?
8. С какого символа начинается в языке Python комментарий?
9. Как в языке Python записывается операция возведения в степень?
10. Как в языке Python записывается операция целочисленного деления?

ГЛАВА 2. ОПЕРАТОРЫ ВЫБОРА

При решении различных задач научного и технического характера, как и в повседневной жизни, часто приходится выбирать один, наиболее оптимальный, вариант действий из двух или более возможных. Для этой цели в языке Python существует оператор выбора `if`. Оператор `if` имеет несколько разновидностей, а какую именно из них следует использовать в конкретном случае, зависит от поставленной перед программистом задачи.

Ниже приведен общий вид оператора выбора.

```
if условие:  
    блок 1  
else:  
    блок 2
```

Механизм работы данного оператора следующий. Сначала проверяется условие, которое находится в заголовке оператора после ключевого слова `if`. После условия обязательно ставится двоеточие. Как правило, условие представляет собой операцию сравнения каких-либо двух величин. Всего в Python существует 6 операций сравнения:

`<` (меньше) `<=` (меньше или равно) `>` (больше)
`>=` (больше или равно) `=` (равно) `!=` (не равно)

Результатом операции сравнения является логическая величина, значение которой может быть равно `True` (истина) или `False` (ложь). Если условие, приведенное в заголовке, выполняется, то значение выражения равно `True`, и выполняется блок1. Если же это условие не соблюдается, то его значение равно `False`, и выполняется блок2, находящийся после ключевого слова `else` с двоеточием.

Каждый блок может состоять из одного или нескольких операторов языка Python. Для того, чтобы можно было понять, где начинается и где заканчивается блок, в Python принято следующее правило: каждая строка блока должна быть сдвинута вправо относительно строки, содержащей служебное слово `if` или слово `else` как минимум на 4 позиции. Подобная форма записи условного оператора `if` называется полной.

Наряду с полной формой записи оператора выбора в языке Python используется и сокращенная. В этом случае условный оператор будет выглядеть следующим образом.

```
if условие:  
    блок
```

То есть служебное `else` и связанный с ним блок в такой записи отсутствуют. Механизм работы оператора выбора изменяется следующим образом: если условие, написанное после слова `if` истинно, то выполняется

записанный после этого условия блок. Если же это условие ложно, то данный оператор никаких действий не выполняет, и программа переходит к выполнению следующего оператора.

Для того, чтобы на практике освоить работу оператора выбора, составим программу, которая находит наибольшее и наименьшее в группе из 3 чисел. В этой программе представлены и полная, и сокращенная форма условного оператора, а сама программа представлена в листинге 2.1.

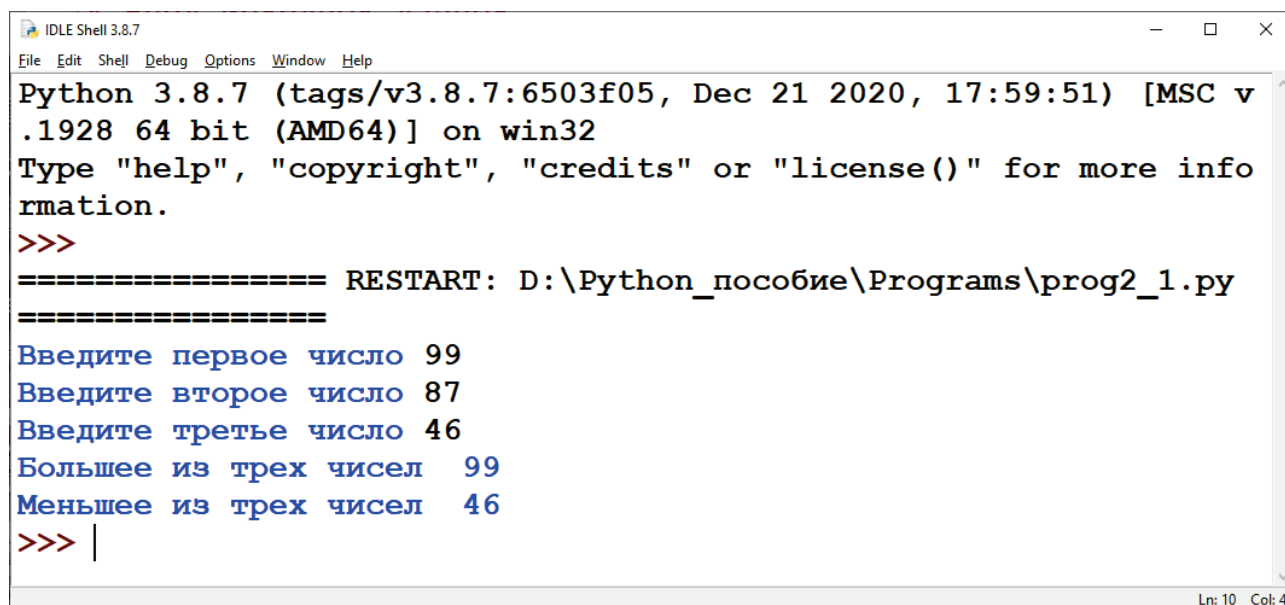
Листинг 2.1. Нахождение наибольшего и наименьшего в группе из трех чисел, вводимых с клавиатуры

```
# Нахождение наибольшего в группе из трех чисел
# Ввод исходных данных
a=int(input("Введите первое число"))
b=int(input("Введите второе число"))
c=int(input("Введите третье число"))
# Поиск и вывод наибольшего числа
if a>b:
    x=a
else:
    x=b
if c>x:
    x=c
print("Большее из трех чисел ",x)
# Поиск и вывод наименьшего числа
if a<b:
    y=a
else:
    y=b
if c<y:
    y=c
print("Меньшее из трех чисел ",y)
```

Данная программа начинается с ввода с клавиатуры трех целых чисел a, b и c. Затем производится поиск наибольшего числа. Для этого в начале с помощью полного условного оператора if производится выбор большего из двух чисел a и b. Большее из этих двух чисел присваивается вспомогательной переменной x. Затем производится сравнение x с оставшимся числом c. Для этого используется сокращенный условный оператор if. Если третье число c больше x, то это значение (максимальное из трех) присваивается x. Если это условие не выполняется, то c меньше чем x и, следовательно, в переменной x уже содержится большее из трех чисел. Тогда этот оператор if не должен выполнять никаких действий и управление переходит к следующему оператору print, который выводит наибольшее число на экран компьютера. Аналогичным

образом выполняется и поиск наименьшего числа, только операции сравнения заменяются на противоположные.

На рисунке 2.1 показан результат работы программы, содержащейся в листинге 2.1.



```
IDLE Shell 3.8.7
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v
.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more info
rmation.
>>>
===== RESTART: D:\Python_пособие\Programs\prog2_1.py
=====
Введите первое число 99
Введите второе число 87
Введите третье число 46
Большее из трех чисел 99
Меньшее из трех чисел 46
>>> |
```

Рисунок 2.1 – Результат работы программы «Нахождение наибольшего и наименьшего в группе из трех чисел»

В приведенной выше программе в каждом условном операторе использовалась только одна операция сравнения, то есть проверялось только одно условие. Но в ряде задач для их решения требуется совместная проверка нескольких условий. Для этой цели в языке Python существуют специальные логические операции, которые могут выполняться над логическими величинами, такими как результаты операций сравнения. Таких операций в языке Python существует три: `not`, `or` и `and`. Действуют они следующим образом:

- `not` изменяет значение логической величины на противоположное: если значение этой величины было изначально равно `True`, то оно изменяется на `False`; если же значение было равно `False`, то операция `not` меняет его на `True`. Оператор `not` является унарным, то есть он может быть применен только к одной величине;

- `or` является бинарной операцией, то есть он применяется по отношению к двум логическим величинам. Результат операции `or` будет равен `False` только в том случае, если оба операнда (так называются величины, над которыми выполняется операция) имеют значение `False`. Если же оба операнда имели значение `True` или хотя бы один из операндов имел это значение, то результат операции `or` всегда будет равен `True`;

- `and`, так же как и `or`, является бинарной операцией. Результат этой операции будет равен `True` в том и только в том случае, если оба исходных операнда имеют значение `True`. Если же оба операнда имеют значение `False` или

хотя бы один из них имеет указанное значение, то результат операции `and` будет равен `False`.

Использование логических операций мы рассмотрим в следующей задаче. Требуется найти площадь треугольника по формуле Герона. Эта формула, созданная выдающимся древнегреческим ученым и изобретателем Героном Александрийским еще в I веке нашей эры, позволяет вычислить площадь треугольника по известным длинам его сторон. При этом после ввода исходных данных необходимо проверить, может ли вообще существовать такой треугольник. Как известно для того, чтобы треугольник существовал, нужно чтобы сумма длин любых двух его сторон была больше, чем третья. Программа, решающая эту задачу, приведена в листинге 2.2.

Листинг 2.2. Вычисление площади треугольника по формуле Герона

```
# Проверка существования треугольника и  
# вычисление его площади по формуле Герона  
import math  
# Ввод исходных данных  
a=int(input("Введите длину первой стороны"))  
b=int(input("Введите длину второй стороны"))  
c=int(input("Введите длину третьей стороны"))  
# Проверка введенных данных  
if (a+b)>c and (a+c)>b and (b+c)>a:  
    # Вычисление искомой величины  
    p=(a+b+c)/2  
    s=math.sqrt(p*(p-a)*(p-b)*(p-c))  
    print("Площадь треугольника равна",s)  
else:  
    # Вывод сообщения об ошибке  
    print("Такого треугольника не существует")
```

В начале программы к ней подключается математический модуль `math`. Этот модуль будет необходим, поскольку в формуле Герона используется операция извлечения квадратного корня, которая содержится в данном модуле. Затем производится ввод исходных данных – трех сторон треугольника `a`, `b` и `c`.

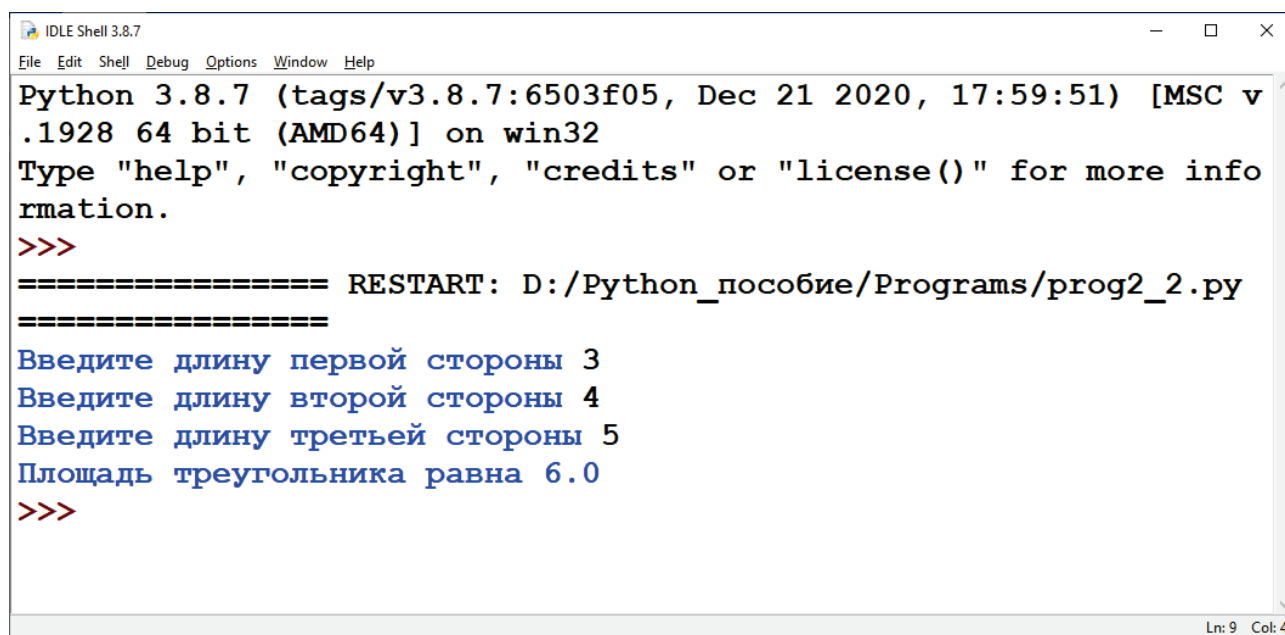
После ввода данных производится их проверка, чтобы выяснить возможность существования треугольника. Проверка осуществляется с помощью двух логических операций `and`, которые объединяют между собой три операции сравнения. Все это логическое выражение находится в заголовке оператора выбора `if`. В каждой из операций сравнения проверяется истинность того, что сумма двух сторон треугольника больше третьей. Только при истинности всех этих трех операций имеет смысл дальнейшее вычисление площади треугольника, которое выполняется в блоке, расположенном после заголовка оператора `if`.

Этот блок, помимо комментария, содержит три оператора. В первом из них вычисляется полупериметр треугольника, необходимый для дальнейших

вычислений. Затем во втором операторе на основе найденного ранее значения полупериметра и известных длин сторон вычисляется сама эта площадь. При вычислениях используется стандартная функция `sqrt`, которая извлекает квадратный корень из ее аргумента. Поскольку данная функция не входит в ядро языка Python, а загружается из внешнего подключенного модуля, то перед названием самой функции необходимо указать название содержащего ее модуля `math` и поставить точку между названием модуля и названием функции. Третий оператор выводит полученный при вычислениях результат на экран компьютера.

Если же хотя бы одно из условий оказалось ложным, то выполняется тот блок условного оператора, который находится после ключевого слова `else`. Этот блок содержит единственный оператор, который выводит на экран компьютера сообщение о том, что треугольника с такими длинами сторон не существует в принципе.

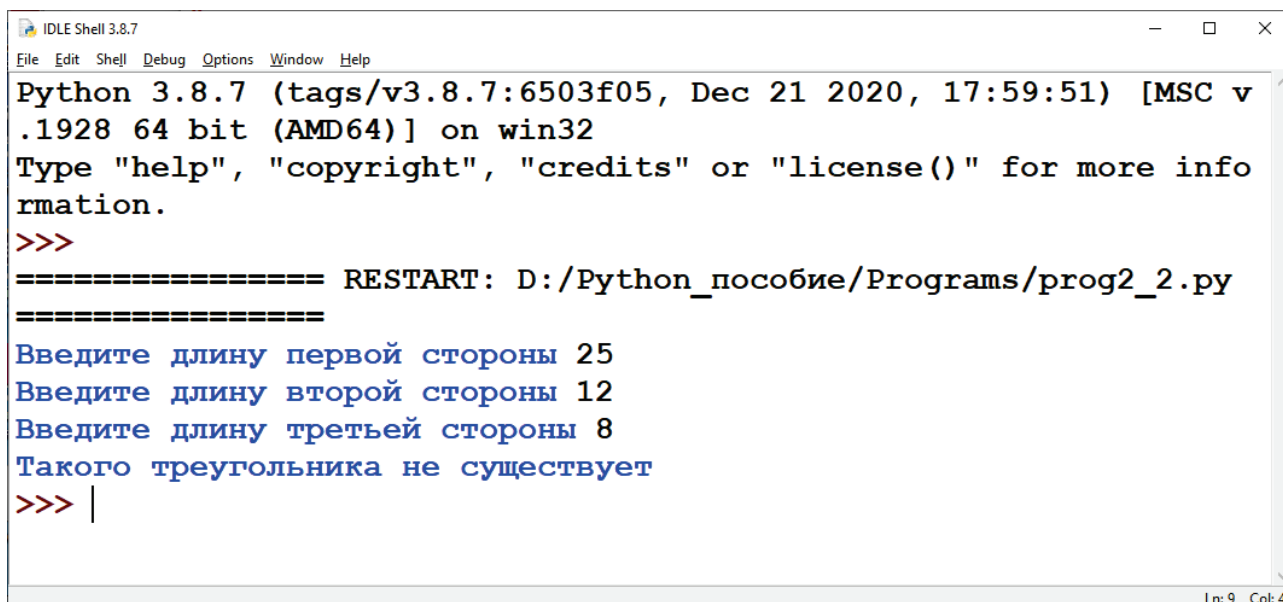
На рисунке 2.2. показана работа программы в том случае, когда исходные данные были введены корректно. В этом случае программа рассчитывает площадь искомого треугольника и выводит его на экран компьютера.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v
.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more info
rmation.
>>>
===== RESTART: D:/Python_пособие/Programs/prog2_2.py
=====
Введите длину первой стороны 3
Введите длину второй стороны 4
Введите длину третьей стороны 5
Площадь треугольника равна 6.0
>>>
```

Рисунок 2.2 – Результат работы программы «Вычисление площади треугольника по формуле Герона» при корректно введенных данных

Если же исходные данные были введены некорректно, то результат работы программы будет таким, как показано на рисунке 2.3 (при вводе данных была указана длина первой стороны треугольника, которая больше, чем сумма длин второй и третьей стороны). Программа никаких вычислений не выполняет, а только выводит сообщение об ошибке.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v
.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more info
rmation.
>>>
===== RESTART: D:/Python_пособие/Programs/prog2_2.py
=====
Введите длину первой стороны 25
Введите длину второй стороны 12
Введите длину третьей стороны 8
Такого треугольника не существует
>>> |
```

Рисунок 2.3 – Результат работы программы «Вычисление площади треугольника по формуле Герона» при некорректно введенных данных

В рассмотренных ранее программах нам необходимо было сделать правильный выбор только из двух возможных вариантов. Но достаточно часто встречаются задачи, в которых количество предлагаемых вариантов действия может быть и значительно больше. В таких случаях для решения задачи можно использовать еще один вариант условного оператора `if`, который будет рассмотрен далее. Общий вид этого оператора следующий:

```
if условие 1:
    блок 1
elif условие 2:
    блок 2
elif условие 3:
    блок 3
.....
elif условие n:
    блок n
else
    блок n+1
```

Механизм работы такого условного оператора следующий. Сначала проверяется то условие, которое находится после ключевого слова `if`. Если это условие истинно, то выполняется находящийся после него блок 1, после чего работа оператора завершается.

Если же условие оказалось ложным, то производится следующая проверка. Теперь проверяется условие 2, которое находится после ключевого слова `elif`. В случае, если это условие оказалось истинным, выполняется расположенный ниже блок2 и на этом работа оператора заканчивается. Если же

и это условие оказалось ложным, то проверяется условие 3, расположенное после следующего ключевого слова `elif` и так далее. Такие проверки могут продолжаться до тех пор, пока программа не дойдет до последнего содержащегося в операторе слова `elif` и соответствующего ему блока.

Если же и это последнее условие окажется ложным, то выполняется тот блок, который находится в операторе после ключевого слова `else`, и только тогда оператор завершает свою работу.

В качестве примера использования приведенной выше структуры рассмотрим следующую задачу. Компьютерная фирма для того, чтобы стимулировать своих клиентов приобретать у нее больше продукции на большие денежные суммы, ввела следующую систему скидок:

- если клиент фирмы делает заказ на сумму менее 3000 руб, то скидка ему не положена;
- если сумма заказа составляет от 3000 до 4999 руб, то величина скидки составляет 3% от суммы заказа;
- если сумма заказа составляет от 5000 до 9999 руб, то величина скидки составляет 5% от суммы заказа;
- если сумма заказа составляет от 10000 до 19999 руб, то величина скидки составляет 10% от суммы заказа;
- если сумма заказа составляет 20000 руб и более, то величина скидки составляет 15% от суммы заказа.

Требуется составить программу, которая по введенной с клавиатуры сумме заказа определяет, имеет ли клиент право на скидку, и, если эта скидка действительно существует, указывает величину этой скидки и итоговую стоимость заказа (то есть его начальную стоимость за вычетом скидки). Исходный код данной программы приведен в листинге 2.3.

Листинг 2.3. Определение суммы скидки на заказ, сделанный клиентом фирмы, и итоговой стоимости заказа

```
# Определение суммы скидки
# Ввод исходных данных
sum=int(input("Введите стоимость заказа в рублях"))
# Вычисление суммы скидки и вывод этой суммы
# и итоговой стоимости заказа
if sum < 3000:
    print("Извините, скидка Вам не положена")
elif (sum >= 3000) and (sum < 5000):
    dis=0.03*sum
    itog=sum-dis
    print ("Ваша скидка составит ",dis," руб")
    print ("Итоговая стоимость заказа составит ",itog," руб")
elif (sum >= 5000) and (sum < 10000):
    dis=0.05*sum
    itog=summa-dis
```

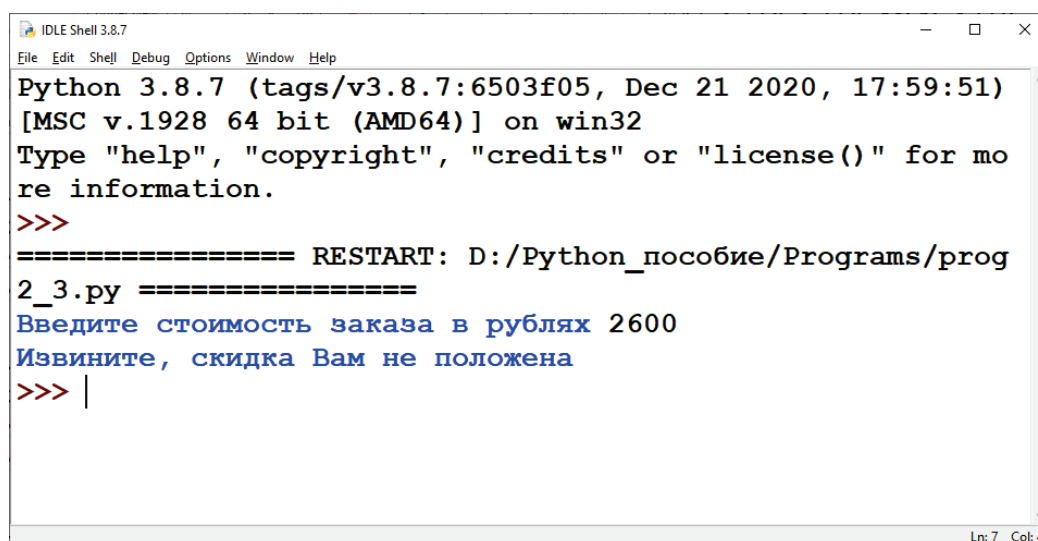
```

    print ("Ваша скидка составит ",dis," руб")
    print ("Итоговая стоимость заказа составит ",itog," руб")
elif (sum >= 10000) and (sum < 20000):
    dis=0.1*sum
    itog=sum-dis
    print ("Ваша скидка составит ",dis," руб")
    print ("Итоговая стоимость заказа составит ",itog," руб")
else:
    dis=0.15*sum
    itog=summa-dis
    print ("Ваша скидка составит ",dis," руб")
    print ("Итоговая стоимость заказа составит ",itog," руб")

```

Рассмотрим основные детали работы данной программы. Программа начинается с ввода исходных данных – суммы заказа, которая присваивается переменной `sum`. Затем в операторе `if...elif...else` рассматриваются все возможные варианты (всего таких вариантов в программе 5). Первый вариант относится к ситуации, когда сумма меньше 3000. В этом случае никаких расчетов производить не надо, и следует только вывести сообщение о том, что в данном случае скидка отсутствует. В следующих вариантах со второго по пятый включительно выполняется одна и та же последовательность действий. Сначала вычисляется сумма скидки, которая присваивается переменной `dis`. Эта сумма определяется процентом скидки, который зависит от значения переменной `sum`. Диапазон значения `sum` задается с помощью 2 операций сравнения, объединенных логической операцией `and`. Затем вычисляется итоговая сумма заказа `itog` путем вычитания величины `dis` из исходной `sum`. Затем обе вычисленные величины выводятся на экран компьютера.

На рисунке 2.4 показан результат работы программы в том случае, когда сумма заказа меньше 3000 и, следовательно, скидка отсутствует.



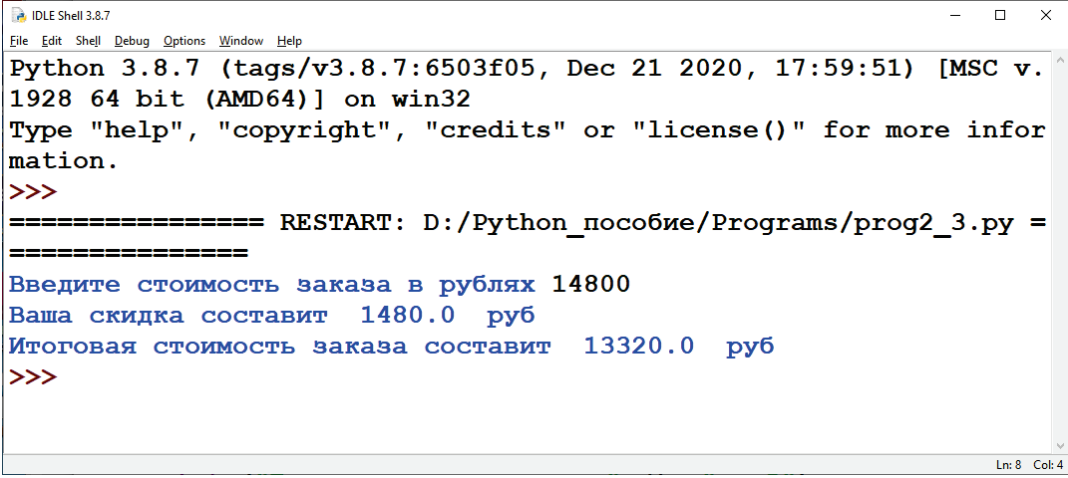
```

IDLE Shell 3.8.7
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51)
[MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
===== RESTART: D:/Python_пособие/Programs/prog
2_3.py =====
Введите стоимость заказа в рублях 2600
Извините, скидка Вам не положена
>>> |

```

Рисунок 2.4 – Результат работы программы «Определение суммы скидки»
(в данном случае скидка отсутствует)

На рисунке 2.5 показан другой вариант работы программы, когда сумма заказа превышает 3000 руб. В этом случае на экран выводится и величина скидки, и итоговая стоимость заказа за вычетом этой скидки.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/Python_пособие/Programs/prog2_3.py =====
Введите стоимость заказа в рублях 14800
Ваша скидка составит 1480.0 руб
Итоговая стоимость заказа составит 13320.0 руб
>>>
```

Рисунок 2.5 – Результат работы программы «Определение суммы скидки»
(выведена сумма скидки и итоговая стоимость заказа)

Хотя вариант условного оператора `if` с добавлением ключевого слова `elif` позволяет выбрать подходящий вариант действий из нескольких возможных, часто для решения подобного рода задач удобнее использовать специальный оператор множественного выбора. При этом следует иметь в виду, что такой оператор появился в языке Python только начиная с версии 3.10, то есть для того, чтобы применять этот оператор, нужно иметь на компьютере соответствующую версию среды программирования или установить ее с сайта разработчика языка python.org.

Пользователям операционной системы Windows необходимо знать, что для установки этой версии Python на компьютере должна быть установлена версия операционной системы не старше Windows 8. На компьютерах с Windows 7 и Windows XP установить Python 3.10 не получится.

Общий вид оператора множественного выбора в языке Python следующий.

```
match x:
    case x1:
        блок 1
    case x2:
        блок 2
    case 3:
        блок 3
    .....
    case n:
        блок n
    case _:
        блок n+1
```

В начале данного оператора записывается строка заголовка, которая содержит ключевое слово `match` и имя переменной `x`. Затем перечисляются несколько ветвей этого оператора, каждая из которых начинается с ключевого слова `case`. Каждая строка с `case` должна быть сдвинута как минимум на 4 позиции вправо относительно строк со словом `match`.

После каждого `case` указывается одно из возможных значений переменной `x`. Можно после `case` указать и группу значений переменной `x`, но в этом случае значения нужно отделить друг от друга вертикальной чертой и заключить эту группу значений в круглые скобки. После значения или группы значений обязательно ставится двоеточие.

Затем после строки с `case` (с отступом в 4 позиции) идет блок, содержащий оператор или группу операторов. Этот блок выполняется в том случае, если переменная `x` принимает соответствующее значение или одно из группы значений.

В последней ветви `case` после ключевого слова ставится символ подчеркивания. Эта ветвь оператора `match` выполняется в том случае, если в программе переменная `x` не принимает ни одного из перечисленных в операторе значений. Часто эта заключительная ветвь используется для защиты от неправильного ввода данных и в ней выводится сообщение об ошибке.

Пример использования оператора `match...case` приведен в листинге 2.4. В этом листинге содержится программа, которая должна по номеру месяца определять, к какому времени года этот месяц относится. Если пользователь случайно ввел несуществующий номер месяца, то программа должна выдавать сообщение об ошибке.

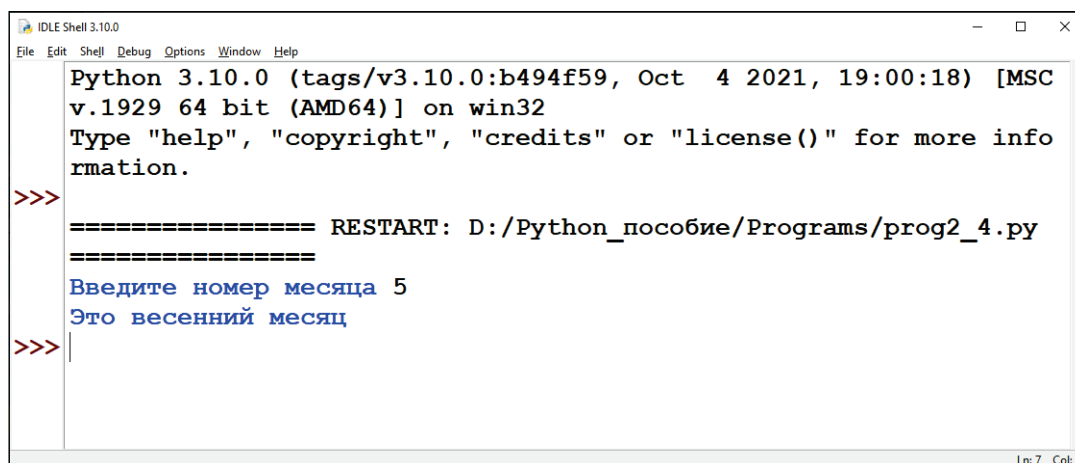
Листинг 2.4. Определение времени года по номеру месяца

```
# Определение времени года
# Ввод исходных данных
n=int(input("Введите номер месяца"))
# Вывод результата
match(n):
    case(1 | 2 | 12):
        print("Это зимний месяц")
    case(3 | 4 | 5):
        print("Это весенний месяц")
    case(6 | 7 | 8):
        print("Это летний месяц")
    case (9 | 10 | 11):
        print("Это осенний месяц")
# Вывод сообщения об ошибке
case _:
    print("Месяца с таким номером не существует")
```

Программа начинается с ввода исходных данных. В данном случае это номер месяца `n`. Обратите внимание, что вводимая с клавиатуры величина

преобразуется к целому типу, потому что переменная, используемая в операторе множественного выбора, не должна быть вещественной. Затем в операторе `match` после каждого из 4 ключевых слов `case` приводится возможный набор значений переменной `n` для соответствующего времени года. Последняя пятая ветвь оператора будет задействована в том случае, если пользователь ввел несуществующий номер месяца.

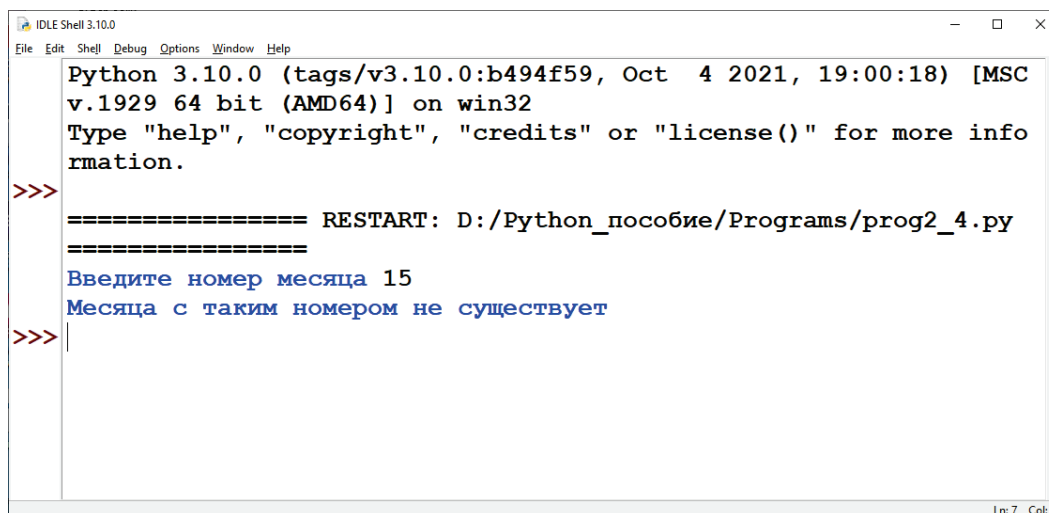
На рисунке 2.6 приведен пример корректного ввода данных в программу. Пользователь ввел номер месяца 5 и получил сообщение о том, что это весенний месяц. Действительно, это весенний месяц май.



```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC
v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more info
rmation.
>>>
===== RESTART: D:/Python_пособие/Programs/prog2_4.py
=====
Введите номер месяца 5
Это весенний месяц
>>>
```

Рисунок 2.6 – Результат работы программы «Определение времени года» при правильном вводе данных

На рисунке 2.7 показана ситуация, когда невнимательный пользователь ввел несуществующий номер месяца 15 и получил от программы сообщение об ошибке. Таким образом, и в том, и в другом случае программа отреагировала на ввод данных корректно.



```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC
v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more info
rmation.
>>>
===== RESTART: D:/Python_пособие/Programs/prog2_4.py
=====
Введите номер месяца 15
Месяца с таким номером не существует
>>>
```

Рисунок 2.7 – Результат работы программы «Определение времени года» при ошибочном вводе данных

Контрольные вопросы

1. С какого ключевого слова всегда начинается оператор выбора в языке Python?
2. Какие операции сравнения существуют в языке Python?
3. Какой символ должен стоять в языке Python после операции сравнения?
4. В чем разница между полным и сокращенным условным оператором в языке Python?
5. Как в языке Python создать условный оператор, в котором выбор можно производить не из двух, а из нескольких значений?
6. Перечислите известные Вам логические операции в языке Python.
7. Какое количество отступов нужно делать в начале блока, находящегося после заголовка условного оператора.
8. Какая логическая операция является унарной?
9. Какие логические операции являются бинарными?
10. Как записывается в языке Python оператор множественного выбора?

ГЛАВА 3. ОПЕРАТОРЫ ЦИКЛА

При решении многих задач из области программирования возникает необходимость многократного повторения определенной последовательности действий. Для этой цели используется одна из базовых структур программирования, которая называется циклом. Любой цикл состоит из двух основных компонентов:

а) заголовок, в котором задается условие повторения цикла;

б) повторяющийся оператор или группа операторов, которая называется телом цикла.

В языке Python существует два основных вида оператора цикла: `for` и `while`. Начнем рассмотрение циклов с оператора `for`, который применяется в тех ситуациях, когда число повторений цикла заранее известно. Общий вид этого оператора следующий.

`for i in sequence:`

тело_цикла

где, **`for`** и **`in`** – это ключевые слова, **`i`** – переменная цикла (она должна принадлежать к целому типу), а **`sequence`** – некоторый набор элементов.

В конце заголовка обязательно должно стоять двоеточие. После заголовка цикла находится тело цикла. Тело цикла представляет собой блок, состоящий из одного или нескольких операторов. Этот блок должен быть смещен вправо относительно заголовка цикла на 4 позиции. В ходе работы цикла последовательно перебираются все элементы набора. Каждый раз, когда из набора выбирается очередной элемент, происходит выполнение тела цикла. Когда все элементы в наборе перебраны, цикл прекращает свою работу.

Но чаще используется другой вариант цикла `for`, в котором присутствует функция `range` (одна из стандартных функций языка Python). Тогда общий вид оператора цикла `for` выглядит так.

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

тело_цикла

Функция `range` позволяет перебирать элементы числовой последовательности в диапазоне, заданном в круглых скобках после ключевого слова **`range`**. Механизм работы оператора цикла **`for`** с использованием функции **`range`** следующий. В начале переменной цикла **`i`** присваивается значение **`a`**, и затем первый раз выполняется тело цикла. После этого значение переменной цикла увеличивается на единицу, и вновь выполняется тело цикла. Потом переменная цикла еще раз увеличивается на единицу, тело цикла снова выполняется и так продолжается несколько раз до тех пор, пока переменная цикла не достигнет конечного значения, которое равно **`b-1`** (а не **`b`**). Тогда тело цикла выполняется в последний раз и цикл на этом завершает свою работу.

Рассмотрим использование оператора цикла **for** на примере программы, которая для введенного с клавиатуры натурального числа **n** находит сумму квадратов всех натуральных чисел от 1 до **n**. Текст этой программы приведен в листинге 3.1.

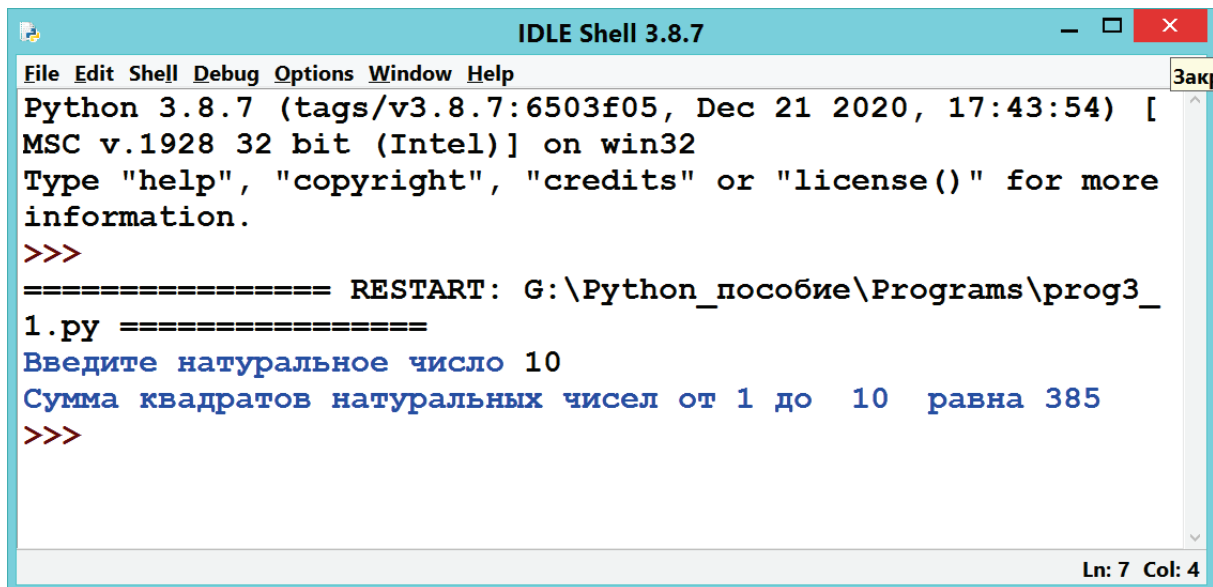
Листинг 3.1. Вычисление суммы квадратов натуральных чисел

```
# Вычисление суммы квадратов натуральных чисел
# Ввод исходных данных
n=int(input("Введите натуральное число"))
# Вычисление искомого значения
s=0
for i in range(1,n+1):
    s=s+i**2
# Вывод результатов вычислений
print("Сумма квадратов натуральных чисел от 1 до ",n," равна",s)
```

Данная программа начинается с ввода с клавиатуры натурального числа **n**. Затем в программе вводится еще одна переменная **s**, в которой будут суммироваться квадраты всех чисел в заданном диапазоне. Для того чтобы не исказить получаемый результат, ячейку, содержащую переменную **s** надо предварительно обнулить, что и делается в программе.

Затем для вычислений используется цикл с заранее известным числом повторений, в котором переменная цикла **i** меняется в диапазоне от 1 до **n+1**. При каждой очередной итерации (так называется повторение тела цикла) к значению переменной **s** прибавляется квадрат текущего значения переменной **i**. Таким образом, мы просуммируем квадраты всех натуральных чисел от 1 до **n** (если бы мы взяли диапазон изменения переменной от 1 до **n**, то получили бы ряд значений только до **n-1**).

Далее остается только вывести полученный результат на экран компьютера с помощью оператора **print**. Список вывода в данном случае состоит из 4 элементов. Это два текстовых фрагмента и значения двух переменных **n** и **s**. На рисунке 3.1 представлен экран с результатами работы данной программы для **n**, равного 10.



```
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [
MSC v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
===== RESTART: G:\Python_пособие\Programs\prog3_
1.py =====
Введите натуральное число 10
Сумма квадратов натуральных чисел от 1 до 10 равна 385
>>>
```

Рисунок 3.1 – Результат работы программы «Вычисление суммы квадратов натуральных чисел»

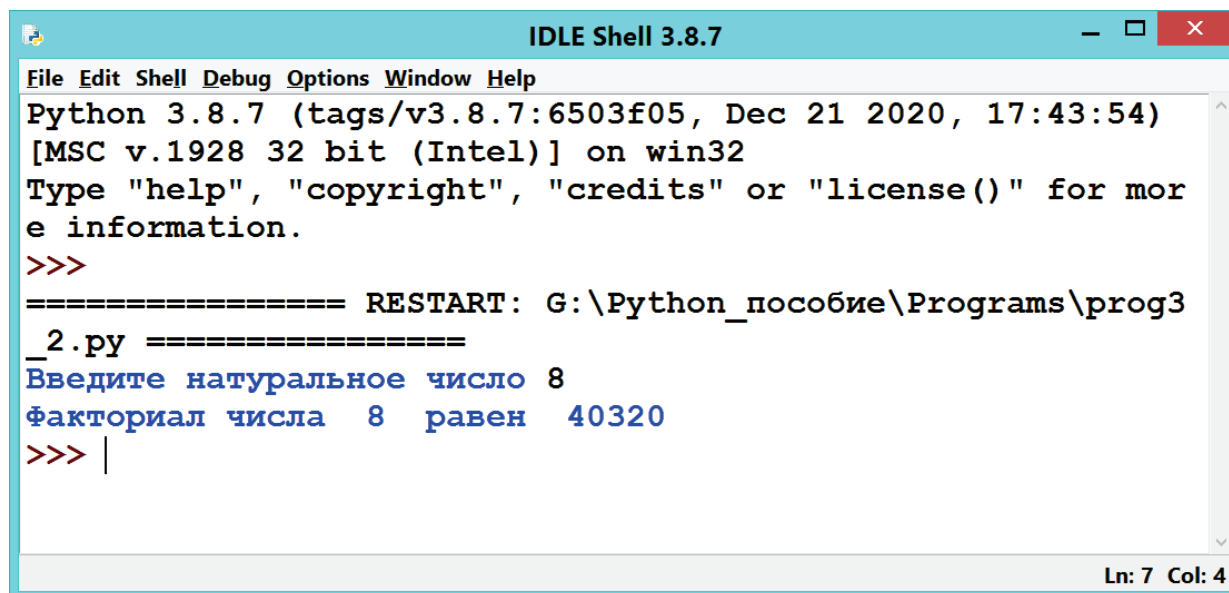
В следующей программе рассматривается вычисление факториала натурального числа. Факториалом числа n , как известно, называется произведение всех натуральных (то есть целых и положительных чисел) от 1 до n включительно. Текст программы, решающей эту задачу, представлен в листинге 3.2.

Листинг 3.2. Вычисление факториала натурального числа

```
# Вычисление факториала натурального числа
# Ввод исходных данных
n=int(input("Введите натуральное число"))
# Вычисление значения и вывод результатов
if n>0:
    f=1
    for i in range(1,n+1):
        f=f*i
    print("Факториал числа ",n," равен ",f)
else:
    print("Вводимое число должно быть больше или равно 0")
```

Программа начинается с ввода числа, для которого вычисляется факториал. Введенное с клавиатуры значение преобразуется к целому типу поскольку, как уже говорилось, понятие факториала имеет смысл только для целого положительного числа. Затем введенное число n проверяется с помощью уже знакомого нам условного оператора. Если n больше нуля, то задействуется та ветвь оператора `if`, где вычисляется факториал. Вычисления начинаются с того, что переменной f , в которой будет находиться искомый факториал, присваивается значение, равное единице. Это минимально

возможное значение факториала, так как факториал числа 1 (наименьшего натурального числа) равен единице. Затем при расчетах используется цикл `for`. На каждой итерации этого цикла значение переменной `f` умножается на текущее значение переменной `i`. В итоге работы цикла мы получаем искомую величину, которую остается только вывести на экран компьютера с помощью оператора `print`.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54)
[MSC v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for mor
e information.
>>>
===== RESTART: G:\Python_пособие\Programs\prog3
_2.py =====
Введите натуральное число 8
Факториал числа 8 равен 40320
>>> |
```

Рисунок 3.2 – Результат работы программы «Вычисление факториала натурального числа»

Если же в программу было введено отрицательное число или нуль, то задействуется другая ветвь условного оператора, в которой оператором `print` выводится сообщение об ошибке. Таким образом, из приведенного примера видно, что оператор цикла может входить в состав другого оператора (в данном случае оператора выбора). Результат работы программы для натурального числа 8 представлен на рисунке 3.2. Следует сказать, что в языке Python существует специальная функция для вычисления факториала под названием `factorial()`, но приведенная выше программа с одной стороны достаточно проста, а с другой наглядно демонстрирует возможности цикла `for`.

В следующей программе рассматривается создание таблицы для перевода температур из градусов Цельсия в температуру по шкале Кельвина и Фаренгейта. В этой программе рассматривается еще одна модификация цикла `for`. В предыдущих примерах в круглых скобках после ключевого слова `for` указывалось только два параметра – начальная и конечная границы изменения переменной цикла, а шаг изменения цикла по умолчанию был принят равным единице. Но можно в скобках указать и третий параметр, задающий шаг изменения переменной цикла, отличный от единицы, что и продемонстрировано в листинге 3.3.

Листинг 3.3. Создание таблицы для перевода температур

```
# Создание таблицы перевода температур
# по Цельсию в температуру по Кельвину и Фаренгейту
a=int(input("Введите начальную температуру в градусах Цельсия"))
b=int(input("Введите конечную температуру в градусах Цельсия"))
d=int(input("Укажите шаг изменения температуры"))
print("Температура по шкале:")
print("Цельсия Кельвина Фаренгейта")
for c in range(a,b+d,d):
    k=c+273.15
    f=c*9/5+32
    print('%7d %11.2f %12.2f'%(c,k,f))
```

Данная программа начинается, как обычно, с ввода исходных данных. С клавиатуры вводится начальная и конечная точки температурной шкалы a и b , а также шаг изменения этой шкалы d . Затем в программе создается цикл `for`. В заголовке цикла у функции `range` появился третий параметр – шаг изменения переменной цикла c . Также следует обратить внимание что значение переменной цикла изменяется не до b , а до $b+d$ (иначе последняя строка таблицы не будет выводиться на экран компьютера). В теле цикла по известным из курса физики формулам производится пересчет данных из одной температурной шкалы в две других, и вывод исходного значения и полученных при пересчете результатов на экран. В операторе вывода шаблон задается для трех величин. Первая выводимая величина является целой и шаблон для нее задается в формате `%xd`, где x – количество символов, отводимое под данную величину. В данном случае $x=7$. Шаблон вывода вещественных величин аналогичен тому, который был описан в главе 1 данного пособия для листинга 1.2. После кавычек указываются 3 переменных, значения которых должны быть отформатированы в соответствии с указанными в кавычках шаблонами. Поскольку эти переменные рассматриваются как единое выражение, они заключаются в круглые скобки, перед которыми ставится символ `%`. Пример работы программы для шкалы Цельсия от 0 до 30 градусов приведен на рисунке 3.3.

```
IDLE Shell 3.8.7
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51)
[MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for mor
e information.
>>>
===== RESTART: D:\Python_пособие\Programs\prog
3_3.py =====
Введите начальную температуру в градусах Цельсия 0
Введите конечную температуру в градусах Цельсия 30
Укажите шаг изменения температуры 5
Температура по шкале:
Цельсия      Кельвина      Фаренгейта
      0         273.15         32.00
      5         278.15         41.00
     10         283.15         50.00
     15         288.15         59.00
     20         293.15         68.00
     25         298.15         77.00
     30         303.15         86.00
>>>
```

Рисунок 3.3 – Результат работы программы «Создание таблицы для перевода температур»

Как уже говорилось в начале данной главы, помимо цикла `for` в языке Python существует еще один вид цикла – цикл `while`. Данный цикл рекомендуется использовать в том случае, когда число повторений цикла заранее не определено. Общий вид данного цикла следующий.

while условие:
 тело_цикла

В заголовке цикла **while** – ключевое слово языка Python, а **условие** – некоторое выражение, результатом вычисления которого должна быть логическая величина, имеющая значение **True** или **False**. **Тело_цикла** представляет собой блок, состоящий из одно или нескольких операторов языка Python. Операторы блока должны быть смещены на 4 позиции вправо по отношению к заголовку цикла.

Механизм работы данного вида цикла состоит в следующем. Перед началом работы цикла проверяется условие, находящееся в его заголовке. Если условие имеет значение **True**, то происходит выполнение тела цикла, а затем новая проверка условия. Если же значение условия равно **False**, то цикл завершает свою работу. Такой вид цикла называют циклом с предусловием.

Таким образом, если условие изначально оказалось ложным, то цикл может не выполниться ни разу. С другой стороны, если условие постоянно будет оставаться истинным, то цикл будет повторяться бесконечное количество раз. Такую ситуацию часто называют заикливанием программы. Для того, чтобы избежать заикливания, в теле цикла следует предусмотреть какую-либо возможность влиять на условие, находящееся в заголовке. Тогда через какое-то количество итераций значение условия изменится на False, и цикл все-таки завершит свою работу.

Рассмотрим работу с циклом while на следующем примере. В группе студентов требуется отобрать участников на проводимую в вузе олимпиаду по информатике и программированию. В олимпиаде могут принять участие все желающие за одним исключением. Участниками олимпиады должны быть хорошо успевающие студенты, которые по итогам последней сданной ими сессии имеют средний балл не ниже 4. Программа, которая сообщает пользователю количество участников олимпиады в данной группе после ввода с клавиатуры компьютера информации об успеваемости каждого претендента на участие, приведена в листинге 3.4.

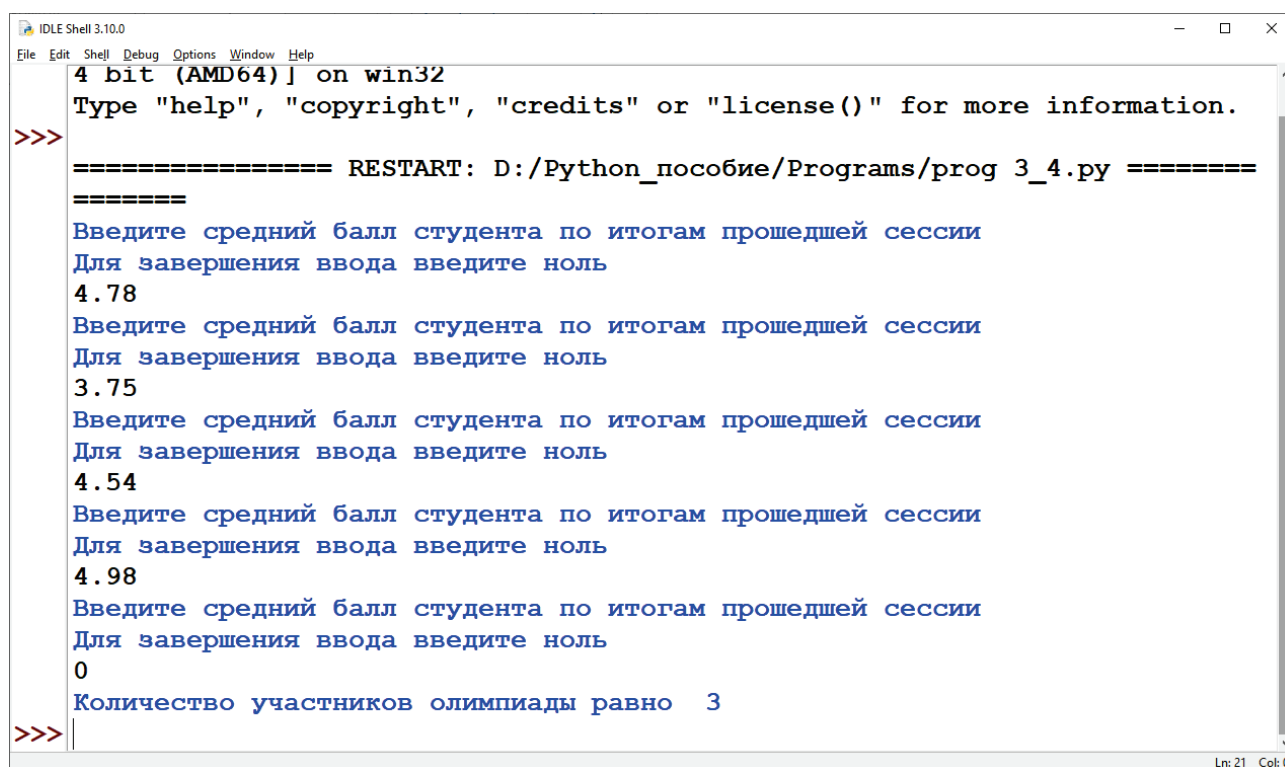
Листинг 3.4. Определение количества участников олимпиады

```
# Подсчет количества участников олимпиады  
# Ввод исходных данных  
n=0  
ball=0.001  
while ball!=0:  
    print("Введите средний балл студента по итогам прошедшей сессии")  
    print("Для завершения ввода введите ноль")  
    ball=eval(input())  
    if ball>=4:  
        n=n+1  
print("Количество участников олимпиады равно ",n)
```

В данной программе в самом ее начале обнуляется переменная n – счетчик количества участников олимпиады. Ввод данных о среднем балле каждого студента вводится в цикле while в переменную ball. В заголовке цикла определено, что цикл будет работать до тех пор, пока в очередной раз при вводе среднего балла не будет указано значение равное нулю. Понятно, что средний балл даже самого ленивого студента не может быть равен нулю (даже если этот студент завалит все экзамены, то его средний балл будет равен 2), поэтому ввод нулевого значения является признаком окончания работы цикла. Следует обратить внимание, что перед началом работы цикла переменной ball присваивается ненулевое значение, поскольку если ячейка памяти, содержащая эту переменную, в начале работы программы окажется пустой, то цикл так и не начнет свою работу.

Тело цикла содержит 4 оператора. Первый и второй операторы являются операторами вывода `print`, которые выводят на экран компьютера приглашение на ввод исходных данных. Обычно такое приглашение содержится в операторе `input`, но в данном случае приглашение занимает 2 строки, поэтому для его размещения пришлось использовать такой не совсем стандартный способ. Третьим оператором является `input`, который, собственно, и присваивает новое значение переменной `ball`. Поскольку приглашение на ввод вынесено в отдельные операторы, то круглые скобки в `input` остаются пустыми. Четвертым в теле цикла является условный оператор, который увеличивает счетчик количества участников на единицу если очередной введенный балл оказывается больше или равен четырем.

По окончании работы цикла остается только вывести полученный результат на экран компьютера. Возможные результаты работы программы приведены на рисунке 3.4.



```

IDLE Shell 3.10.0
File Edit Shell Debug Options Window Help
4 bit (AMD64) on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/Python_пособие/Programs/prog 3_4.py =====
=====
Введите средний балл студента по итогам прошедшей сессии
Для завершения ввода введите ноль
4.78
Введите средний балл студента по итогам прошедшей сессии
Для завершения ввода введите ноль
3.75
Введите средний балл студента по итогам прошедшей сессии
Для завершения ввода введите ноль
4.54
Введите средний балл студента по итогам прошедшей сессии
Для завершения ввода введите ноль
4.98
Введите средний балл студента по итогам прошедшей сессии
Для завершения ввода введите ноль
0
Количество участников олимпиады равно 3
>>>
Ln: 21 Col: 0

```

Рисунок 3.4 –Результат работы программы «Определение количества участников олимпиады»

Во многих языках программирования кроме двух рассмотренных выше видов циклов существует еще и третий – цикл с постусловием. Механизм работы такого цикла следующий. В начале всегда выполняется тело цикла (таким образом оно всегда будет выполнено по крайней мере один раз), а затем после тела цикла располагается заголовок с условием, истинность или ложность которого определяет будет ли цикл выполняться далее или его работа завершится. Такой вид цикла достаточно часто применяется в тех случаях, когда нужно защитить программу от неправильного ввода данных.

В языке Python такого вида цикла не существует, но можно искусственно создать его аналог, для чего используется цикл `while`. Выражение, находящееся после `while` в этом случае представляет собой константу, значение которой равно `True`. Таким образом тело цикла в этой конструкции гарантированно будет выполнено хотя бы один раз. Поскольку значение константы, находящейся в заголовке, в ходе работы цикла не изменяется, цикл и далее будет повторяться бесконечное количество раз. Но такой цикл все же можно завершить. Для этого в тело цикла вставляется условный оператор, в одной из ветвей которого находится вложенный оператор `break`. Этот оператор производит принудительное прерывание работы цикла. При очередном выполнении тела цикла в условном операторе включается его ветвь, содержащая `break`, и тогда цикл завершает свою работу.

Пример работы такого аналога цикла с постусловием приведен в листинге 3.5. Этот листинг содержит усовершенствованную программу вычисления факториала, которая имеет защиту от неверного ввода данных. Если пользователь попытается ввести отрицательное или нулевое значение факториала, то программа не закроется, а снова вернет пользователя к вводу данных. Такое повторение ввода будет происходить до тех пор, пока пользователь не введет наконец положительное значение.

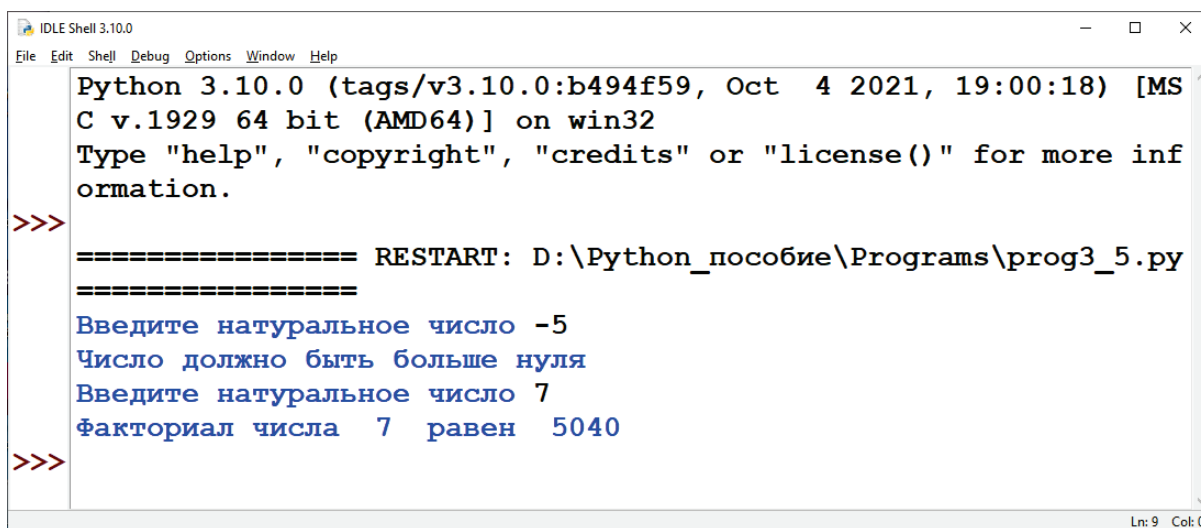
Листинг 3.5. Усовершенствованная программа вычисления факториала

```
# Модифицированная программа вычисления факториала
while True:
    n=int(input("Введите натуральное число"))
    if n>0:
        f=1
        for i in range(1,n+1):
            f=f*i
        print("Факториал числа ",n," равен ",f)
        break
    else:
        print("Число должно быть больше нуля")
```

Данная программа целиком состоит из одного цикла `while`. В теле цикла пользователю предлагается ввести натуральное число в переменную `n`. Затем значение `n` проверяется на предмет того, действительно ли он является натуральным, то есть больше нуля. Если это условие соблюдено, то по уже известному нам алгоритму значение факториала вычисляется и выводится на экран. Затем оператор `break` прерывает работу цикла и тем самым завершает работу программы в целом. Если же проверяемое условие не выполнено, то другая ветвь оператора выдаст сообщение об ошибке и цикл будет повторяться до тех пор, пока пользователь не введет натуральное число.

На рисунке 3.5 приведен пример работы данной программы. В начале пользователь ввел отрицательное число -5. Программа попросила его повторить

ввод данных. После ввода натурального числа 7 программа приняла это число и выдала искомый результат – значение факториала.



```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>>
===== RESTART: D:\Python_пособие\Programs\prog3_5.py
=====
Введите натуральное число -5
Число должно быть больше нуля
Введите натуральное число 7
Факториал числа 7 равен 5040
>>>
```

Рисунок 3.5 – Результат работы усовершенствованной программы вычисления факториала

Завершая данную главу, рассмотрим еще одну программу с оператором `for`. В этой программе содержится пример использования переменной-флага, значение которой определяет ход работы программы. Программа, приведенная в листинге 3.6 проверяет, является ли введенное с клавиатуры натуральное число простым. Как известно, натуральное число является простым в том случае, если он делится без остатка только на единицу и само на себя.

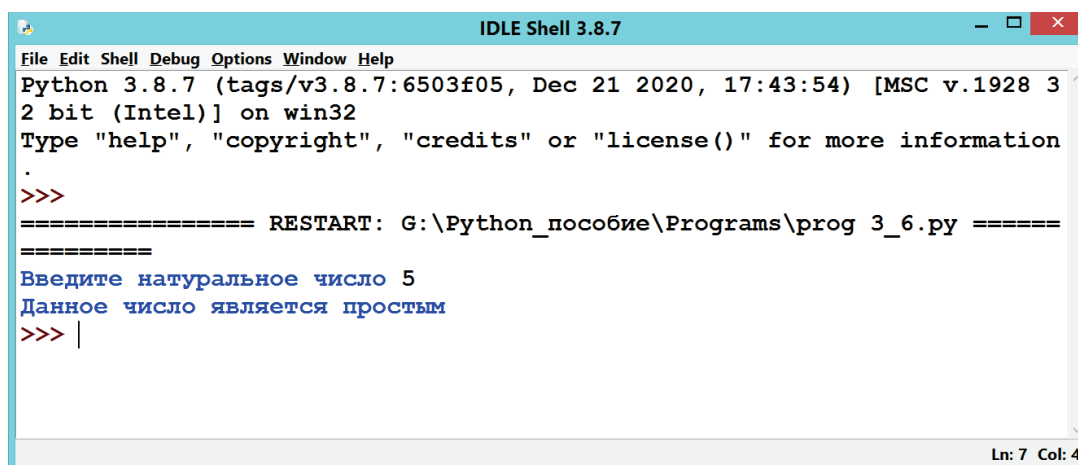
Листинг 3.6. Проверка, является ли число простым

```
# Проверяем, является ли число простым
n=int(input("Введите натуральное число"))
if (n==1) or (n==2) or (n==3):
    print("Данное число является простым")
else:
    flag=0
    for i in range(2,n):
        if n%i==0:
            flag=flag+1
    if flag==0:
        print("Данное число является простым")
    else:
        print("Данное число не является простым")
```

Данная программа начинается с ввода натурального числа `n`. Затем это число проверяется с помощью условного оператора. В одной ветви условного оператора выясняется не равно ли число 1, 2 или 3. Поскольку все эти числа являются простыми, то программа выдает соответствующее сообщение и на этом ее работа завершается.

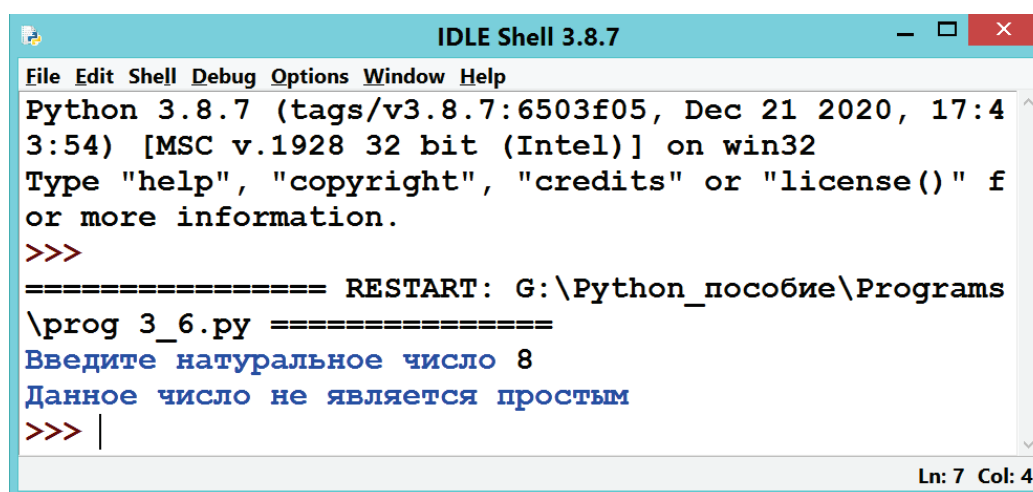
Если же указанное равенство не соблюдается, то включается другая ветвь условного оператора, находящаяся после ключевого слова `else`. В начале этой ветви происходит обнуление переменной-флага `flag`. Затем в этой ветви происходит работа вложенного цикла `for`, который проверяет делится ли без остатка введенное число `n` на какое-либо целое число, начиная с двух и заканчивая `n-1`. Если обнаруживается что число `n` действительно делится без остатка на одно из этих чисел, то значение переменной-флага увеличивается на единицу. После того как цикл завершает свою работу, проводится проверка состояния переменной `flag`. Если ее значение осталось равным нулю, то, следовательно, число `n` не делится ни на одно из чисел, относящихся к заданному в цикле диапазону и, следовательно, является простым. Если же значение `flag` отлично от нуля, то число `n` простым не является.

На рисунках 3.6 и 3.7 приводятся результаты работы программы при различных исходных данных. В одном случае введено простое число 5, а в другом число 8, которое не является простым. В обоих случаях программа выдает правильные результаты.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [MSC v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information
>>>
===== RESTART: G:\Python_пособие\Programs\prog 3_6.py =====
Введите натуральное число 5
Данное число является простым
>>> |
```

Рисунок 3.6 – Результат работы программы проверки, является ли число простым. Введено простое число



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [MSC v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: G:\Python_пособие\Programs\prog 3_6.py =====
Введите натуральное число 8
Данное число не является простым
>>> |
```

Рисунок 3.7 – Результат работы программы проверки, является ли число простым. Введено непростое число

Контрольные вопросы

1. Какие основные виды циклов используются в языке Python?
2. Какой вид цикла рекомендуется использовать в том случае, если число повторений цикла заранее известно?
3. Какой вид цикла рекомендуется использовать в том случае, если число повторений цикла заранее не определено?
4. Как в языке Python определить, где начинается и заканчивается тело цикла?
5. Какой символ обязательно должен находиться в конце заголовка цикла?
6. Допускается ли в языке Python вложение одного цикла в другой?
7. Чем в языке Python можно заменить отсутствующий в данном языке цикл с постусловием?
8. В каких случаях рекомендуется использовать в языке Python аналог цикла с постусловием?
9. Что такое заикливание программы? Как можно его избежать?
10. К какому типу не может относиться переменная цикла?

ГЛАВА 4. ФУНКЦИИ

При решении достаточно объемных и сложных задач из области программирования часто бывает удобно выделить внутри одной большой задачи отдельные более простые задачи и оформить их в виде отдельных автономных разделов программы, называемых функциями. Кроме того, нередко в разных частях программы приходится выполнять одну и ту же последовательность вычислений или действий. Такую последовательность также можно оформить в виде отдельной функции, которая вызывается из разных частей программы и используется по мере необходимости.

Для того чтобы функцию можно было использовать в программе, ее необходимо предварительно описать. Такое описание, как правило, располагается в начале программного модуля и выглядит следующим образом.

def имя_функции(параметры_функции):
тело_функции

В заголовке функции **def** – ключевое слово языка Python, **имя_функции** – собственное имя функции, которое дается по тем же правилам, что и имена переменных, **параметры_функции** – список параметров функции, перечисляемых через запятую. Параметры, указанные в описании функции, называются формальными. В конце заголовка всегда ставится двоеточие.

Тело функции – это блок, содержащий оператор или группу операторов, которые выполняются при вызове функции. Операторы блока должны быть сдвинуты вправо на 4 позиции относительно ее заголовка. В теле функции может вычисляться ее значение, которое возвращается в основную часть программы оператором `return`. При вычислении значения функции могут использоваться собственные переменные функции, которые называются локальными. Локальные переменные доступны только внутри данной функции. В отличие от них глобальные переменные не относятся только к какой-либо одной функции и доступны во всей программе.

При вызове функции из какого-либо места программы необходимо указать ее имя и список параметров, заключенных в круглые скобки. Параметры, используемые при вызове функции, называются фактическими. Количество и тип фактических параметров должны соответствовать ее формальным параметрам. При вызове функции формальные параметры заменяются на фактические, с использованием фактических параметров вычисляется значение функции, возвращаемое в программу.

В качестве примера применения функции рассмотрим программу, которая определяет число сочетаний из n по m . Эта величина подсчитывается по следующей формуле:

$$C_n^m = \frac{n!}{(n-m)!m!}.$$

Как видно из данной формулы, в ней 3 раза (только с разными параметрами) выполняется одно и то же действие – вычисление факториала. Поэтому в данной программе целесообразно оформить вычисление факториала в виде отдельной функции. Текст программы, решающей данную задачу, приведен в листинге 4.1.

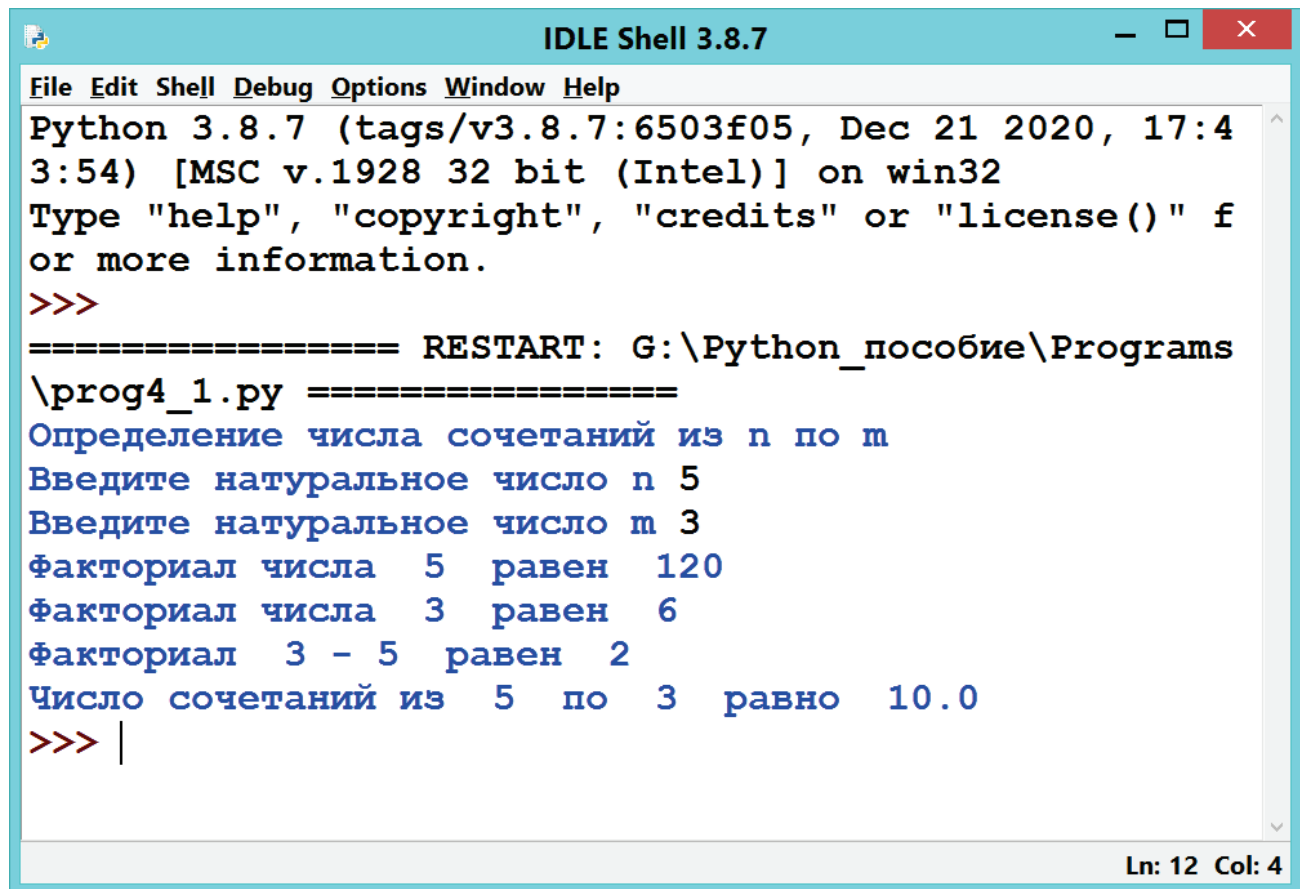
Листинг 4.1. Определение числа сочетаний из n по m

```
# Определение числа сочетаний  
# Описание функции  
def faktor(k):  
    f=1  
    for i in range(1,k+1):  
        f=f*i  
    return(f)  
# Ввод исходных данных  
print("Определение числа сочетаний из n по m")  
n=int(input("Введите натуральное число n"))  
m=int(input("Введите натуральное число m"))  
# Вычисление искомой величины  
fn=faktor(n)  
print ("Факториал числа ",n," равен ",fn)  
fm=faktor(m)  
print("Факториал числа ",m," равен ",fm)  
t=n-m  
ft=faktor(t)  
print("Факториал ",m,"-",n," равен ",ft)  
c=fn/(fm*ft)  
# Вывод результата  
print("Число сочетаний из ",n," по ",m," равно ",c)
```

В начале программы находится описание функции `faktor`, вычисляющей значение факториала. В заголовке функции после ключевого слова `def` и собственного имени функции `faktor` в круглых скобках указывается единственный параметр функции `k`. Затем в теле функции по уже известному нам по главе 3 данного пособия алгоритму вычисляется факториал, значение которого присваивается переменной `f`. Это значение будет возвращено в основную программу оператором `return`.

Затем уже в основной части программы вводятся исходные данные, которые присваиваются переменным `n` и `m`. Затем для нахождения факториала в программе трижды вызывается функция `faktor`. При каждом вызове функции вместо формального параметра `k` в нее подставляются формальные параметры: при первом вызове `n`, при втором `m`, а при третьем `t`, равное `n-m`. Полученные в результате вызова значения присваиваются вспомогательным переменным `fn`, `fm` и `ft`. Для контроля правильности этих промежуточных значений они выводятся на экран компьютера операторами `print`. Затем с использованием

этих вспомогательных переменных вычисляется искомое число сочетаний, которое также выводится на экран. На рисунке 4.1 приведен результат работы данной программы.



```
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:4
3:54) [MSC v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" f
or more information.
>>>
===== RESTART: G:\Python_пособие\Programs
\prog4_1.py =====
Определение числа сочетаний из n по m
Введите натуральное число n 5
Введите натуральное число m 3
Факториал числа 5 равен 120
Факториал числа 3 равен 6
Факториал 3 - 5 равен 2
Число сочетаний из 5 по 3 равно 10.0
>>> |
```

Рисунок 4.1 – Результат работы программы «Определение числа сочетаний из n по m»

Во многих языках программирования помимо функций существуют также автономные части программы, называемые процедурами. Их отличие от функций состоит в том, что они не должны возвращать какое-либо значение, но при этом они выполняют определенную последовательность действий, необходимую в данной программе. В языке Python не существует отдельного понятия процедуры, но при этом можно создавать функции, не возвращающие никакого значения, но производящие определенные действия. Фактически такие функции являются аналогами процедур.

Ниже приводится пример программы, использующей такую функцию-процедуру. Эта программа находит в заданном диапазоне все числа, кратные 5 и выводит их на экран компьютера.

Листинг 4.2. Поиск и вывод на экран чисел кратных 5

```
# Вывод чисел кратных 5
# Описание функции
def number(t):
```

```

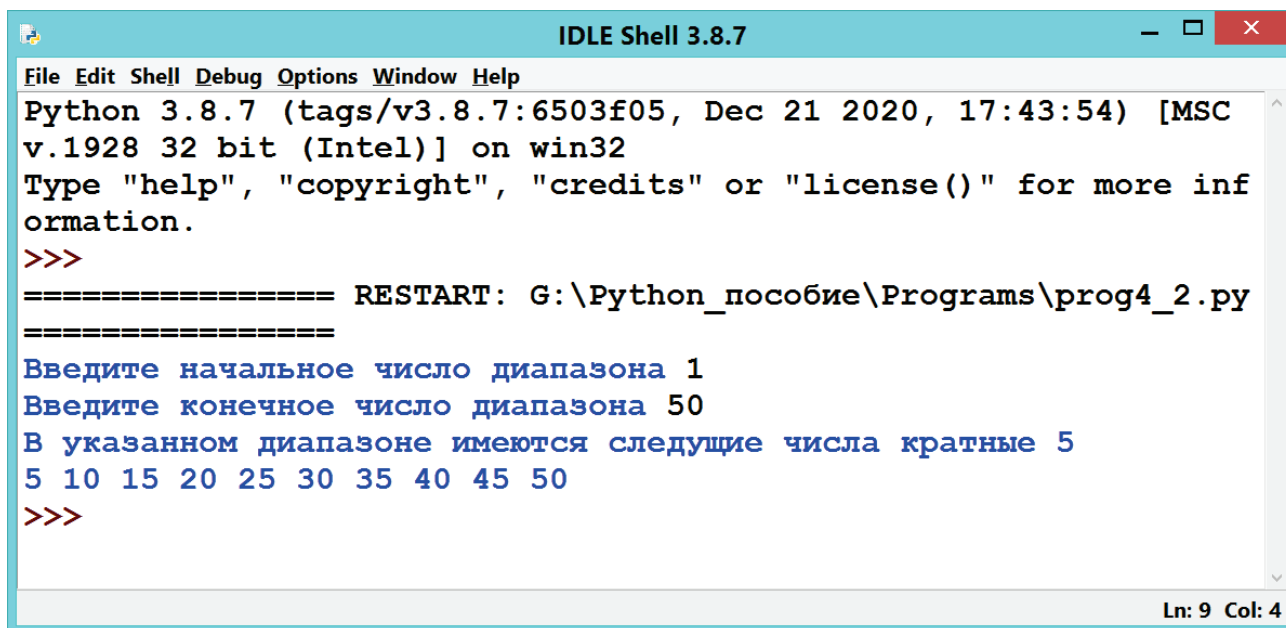
    if t%5==0:
        print(t,end=" ")
# Ввод исходных данных
a=int(input("Введите начальное число диапазона"))
b=int(input("Введите конечное число диапазона"))
# Вывод результата
print ("В указанном диапазоне имеются следующие числа кратные 5")
for i in range(a,b+1):
    number(i)

```

Данная программа начинается с описания функции number с параметром t. В теле функции с помощью сокращенного условного оператора if проверяется, делится ли число t без остатка на 5. Если число действительно делится на 5, то оно выводится на экран компьютера. Если же число не делится на 5, то функция никаких действий не выполняет и просто завершает свою работу.

По умолчанию оператор print после вывода находящейся в круглых скобках информации выполняет перевод строки. Но в данном случае желательно, чтобы вывод всех чисел производился в одну строку. Поэтому здесь используется такой прием как отмена перевода строки. Перевод строки осуществляется не отображающимся на экране символом, который является в этой строке завершающим. Завершающий строку символ содержится в переменной end. Значение этой переменной, как и любой другой, можно менять с помощью оператора присваивания. Поэтому в данной функции в операторе print вместо символа конца строки переменной end присваиваем значение, равное пробелу. В итоге после вывода числа не будет происходить перевод строки, а будет вставляться пробел.

Основная часть программы начинается с ввода исходных данных, которыми являются начальное и конечное число диапазона, в котором отбираются числа кратные 5. Далее сообщается о выводе этих чисел и организуется цикл типа for, в котором проверяются все числа из заданного диапазона. Проверка производится путем вызова функции number, параметром которой является текущее значение переменной цикла. В итоге работы программы на экран компьютера выводятся в одну строку все числа, кратные 5 и принадлежащие к указанному пользователем диапазону. Пример работы программы для диапазона от 1 до 50 показан на рисунке 4.2.



```
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [MSC
v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>>
===== RESTART: G:\Python_пособие\Programs\prog4_2.py
=====
Введите начальное число диапазона 1
Введите конечное число диапазона 50
В указанном диапазоне имеются следующие числа кратные 5
5 10 15 20 25 30 35 40 45 50
>>>
```

Рисунок 4.2 – Результат работы программы, находящей числа, кратные 5, в заданном диапазоне

По материалу, изложенному в предыдущих главах, нам известно, что условные операторы и операторы цикла можно вкладывать друг в друга. Но в языке Python допускается и вложение функций. Как правило, вложение функций осуществляется следующим образом. В описании внешней функции сразу после ее заголовка вставляется описание внутренней, вложенной функции, включающее ее заголовок и тело. Затем из тела внешней функции производится вызов внутренней функции, и значение этой внутренней функции возвращается во внешнюю функцию.

Рассмотрим использование вложенных функций на примере программы, которая находит все простые числа в заданном диапазоне. Текст этой программы приведен в листинге 4.3.

Листинг 4.3. Поиск простых чисел в заданном диапазоне

Поиск простых чисел в заданном диапазоне

Описание функции поиска

```
def search(x,y):
    def number(t):
        if (t==1) or (t==2) or (t==3):
            print(t,end=" ")
        else:
            flag=0
            for i in range(2,t):
                if (t%i)==0:
                    flag=flag+1
            if flag==0:
                print(t,end=" ")
    for j in range (x,y):
```

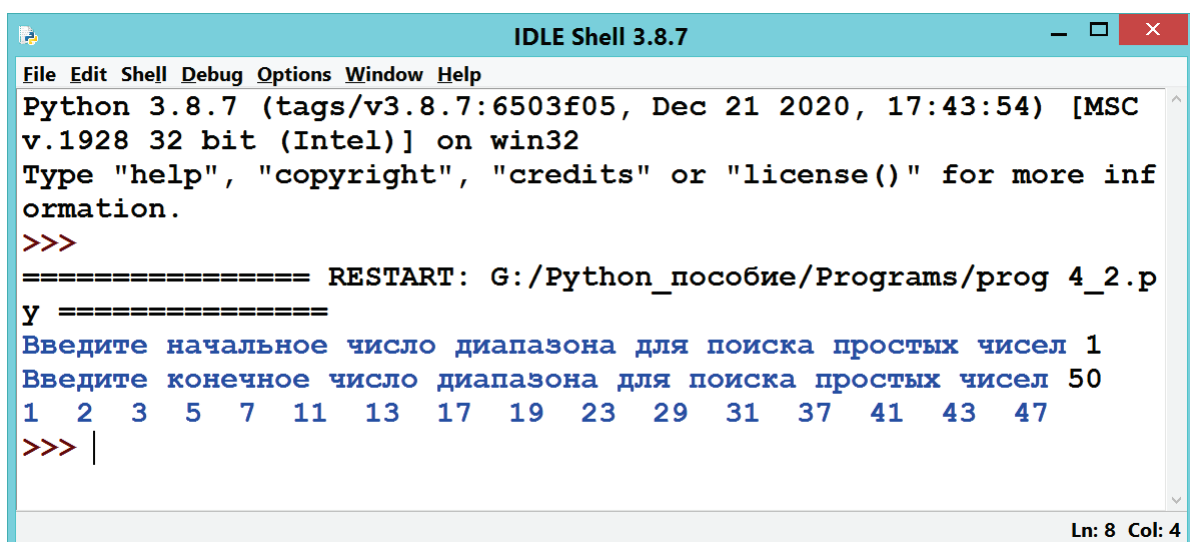
```

    number(j)
# Ввод исходных данных
a=int(input("Введите начальное число диапазона для поиска простых чисел"))
b=int(input("Введите конечное число диапазона для поиска простых чисел"))
# Вызов функции
search(a,b)

```

Данная программа начинается с описания функции `search`, которая и выполняет этот поиск. Функция `search` имеет два формальных параметра `x` и `y`, которые определяют границы диапазона поиска. Затем после заголовка этой внешней функции описывается вложенная функция `number`, имеющая единственный параметр `t`. Функция `number` проверяет число `t` для того, чтобы установить, является ли оно простым числом или нет. Алгоритм проверки числа по этому критерию уже описан в конце 3 главы в листинге 3.6, и поэтому повторять его здесь нет необходимости. Результатом же работы вложенной функции является вывод числа `t` на экран, но только в том случае, если это число является простым. В теле внешней функции `search` в цикле `for` производится перебор всех значений в заданном диапазоне с обращением к вложенной функции `number`. В итоге, результатом работы функции `search` будет вывод на экран компьютера всех простых чисел в указанном диапазоне.

В основной же части программы производится только ввод с клавиатуры величин `a` и `b`, определяющих границы диапазона для поиска простых чисел и обращение к функции `search` с этими параметрами. Результат поиска данной программой простых чисел в диапазоне от 1 до 50 показан на рисунке 4.3.



```

IDLE Shell 3.8.7
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [MSC
v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>>
===== RESTART: G:/Python_пособие/Programs/prog 4_2.p
y =====
Введите начальное число диапазона для поиска простых чисел 1
Введите конечное число диапазона для поиска простых чисел 50
1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
>>> |

```

Рисунок 4.3 – Результат выполнения программы «Поиск простых чисел в заданном диапазоне»

В языке Python, наряду с обычными и вложенными функциями, допускается использование рекурсивных функций. Рекурсией называется

обращение функции к самой себе. Процесс рекурсии происходит следующим образом: сначала происходит вызов некоторой функции. Затем, еще до завершения своей работы, эта же функция вызывает саму себя, и в результате в памяти компьютера появляется второй экземпляр той же функции; второй экземпляр в ходе работы создает третий, и так далее. Естественно, что такой процесс может продолжаться до бесконечности, поэтому для решения какой-либо конкретной задачи в рекурсивной функции обязательно должен содержаться условный оператор, одна из ветвей которого обеспечивает прерывание этой бесконечной цепочки и завершение работы всех экземпляров функции. Внутри этого оператора и должен находиться вызов функции.

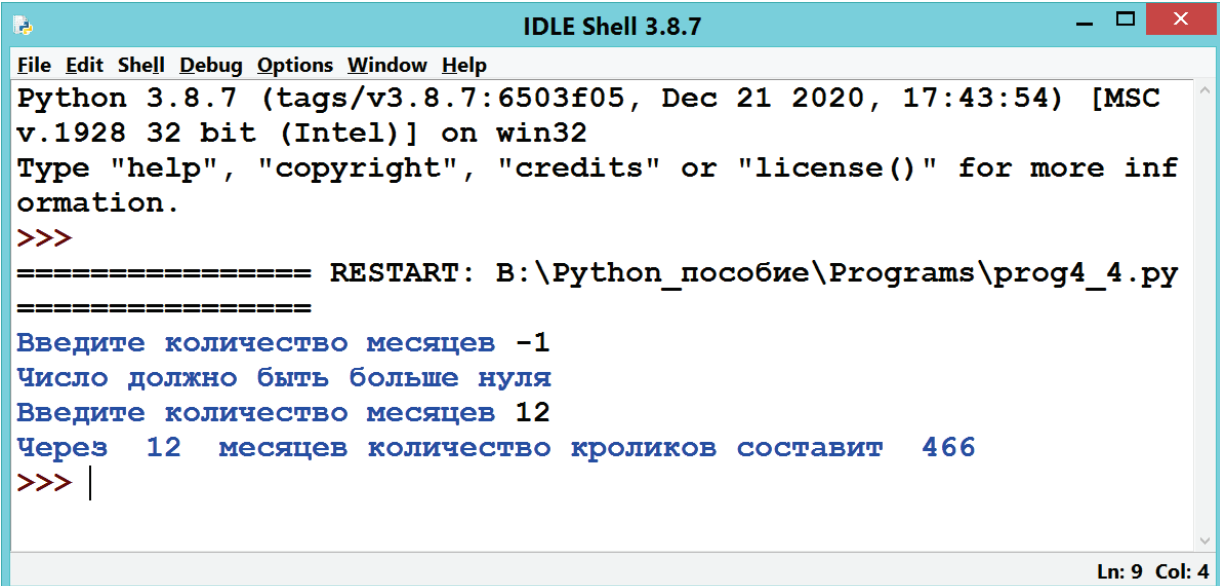
Пример использования рекурсии – это решение задачи, поставленной в XIII веке итальянским математиком Леонардо Фибоначчи. Условия задачи таковы. Фермер привез в свою усадьбу пару новорожденных кроликов. Эти кролики через 2 месяца дали потомство – еще одну пару кроликов, затем они стали приносить приплод по одной паре кроликов каждый месяц. Так же каждая последующая рожденная на ферме пара кроликов начинает давать потомство через 2 месяца и затем приносит новую пару каждый месяц. Ферма окружена крепкой высокой оградой, так что хищники кроликам не грозят. Все кролики здоровы, не болеют и не умирают, могут размножаться беспрепятственно. Требуется определить, сколько кроликов будет на ферме через год. Решение этой задачи представлено в листинге 4.4.

Листинг 4.4. Программа «Числа Фибоначчи»

```
# Пример работы с рекурсивной функцией
# Описание функции
def fibon(n):
    if n==0:
        return 1
    elif n==1:
        return 1
    else:
        return fibon(n-1)+fibon(n-2)
# Ввод исходных данных
while True:
    k=int(input("Введите количество месяцев"))
    if k>=0:
        # Вызов функции
        r=fibon(k)
        # Вывод результата
        print("Через ",k," месяцев количество кроликов составит ",r*2)
        break
    else:
print("Число должно быть больше нуля")
```

Задача Фибоначчи решается следующим образом. Для решения данной задачи используем рекурсивную функцию `fibon(n)`, которая определяет количество пар кроликов через n месяцев после начала года. В начале года существовала только одна пара кроликов, то есть `fibon(0)=1`, через месяц их по-прежнему одна пара, следовательно, `fibon(1)=1`. Через 2 месяца количество пар кроликов увеличивается на 1, то есть на столько, сколько их было в начале года. Таким образом, `fibon(2)=2` или `fibon(2)=fibon(1)+fibon(0)`. Продолжая наши рассуждения, получаем следующую зависимость: `fibon(n)=fibon(n-1)+fibon(n-2)`. По этой формуле можно вычислить значение функции для любого n , кроме $n=0$ и $n=1$, для которых значения уже известны.

Программа начинается с описания функции `fibon(n)`, механизм работы которой рассмотрен выше. Затем в основной программе идет ввод исходных данных, вызов функции `fibon` для выполнения необходимых вычислений и вывод полученного результата на экран компьютера. Все операторы, выполняющие эти действия, находятся внутри цикла `while`, который в данном случае является аналогом цикла с постусловием и выполняет защиту программы от ошибочного ввода данных. На рисунке 4.4 показаны результаты работы программы «Числа Фибоначчи» как при правильном, так и при неправильном вводе исходных данных.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [MSC
v.1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>>
===== RESTART: B:\Python_пособие\Programs\prog4_4.py
=====
Введите количество месяцев -1
Число должно быть больше нуля
Введите количество месяцев 12
Через 12 месяцев количество кроликов составит 466
>>> |
```

Рисунок 4.4 – Результат работы программы «Числа Фибоначчи» при неправильном и правильном вводе данных

Контрольные вопросы

1. Укажите порядок описания функции в языке Python.
2. С какого ключевого слова начинается описание функции?
3. Обязательно ли функция в языке Python должна иметь аргументы?

4. Какой оператор языка Python позволяет вернуть значение функции?
5. Обязательно ли функция в языке Python должна возвращать результат?
6. Чем отличаются формальные и фактические параметры функции?
7. Какие переменные называются локальными?
8. Какие переменные называются глобальными?
9. Что такое рекурсивная функция?
10. Что такое вложенная функция?

ГЛАВА 5. СПИСКИ

В большинстве современных языков программирования существует структура, называемая массивом. Массив представляет собой упорядоченную последовательность однородных элементов. Что касается языка Python, то в нем имеется похожая структура, называемая списком. Основное отличие списка от массива заключается в том, что список не обязательно должен быть однородным и может содержать элементы, относящиеся к разным типам.

Список создается следующим образом. После имени списка и знака равенства в квадратных скобках через запятую перечисляются элементы списка. Общий вид такого описания:

```
list=[k1,k2,k3,...]
```

Здесь list – имя списка, k1, k2 и так далее – это элементы списка.

Такое описание используется, когда заполнение списка происходит непосредственно в программе. Если же содержание списка заранее неизвестно, то можно создать пустой список следующим оператором

```
list=[]
```

В этом случае при создании списка нужно указывать только его имя, а заполнение списка можно производить, например, путем ввода его значений с клавиатуры. Для этого можно использовать метод append. Название метода указывается после имени списка и отделяется от него точкой. После ключевого слова append ставятся круглые скобки, в которых содержится либо имя вводимого элемента, либо команда ввода данных с клавиатуры.

Если все элементы списка относятся к одному типу, то можно провести их сортировку методом sort. Метод sort указывается после имени списка и отделяется от него точкой, а после ключевого слова sort ставятся круглые скобки. По умолчанию сортировка производится в порядке возрастания. Если нужно отсортировать элементы списка в порядке их убывания, то в круглых скобках указывается команда reverse=True.

Ниже приводится листинг программы, в которой массив заполняется данными, введенными с клавиатуры, а затем сортируется и в порядке возрастания элементов, и в порядке их убывания.

Листинг 5.1. Ввод данных в список с клавиатуры и сортировка списка

```
# Создание списка и его сортировка
s=[]
# Заполнение списка данными с клавиатуры
n=int(input("Введите количество элементов списка"))
for i in range(0,n):
    s.append(int(input("Введите очередной элемент (число)")))
# Вывод исходного списка
print("Неупорядоченный список")
for i in range(0,n):
```

```

    print(s[i],end=" ")
print("\nСписок упорядоченный по возрастанию")
s.sort()
for i in range(0,n):
    print(s[i],end=" ")
print("\nСписок упорядоченный по убыванию")
s.sort(reverse=True)
for i in range(0,n):
    print(s[i],end=" ")

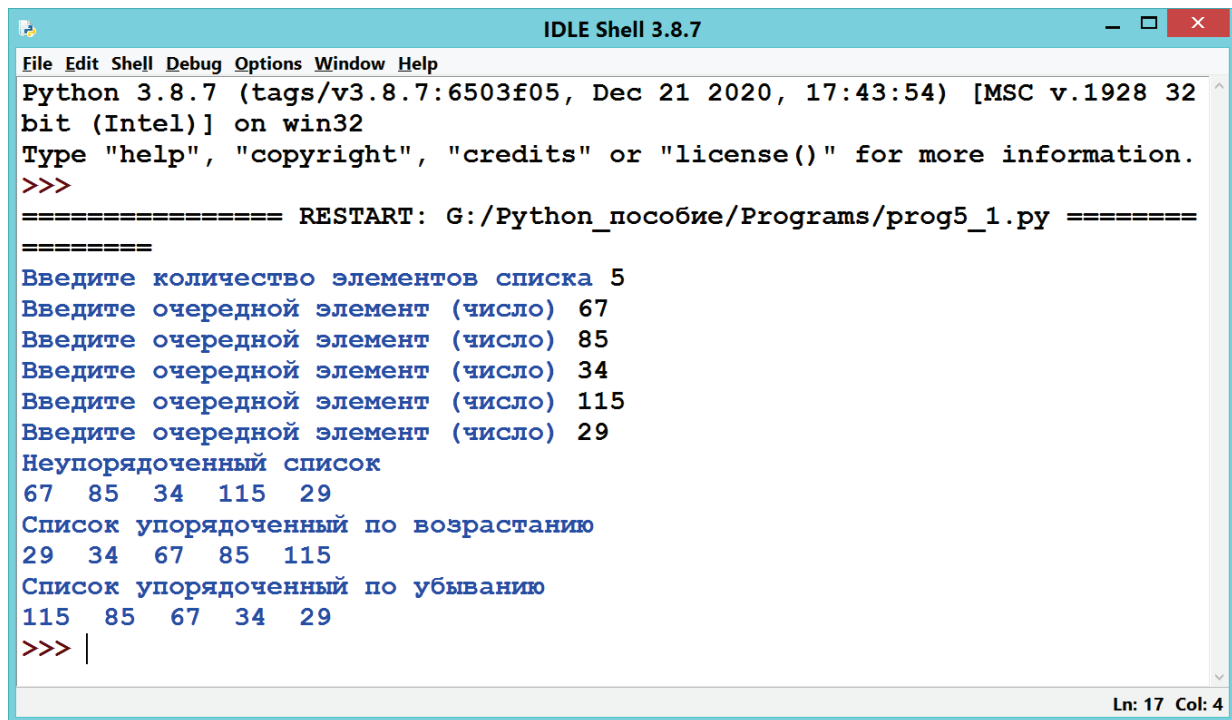
```

В данной программе сначала создается пустой список, а затем с клавиатуры вводится количество элементов этого списка. Далее используется цикл с заранее известным количеством повторений, в котором производится заполнение списка данными, введенными с клавиатуры с помощью метода `append`. В этой программе создается однородный список, содержащий только целочисленные элементы, поэтому для ввода значений элементов применяется уже знакомый нам оператор `int(input)`, который в данном случае выступает в качестве аргумента для метода `append`. Затем этот список также в цикле с заранее известным числом повторений выводится оператором `print` в одну строку на экран компьютера.

Далее с помощью метода `sort` производится сортировка списка в порядке возрастания значений его элементов. Отсортированный список также в цикле с заранее известным количеством повторений в одну строку выводится на экран компьютера.

Затем с помощью того же метода `sort` с параметром `reverse=True` выполняется сортировка элементов списка в порядке их убывания. Результат этой сортировки также выводится в одну строку на экран компьютера.

Пример работы программы, в котором происходит заполнения списка 5 элементами, введенными с клавиатуры, а затем его сортировка приведен на рисунке 5.1.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [MSC v.1928 32
bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: G:/Python_пособие/Programs/prog5_1.py =====
=====
Введите количество элементов списка 5
Введите очередной элемент (число) 67
Введите очередной элемент (число) 85
Введите очередной элемент (число) 34
Введите очередной элемент (число) 115
Введите очередной элемент (число) 29
Неупорядоченный список
67 85 34 115 29
Список упорядоченный по возрастанию
29 34 67 85 115
Список упорядоченный по убыванию
115 85 67 34 29
>>> |
```

Рисунок 5.1. Результат работы программы «Заполнение списка с клавиатуры и сортировка списка»

Если в приведенной выше программе все элементы списка относились к одному типу, то далее мы рассмотрим пример программы, где создается список, элементы которого относятся к разным типам – вещественному и строковому. В этой программе нужно создать список, в котором содержатся итоги сдачи студентами сессии. В списке должны быть как строковые элементы (фамилия, имя и отчество каждого студента), так и вещественные (средний балл каждого студента по итогам сдачи сессии). Текст программы, выполняющей поставленную задачу, содержится в листинге 5.2.

Листинг 5.2. Листинг программы «итоги сессии»

```
# Подведение итогов сессии
# Создание списков
spisok=[]
ball=[]
# Инициализация переменных цикла
i=0
j=0
# Ввод исходных данных
while True:
    spisok.append(input("Введите ФИО студента"))
    if spisok[i]=="0":
        break
    else:
        i=i+1
```

```

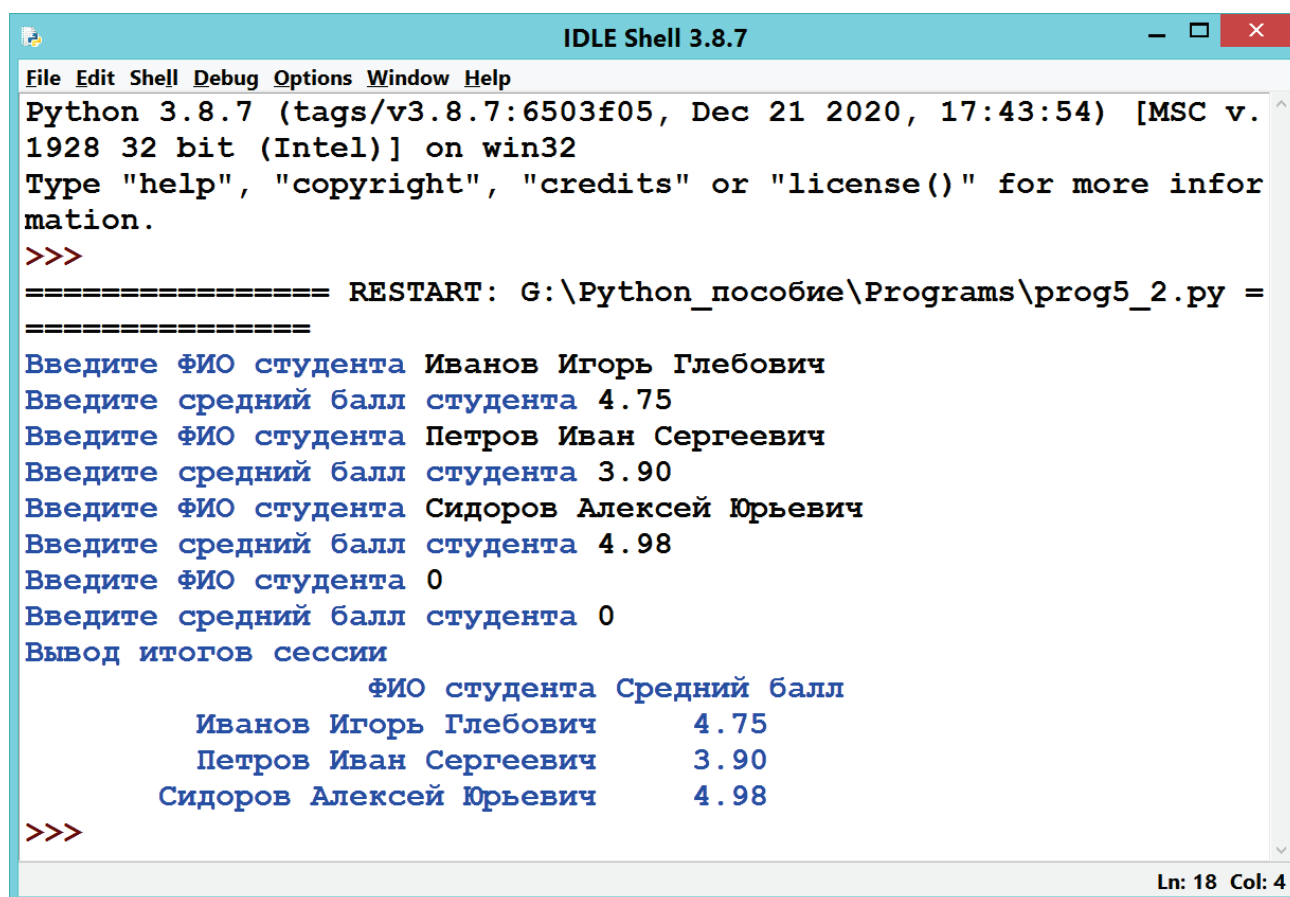
ball.append(eval(input("Введите средний балл студента")))
if ball[j]==0:
    break
else:
    j=j+1
# Создание итогового списка и заполнение его данными
vedom=[]
for k in range(0,len(spisok)-1):
    vedom.append(spisok[k])
    vedom.append(ball[k])
# Вывод содержимого итогового списка
print("Вывод итогов сессии")
print('%30s %10s' %("ФИО студента", "Средний балл"))
for k in range (0,len(vedom),2):
    print('%30s %8.2f' %(vedom[k],vedom[k+1]))

```

В начале программы создаются два однородных списка. Список `spisok` будет содержать только строковые элементы – ФИО студентов, а список `ball` только вещественные – средний балл каждого студента. Затем производится инициализация переменных, которые будут использованы для заполнения списков данным. Само заполнения списка данными производится с помощью цикла `while True`, являющегося аналогом цикла с постусловием. Признаком окончания работы цикла (и заполнения списка) является ввод очередного значения, равного нулю.

Далее создается итоговый список `vedom`, который заполняется поочередно данными из списков `spisok` и `ball`. Заполнение производится в цикле с заранее определенным количеством повторений. Количество повторений определяется с помощью стандартной функции `len`, которая находит длину списка. Это количество должно быть меньше на единицу, чем длина списка, так как последнее вводимое нулевое значение является только признаком окончания ввода данных. Затем получившийся список выводится на экран компьютера также с помощью цикла с заранее известным количеством повторений. При выводе данных в каждой строке отображается 2 значения – ФИО очередного студента и его средний балл. Для отображения строковых величин используется шаблон `%ns`, где `n` – количество символов, резервируемых под соответствующую строковую переменную.

Возможный результат работы программы «Итоги сессии» показан на рисунке 5.2.



```
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [MSC v.
1928 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more infor
mation.
>>>
===== RESTART: G:\Python_пособие\Programs\prog5_2.py =
=====
Введите ФИО студента Иванов Игорь Глебович
Введите средний балл студента 4.75
Введите ФИО студента Петров Иван Сергеевич
Введите средний балл студента 3.90
Введите ФИО студента Сидоров Алексей Юрьевич
Введите средний балл студента 4.98
Введите ФИО студента 0
Введите средний балл студента 0
Вывод итогов сессии
                ФИО студента Средний балл
        Иванов Игорь Глебович      4.75
        Петров Иван Сергеевич      3.90
        Сидоров Алексей Юрьевич    4.98
>>>
```

Ln: 18 Col: 4

Рисунок 5.2 – Результат работы программы «Итоги сессии»

Следующая программа демонстрирует возможность создания списка с выбором для него не всех элементов, а только необходимых. В главе 3 была создана программа, которая подсчитывает количество студентов, которые могут принять участие в вузовской олимпиаде. При этом информация была обезличенной. Организаторам олимпиады конечно удобнее иметь перед глазами список, в котором видно не только количество участников, но и их фамилии, имена и отчества, а также средний балл по итогам сдачи последней сессии. Принцип отбора остается прежним, то есть в олимпиаде могут участвовать студенты, средний балл которых не ниже 4. Текст программы представлен в листинге 5.3.

Листинг 5.3. Программа «Отбор участников олимпиады»

```
# Отбор участников олимпиады
# Создание списков
spisok=[]
ball=[]
# Инициализация переменных цикла
i=0
j=0
# Ввод исходных данных
while True:
```

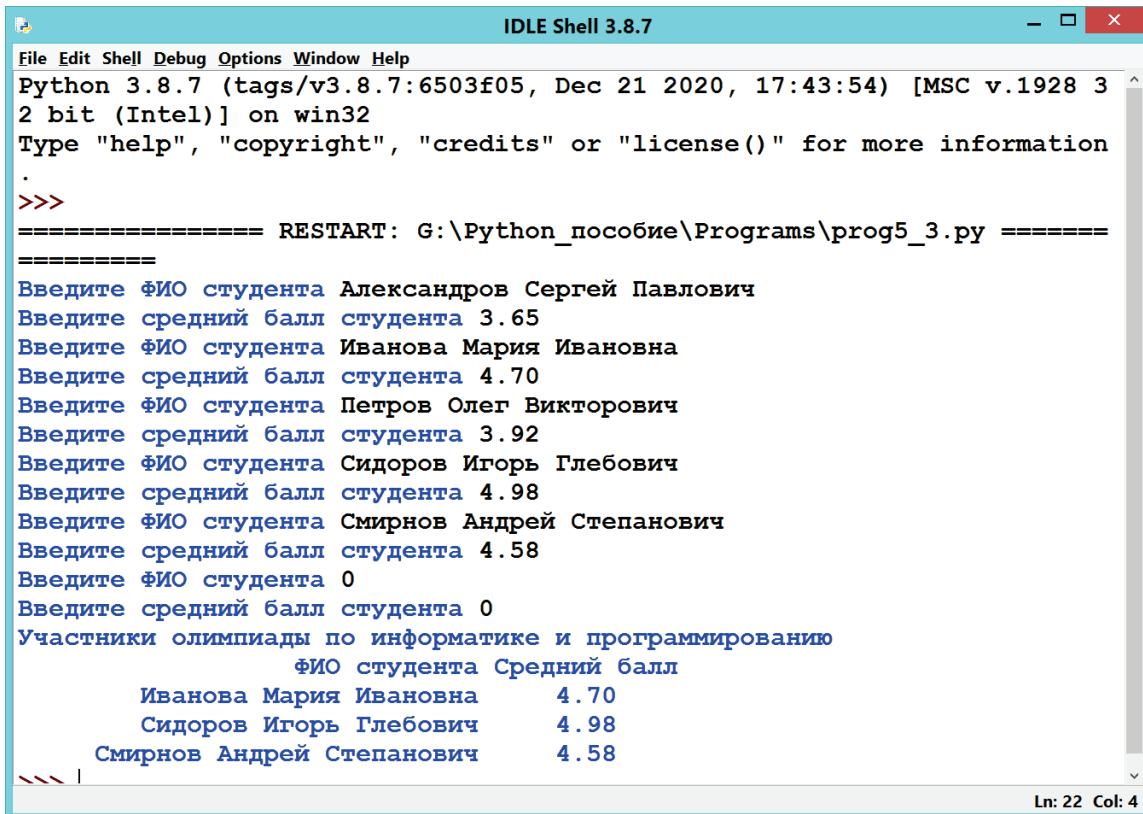
```

spisok.append(input("Введите ФИО студента"))
if spisok[i]=="0":
    break
else:
    i=i+1
ball.append(eval(input("Введите средний балл студента")))
if ball[j]==0:
    break
else:
    j=j+1
# Создание новых списков
new_spisok=[]
new_ball=[]
# Отбор студентов с высоким средним баллом
for t in range(0,len(ball)):
    if ball[t] >= 4.0:
        new_ball.append(ball[t])
        new_spisok.append(spisok[t])
# Создание итогового списка и заполнение его данными
vedom=[]
for k in range(0,len(new_spisok)):
    vedom.append(new_spisok[k])
    vedom.append(new_ball[k])
# Вывод содержимого итогового списка
print("Участники олимпиады по информатике и программированию")
print('%30s %10s' %("ФИО студента","Средний балл"))
for k in range (0,len(vedom),2):
    print('%30s %8.2f' %(vedom[k],vedom[k+1]))

```

Сначала в программе создаются 2 списка `spisok` и `ball`. В первый из них вводятся ФИО студентов, а во второй – их средние баллы аналогично тому, как это было сделано в предыдущей программе.

Затем создаются еще 2 списка `new_spisok` и `new_ball`. В них с помощью цикла с заранее заданным количеством повторений переносятся данные из предыдущих 2 списков. Но в цикле содержится условный оператор, который производит вставку данных в новые списки только при условии, что значение очередного элемента из старого списка `ball` не менее 4.



```
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:43:54) [MSC v.1928 3
2 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information
.>>>
===== RESTART: G:\Python_пособие\Programs\prog5_3.py =====
=====
Введите ФИО студента Александров Сергей Павлович
Введите средний балл студента 3.65
Введите ФИО студента Иванова Мария Ивановна
Введите средний балл студента 4.70
Введите ФИО студента Петров Олег Викторович
Введите средний балл студента 3.92
Введите ФИО студента Сидоров Игорь Глебович
Введите средний балл студента 4.98
Введите ФИО студента Смирнов Андрей Степанович
Введите средний балл студента 4.58
Введите ФИО студента 0
Введите средний балл студента 0
Участники олимпиады по информатике и программированию
      ФИО студента Средний балл
      Иванова Мария Ивановна      4.70
      Сидоров Игорь Глебович      4.98
      Смирнов Андрей Степанович    4.58
```

Рисунок 5.3 – Результат работы программы «Отбор участников олимпиады»

Затем оба новых списка `new_ball` и `new_spisok` объединяются в общий список `vedom` аналогично тому, как это было сделано в предыдущей программе. Этот общий список выводится на экран компьютера с помощью цикла с заранее известным количеством повторений, и на этом программа завершает свою работу.

Возможный результат работы программы показан на рисунке 5.3.

Контрольные вопросы

1. Должны ли все элементы списка относиться к одному и тому же типу?
2. Чем списки отличаются от массивов?
3. Можно ли создать список, заранее не указывая его размерность?
4. Какие скобки (круглые, квадратные или фигурные) нужно поставить после имени списка, чтобы указать количество элементов в нем?
5. С какого числа начинается нумерация элементов списка?
6. Какой метод используется для сортировки элементов списка по возрастанию?
7. Какой дополнительный параметр и с каким значением нужно использовать для сортировки элементов списка по убыванию?
8. Каким образом можно удалить из списка элемент с определенным индексом?
9. Как добавить новый элемент в конец списка?
10. Как удалить из списка элемент с определенным значением?

ЗАКЛЮЧЕНИЕ

Данное учебное пособие ставит своей целью обучить студентов только основным, базовым структурам и концепциям программирования на языке Python. За рамками пособия остались такие вопросы, как объектно-ориентированное программирование в Python, работа с такими структурами данных, как множества и словари, декораторы классов и функций и ряд других. Также в пособии не рассмотрены возможности работы с компьютерной графикой и создание анимированных изображений с использованием таких пакетов, как Tkinter и PyGame, создание диаграмм и графиков (в том числе объемных и анимированных) с помощью библиотеки Matplotlib.

Но, тем не менее, автор выражает надежду, что изучение данного пособия может помочь студентам в освоении такого универсального перспективного языка программирования, каким является язык Python.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Васильев, А. Н. Python на примерах. Практический курс программирования / А. Н. Васильев. – СПб.: Наука и техника, 2016. – 432 с. – Текст: непосредственный.
2. Доусон, М. Програмируем на Python / М. Доусон. – СПб.: Питер, 2014. – 416 с. – Текст: непосредственный.
3. МакГрат, М. Программирование на Python для начинающих / М. МакГрат. – М.: Эксмо, 2015. – 192 с. – Текст: непосредственный.
4. Мэтиз, Э. Изучаем Python / Э. Мэтиз. – 2-е изд. – СПб.: Питер, 2017. – 496 с. – ISBN 978-5-4461-0479-6. – Текст: непосредственный.
5. Прохоренок, Н. А. Python 3 и PyQt 5. Разработка приложений / Н. А. Прохоренок, В. А. Дронов. – СПб.: БХВ-Петербург, 2016. – 832 с. – ISBN 978-5-9775-3648-6. – Текст: непосредственный.
6. Прохоренок, Н. А. Python 3. Самое необходимое / Н. А. Прохоренок, В. А. Дронов. – СПб.: БХВ-Петербург, 2019. – 608 с. – ISBN 978-5-9775-3994-4. – Текст: непосредственный.
7. Шоу, З. Легкий способ выучить Python / З. Шоу ; пер. с англ. М. А. Райтман. – М.: Издательство «Э», 2017. – 352 с. – ISBN 978-5-699-98251-6. – Текст: непосредственный.

Учебное издание

Маслобоев Артур Николаевич

**Языки и методы
программирования**

**Основы программирования
на языке Python**

Учебное пособие

Редактор и корректор Д. А. Романова
Техн. редактор Д. А. Романова

Темплан 2022 г., поз. 5052/22

Подписано к печати 20.02.2024.	Формат 60x84/16.	Бумага тип № 1.
Печать офсетная.	Печ. л. 4,3.	Уч.-изд. л. 4,3.
Тираж 30 экз.	Изд. №5052/22.	Цена «С». Заказ №

Ризограф Высшей школы технологии и энергетики СПбГУПТД,
198095, Санкт-Петербург, ул. Ивана Черных, 4.