МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ИМ. В. Г. ШУХОВА

Р. У. Стативко

**Алгоритмы и Структуры Данных**

Учебное пособие

Белгород

2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ИМ. В. Г. ШУХОВА

Р. У. Стативко

Алгоритмы и структуры данных

*Утверждено ученым советом университета в качестве*

*учебного пособия для студентов 1-го курса очной и заочной форм обучения направления подготовки 09.03.02 – Информационные системы и технологии*

Белгород

2022

УДК 004.42(07)

ББК 32.973-018я7

С 78

Рецензенты:

Кандидат технических наук, доцент Белгородского государственного технологического университета им. В.Г. Шухова *В.А. Порхало*

Кандидат технических наук, доцент Белгородского государственного национального исследовательского университета *Ю.Г. Чашин*

С 78

Стативко, Р. У.

Алгоритмы и структуры данных: учеб. пособие/ Р. У. Стативко. – Белгород: Изд-во БГТУ, 2020. – 79 с.

В учебном пособии изложен материал, способствующий прочному и сознательному освоению основ алгоритмизации и программирования, формированию практических умений - профессиональных, учебных, интеллектуальных, необходимых студенту согласно соответствующим компетенциям.

Учебное пособие предназначено для студентов 1-го курса очной и заочной форм обучения направления подготовки 09.03.02 – Информационные системы и технологии.

Данное издание публикуется в авторской редакции.

**УДК 004.42(07)**

**ББК 32.973-018я7**

Белгородский государственный

технологический университет

(БГТУ) им. В. Г. Шухова, 2020

**Оглавление**

[ВВЕДЕНИЕ 6](#_Toc119622192)

[1. Разработка структуры программы и модульное программирование 8](#_Toc119622193)

[1.1. Цель модульного программирования 8](#_Toc119622194)

[2. Постановка и алгоритмизация задач 8](#_Toc119622195)

[2.1. Понятие алгоритма 8](#_Toc119622196)

[2.2. Способы описания алгоритмов 9](#_Toc119622197)

[2.3. Структурные схемы алгоритмов 13](#_Toc119622198)

[Контрольные вопросы 17](#_Toc119622199)

[Задания для самопроверки 17](#_Toc119622200)

[3. Основы языка 17](#_Toc119622201)

[3.1. Алфавит языка 17](#_Toc119622202)

[3.2. Структура программы 19](#_Toc119622203)

[Контрольные вопросы 20](#_Toc119622204)

[Задания для самопроверки 20](#_Toc119622205)

[4. Типы данных 20](#_Toc119622206)

[4.1. Целые типы 21](#_Toc119622207)

[4.2. Вещественные типы 22](#_Toc119622208)

[Контрольные вопросы 24](#_Toc119622209)

[Задания для самопроверки 24](#_Toc119622210)

[5. Линейные алгоритмы 24](#_Toc119622211)

[5.1. Оператор присваивания 24](#_Toc119622212)

[5.3. Простейший ввод и вывод 26](#_Toc119622213)

[Контрольные вопросы 32](#_Toc119622214)

[Задания для самопроверки 32](#_Toc119622215)

[6. Разветвляющиеся алгоритмы 33](#_Toc119622216)

[6.1. Простой условный оператор 33](#_Toc119622217)

[6.2. Множественное ветвление 34](#_Toc119622218)

[Контрольные вопросы 35](#_Toc119622219)

[Задания для самопроверки 35](#_Toc119622220)

[7. Циклические алгоритмы 35](#_Toc119622221)

[7.1. Циклы с параметром 35](#_Toc119622222)

[7.2. Циклы с условием 36](#_Toc119622223)

[Контрольные вопросы 37](#_Toc119622224)

[Задания для самопроверки 37](#_Toc119622225)

[8. ПОСЛЕДОВАТЕЛЬНОСТИ 37](#_Toc119622226)

[8.1. СТРОКИ 37](#_Toc119622227)

[8.2. СПИСКИ 39](#_Toc119622228)

[8.3. КОРТЕЖИ (tuple) 44](#_Toc119622229)

[Контрольные вопросы 45](#_Toc119622230)

[Задания для самопроверки 45](#_Toc119622231)

[9. Функция 45](#_Toc119622232)

[Контрольные вопросы 54](#_Toc119622233)

[Задание для самопроверки 54](#_Toc119622234)

[10. Файлы 54](#_Toc119622235)

[Контрольные вопросы 54](#_Toc119622236)

[Задания для самопроверки 54](#_Toc119622237)

[11. Модули и пакеты 54](#_Toc119622238)

[11.1. Основные понятия 54](#_Toc119622239)

[11.2. Модули и пакеты в Python 55](#_Toc119622240)

[11.3. Подключение и использование 56](#_Toc119622241)

[Контрольные вопросы 56](#_Toc119622242)

[Задание для самопроверки 57](#_Toc119622243)

[11. Графика 57](#_Toc119622244)

[Контрольные вопросы 58](#_Toc119622245)

[ЗАКЛЮЧЕНИЕ 58](#_Toc119622246)

[Библиографический список 60](#_Toc119622247)

ВВЕДЕНИЕ

Отличным решением с точки зрения преподавания основ программирования и алгоритмизации студентам может стать изучение языка Python, который относится к категории свободно распространяемого программного обеспечения.

Python - современный, активно развивающийся язык программирования, который привлекает разработчиков всего мира. Он используется во многих крупных компаниях, таких как Google, IBM, Microsoft, Red Hat, Yahoo! и др. Это бесплатная система с открытым кодом, поддерживающая динамическую типизацию (не надо заранее объявлять переменные), объектно-ориентированное программирование, интеграцию с другими языками, такими как С, C++, Java, обладает кроссплатформенностью (независимостью от той или иной операционной системы) и многими другими достоинствами.

Тем не менее, проведенный анализ литературы по языку программирования Python, имеющейся на рынке книгопечатной продукции, показал, что ее содержание практически не отвечает целям совместной работы студента и преподавателя в конкретном учебном заведении и на конкретном занятии. В основном книги по программированию на Python имеют характер справочных изданий либо вовлекают читателя в написание игровых моментов. Вероятно, подобное изложение представляет интерес для начинающего, но при этом рабочие программы дисциплин, связанных с обучением программированию, подразумевают совершенно иной алгоритм знакомства с языком.

Таким образом, главная цель настоящего пособия - обеспечить прочное и сознательное освоение основ алгоритмизации и программирования, формирование практических умений - профессиональных, учебных, интеллектуальных, необходимых студенту. Помимо приобретения чисто практических умений, ценных с точки зрения освоения компьютерной грамотности, пользователь получит наглядное представление о возможностях, предоставляемых компьютером человеку, выработают при решении поставленных задач такие профессионально значимые качества, как самостоятельность, ответственность, точность, творческая инициатива.

Учебное пособие построено так, что непосредственно к программированию пользователь приступает с самого начала - первая программа описывается уже во второй главе (в *первой главе* приводятся теоретические основы алгоритмизации и программирования).

В *первой главе* дано определение модуля программ.

Во *второй, третьей, четвертой и пятой главах* рассматриваются виды, способы описания алгоритма, основы языка и типы данных.

В *пятой главе* изложен процесс работы с оператором присваивания и простейший ввод, и вывод. В *шестой* главе рассмотрены алгоритмы ветвления.

Материал *седьмой главы* даст представление о работе с простым условным оператором и множественным ветвлением.

В *седьмой главе* объясняется работа с циклическими алгоритмами.

В *восьмой главе* речь пойдет о работе с последовательностями, а именно: строки, списки, кортежи.

*Девятая глава* посвящена созданию пользовательских функций, технике написания и импортирования собственных модулей.

*Десятая и одиннадцатая главы* содержат информацию о работе с файлами, умение работать с которыми - необходимая компетенция человека, постигающего азы программирования, а также объектно-ориентированному и событийно-ориентированному программированию. Читатель получит представление о создании таких объектов, как классы, научится писать методы, расширяющие функциональность классов, а также создавать новые классы на основе уже существующих.

Кроме того, знакомство с программированием виджетов позволит приобрести навыки создания Windows-приложений на языке Python.

При наборе на компьютере программ, приведенных в учебном пособии, следует учитывать, что в Python отступы (четыре пробела или нажатие клавиши Tab) являются частью синтаксиса программных конструкций.

Преподаватели могут оценить учебное пособие с точки зрения методики преподавания. Многие программы специально написаны таким образом, чтобы учащиеся имели возможность доработать или оптимизировать их код, что поможет педагогу создать творческую атмосферу на занятиях. Практически к каждому из разделов приведены примеры решения задач, упражнения, задачи для самостоятельного решения, контрольные вопросы, позволяющие оценить уровень подготовки студента.

Следует отметить, что учебное пособие не может претендовать на функциональную полноту или на полную оригинальность приведенных методов, алгоритмов и программ. Автор преследовал иную цель: привлечь внимание к такому активно развивающемуся языку программирования, как Python, дать почувствовать его преимущественные отличия от других языков. Python доступен для свободного скачивания по адресу http://www.python.org/.

1. Разработка структуры программы и модульное программирование

1.1. Цель модульного программирования

Приступая к разработке каждой программы ПС, следует иметь в виду, что она, как правило, является большой системой, поэтому должны **БЫТЬ** приняты меры для ее упрощения. Для этого такую программу разрабатывают по частям, которые называются программными модулями. А сам такой метод разработки программ называют ***модульным программированием***.

***Программный модуль*** − это любой фрагмент описания процесса, оформляемый как самостоятельный программный продукт, пригодный для использования в описаниях процесса.

2. Постановка и алгоритмизация задач

2.1. Понятие алгоритма

Для составления программы, предназначенной для решения на ЭВМ какой-либо задачи, требуется составление алгоритма ее решения.

**Алгоритм** − это точное предписание, которое определяет процесс, ведущий от исходных данных к требуемому конечному результату. Алгоритмами, например, являются правила сложения, умножения, решения алгебраических уравнений, умножения матриц и т.п.

Применительно к ЭВМ алгоритм определяет вычислительный процесс, начинающийся с обработки некоторой совокупности возможных исходных данных и направленный на получение определенных этими исходными данными результатов. Термин вычислительный процесс распространяется и на обработку других видов информации, например, символьной, графической или звуковой.

Если вычислительный процесс заканчивается получением результатов, то говорят, что соответствующий алгоритм применим к рассматриваемой совокупности исходных данных. В противном случае говорят, что алгоритм неприменим к совокупности исходных данных. Любой применимый алгоритм обладает следующими ***основными свойствами:***

* + результативностью;
  + определенностью;
  + массовостью;
  + дискретностью;
  + конечностью.

Результативность означает возможность получения результата после выполнения конечного количества операций.

Определенность состоит в совпадении получаемых результатов независимо от пользователя и применяемых технических средств.

Массовость заключается в возможности применения алгоритма к целому классу однотипных задач, различающихся конкретными значениями исходных данных.

Дискретность означает разбиение алгоритма на конечную последовательность действий или шагов при его выполнении.

Конечность означает то, что алгоритм должен выполняться за конечное время.

Для задания алгоритма необходимо описать следующие его элементы:

* + набор объектов, составляющих совокупность возможных исходных данных, промежуточных и конечных результатов;
  + правило начала;
  + правило непосредственной переработки информации (описание последовательности действий);
  + правило окончания;
  + правило извлечения результатов.

Алгоритм всегда рассчитан на конкретного исполнителя. В нашем случае таким исполнителем является ЭВМ. Для обеспечения возможности реализации на ЭВМ алгоритм должен быть описан на языке, понятном компьютеру, то есть на языке программирования.

Таким образом, можно дать следующее определение программы.

Программа для ЭВМ представляет собой описание алгоритма и данных на некотором языке программирования, предназначенное для последующего автоматического выполнения.

2.2. Способы описания алгоритмов

К основным способам описания алгоритмов можно отнести следующие:

* + словесно-формульный;
  + структурный или блок-схемный;
  + с помощью граф-схем;
  + с помощью сетей Петри.

Перед составлением программ чаще всего используются словесно-формульный и блок-схемный способы. Иногда перед составлением программ на низкоуровневых языках программирования типа языка Ассемблера алгоритм программы записывают, пользуясь конструкциями некоторого высокоуровнего языка программирования. Удобно использовать программное описание алгоритмов функционирования сложных программных систем. Так, для описания принципов функционирования ОС использовался Алголоподобный высокоуровневый язык программирования.

При ***словесно***-***формульном*** способе алгоритм записывается в виде текста с формулами по пунктам, определяющим последовательность действий.

Пусть, например, необходимо найти значение следующего выражения:

*у* = 2*а* – (*х* + 6).

Словесно-формульным способом алгоритм решения этой задачи может быть записан в следующем виде:

1. Ввести значения, *а* и *х*.

2. Сложить *х* и 6.

3. Умножить *a* на 2.

4. Вычесть из 2*а* сумму (*х* + 6).

5. Вывести *у*, как результат вычисления выражения.

При ***блок***-***схемном*** описании алгоритм изображается геометрическими фигурами (блоками), связанными по управлению линиями (направлениями потока) со стрелками. В блоках записывается последовательность действий.

Данный способ по сравнению с другими способами записи алгоритма имеет ряд преимуществ. Он наиболее нагляден: каждая операция вычислительного процесса изображается отдельной геометрической фигурой. Кроме того, графическое изображение алгоритма наглядно показывает разветвления путей решения задачи в зависимости от различных условий, повторение отдельных этапов вычислительного процесса и Другие детали.

Оформление программ должно соответствовать определенным требованиям. В настоящее время действует единая система программной документации (ЕСПД), которая устанавливает правила разработки, оформления программ и программной документации. В ЕСПД определены и правила оформления блок−схем алгоритмов (ГОСТ 19.701-90 (ИСО 5807-85) Единая система программной документации (ЕСПД). Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения).

Операции обработки данных и носители информации изображаются на схеме соответствующими блоками. Большая часть блоков по построению условно вписана в прямоугольник со сторонами, *а* и *b*. Минимальное значение, *а* = 10 мм, увеличение, *а* производится на число, кратное 5 мм. Размер *b* = 1,5*a*. Для отдельных блоков допускается соотношение между, *а* и *b*, равное 1:2. В пределах одной схемы рекомендуется изображать блоки одинаковых размеров. Все блоки нумеруются. Виды и назначение основных блоков приведены в табл. 2.1.

Линии, соединяющие блоки и указывающие последовательность связей между ними, должны проводиться параллельно линиям рамки. Стрелка в конце линии может не ставиться, если линия направлена слева направо или сверху вниз. В блок может входить несколько линий, то есть блок может являться преемником любого числа блоков. Из большинства блоков может выходить только одна линия. Блок Решение может иметь в качестве продолжения один из двух блоков, и из него выходят две линии. Если на схеме имеет место слияние линий, то место пересечения выделяется точкой. В случае, когда одна линия подходит к другой и слияние их явно выражено, точку можно не ставить.

Схему ***алгоритма*** следует выполнять как единое целое, однако в случае необходимости допускается обрывать линии, соединяющие блоки.

Если при обрыве линии продолжение схемы находится на этом же листе, то на одном и другом конце линии изображается специальный символ ***соединитель*** − окружность диаметром 0,5*а*. Внутри парных окружностей указывается один и тот же идентификатор. В качестве идентификатора, как правило, используется порядковый номер блока, к которому направлена соединительная линия.

Если схема занимает более одного листа, то в случае разрыва линии вместо окружности используется межстраничный соединитель. Внутри каждого, соединителя указывается адрес − откуда и куда направлена соединительная линия. Адрес записывается в две строки: в первой указывается номер листа, во второй − порядковый номер блока.

Блок-схема должна содержать все разветвления, циклы и обращения к подпрограммам, содержащиеся в программе.

*Таблица 2.1*

**Условные обозначения блоков**

|  |  |  |
| --- | --- | --- |
| Наименование | Обозначение | Функции |
| Процесс |  | Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных |
| Ввод−вывод |  | Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод) |
| Решение |  | Выбор направления выполнения алгоритма в зависимости от некоторых переменных условий |
| Модификация |  | Организация циклических конструкций |
| Предопределенный процесс |  | Использование ранее созданных и отдельно написанных программ (подпрограмм) |
| Пуск−останов |  | Начало, конец, прерывание процесса обработки данных |
| Документ |  | Вывод данных на бумажный носитель |
| Магнитный диск |  | Ввод−вывод данных, носителем которых служит магнитный диск |
| Соединитель |  | Указание связи между прерванными линиями, соединяющими блоки |
| Межстраничный соединитель |  | Указание связи между прерванными линиями, соединяющими блоки, расположенные на разных листах |
| Комментарий |  | Связь между элементом схемы и пояснением |

2.3. Структурные схемы алгоритмов

Одним из свойств алгоритма является ***дискретность*** −возможность расчленения процесса вычислений, предписанных алгоритмом, на отдельные этапы, возможность выделения участков программы с определенной структурой. Можно выделить и наглядно представить графически три простейшие структуры:

* + последовательность двух или более операций;
  + выбор направления;
  + повторение.

Любой вычислительный процесс может быть представлен как комбинация этих элементарных алгоритмических структур. Соответственно, вычислительные процессы, выполняемые на ЭВМ по заданной программе, можно разделить на три основных вида:

* + линейные;
  + ветвящиеся;
  + циклические.

***Линейным*** принятоназывать вычислительный процесс, в котором операции выполняются последовательно, в порядке их записи. Каждая операция является самостоятельной, независимой от каких-либо условий. На схеме блоки, отображающие эти операции, располагаются в линейной последовательности.

Линейные вычислительные процессы имеют место, например, при вычислении арифметических выражений, когда имеются конкретные числовые данные и над ними выполняются соответствующие условию задачи действия. На рис. 2.1 показан пример линейного алгоритма, определяющего процесс вычисления арифметического выражения

*y =* (*b2* − *ас*):(*а + с*)*.*

Вычислительный процесс называется ***ветвящимся,*** если для его реализации предусмотрено несколько направлений (ветвей). Каждое отдельное направление процесса обработки данных является отдельной ветвью вычислений. Ветвление в программе − это выбор одной из нескольких последовательностей команд при выполнении программы. Выбор направления зависит от заранее определенного признака, который может относиться к исходным данным, к промежуточным или конечным результатам. Признак характеризует свойство данных и имеет два или более значений.

Ветвящийся процесс, включающий в себя две ветви, называется простым, более двух ветвей − сложным. Сложный ветвящийся процесс можно представить с помощью простых ветвящихся процессов.

Направление ветвления выбирается логической проверкой, в результате которой возможны два ответа: ‘да’ − условие выполнено и ‘нет’ − условие не выполнено.

Следует иметь в виду, что, хотя на схеме алгоритма должны быть показаны все возможные направления вычислений в зависимости от выполнения определенного условия (или условий), при однократном прохождении программы процесс реализуется только по одной ветви, а остальные исключаются. Любая ветвь, по которой осуществляются вычисления, должна приводить к завершению вычислительного процесса.

На рис. 2.1 показан пример линейного алгоритма, в котором происходит обмен значений переменных а и b:

I. Постановка задачи

Даны переменные a и b. Выполнить обмен значений, используя вспомогательную переменную.

II. a и b – исходные (ввод)

t – вспомогательная

a и b – результирующие

III. Описание алгоритма в виде блок-схемы



Рис 2.1 Описание алгоритма в виде блок-схемы

IV. Код программы

a = input()

b = input()

t = a

a = b

b = t

print(a, b)

На рис. 2.2 показан пример алгоритма разветвления, в котором происходит сравнение значений переменных а и b, для получения наибольшего значения:

I. Постановка задачи

Даны переменные a и b. Выполнить проверку на максимальное число.

II. a и b – исходные (ввод)

a и b – результирующие

III. Описание алгоритма в виде блок-схемы



Рис 2.2 Пример описания алгоритма разветвления

IV. Код программы

a = input()

b = input()

if a > b:

print(a)

else:

print(b)

***Циклическими*** называются программы, содержащие циклы. Цикл− это многократно повторяемый участок программы.

В зависимости от расположения проверки условия окончания цикла различают циклы с нижним и верхним окончаниями (рис.3.3). Для цикла с нижним окончанием (рис.3.3, *б*) тело цикла выполняется как минимум один раз, так как сначала производятся вычисления, а затем проверяется условие выхода из цикла. В случае цикла с верхним окончанием (рис.3.3, *а*) тело цикла может не выполниться ни разу в случае, если сразу соблюдается условие выхода.

б

Рис. 2.3. Примеры циклических алгоритмов

Цикл называется ***детерминированным,*** если число повторений тела цикла заранее известно или определено. Цикл называется ***итерационным,*** если число повторений тела цикла заранее неизвестно, а зависит от значений параметров (некоторых переменных), участвующих в вычислениях.

На рис.2.4 показан пример циклического алгоритма вычисления суммы десяти цифр из числа n.

4. I. Дано натуральное число n, найти сумму цифр, входящих в n.

II. n – исходные

s – результирующие

III. Создание блок-схемы



Рис 2.4 Пример циклического алгоритма вычисления суммы десяти цифр из числа n.

IV. Код программы

n = int(input())

s = 0

while n > 0:

s += n % 10

n = n // 10

print(s)

Для успешного использования ЭВМ в своей профессиональной деятельности пользователь должен уметь формулировать задачи, разрабатывать алгоритмы их решения, записывать алгоритмы на языке, понятном ЭВМ.

Этапы подготовки и решения реальных задач приведены на рис.3.5.

В курсе информатики детально рассматриваются 4, 5 и 6 этапы решения задач, так как они непосредственно связаны с использованием ЭВМ.



Рис.2.5. Этапы решения задач на ЭВМ

Контрольные вопросы

1. Сформулируйте определение алгоритма.
2. Перечислите основные свойства алгоритма.
3. Укажите способы описания алгоритма.
4. Перечислите основные графические блоки.
5. Перечислите виды алгоритмов.
6. Охарактеризуйте линейный алгоритм, алгоритм ветвления и циклический.

Задания для самопроверки

1. Как будет выглядеть блок-схема if-elif-else

2. Как произвести swap 3 элементов

3. В чем преимущества и недостатки python перед с, с++

3. Основы языка

3.1. Алфавит языка

Под ***алфавитом языка*** понимают совокупность допустимых символов. В языке Python используются символы ASCII (американский стандартный код обмена информацией).

Python включает следующий набор основных символов:

1. 26 латинских строчных и 26 латинских прописных букв:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

2. Подчеркивание: \_

3. 10 цифр: 0 1 2 3 4 5 6 7 8 9

4. Знаки операций: + − \* / = <> < > <= >= :=

5. Ограничители: . , ‘ () [ ] { }

6. Спецификаторы: ^ #

7**.** Служебные (**ключевые**) слова: список этих слов можно получить в интерактивном режиме с помощью импорта модуля

>>> import keyword

>>> keyword.kwlist

с вызовом контейнера ключевых слов keyword.kwlist [‘False’, ‘None’, ‘True’, ‘and’, ‘as’, ‘assert’, ‘break’, ‘class’, ‘continue’, ‘def’, ‘del’, ‘elif’, ‘else’, ‘except’, ‘finally’, ‘for’, ‘from’, ‘global’, ‘if’, ‘import’, ‘in’, ‘is’, ‘lambda’, ‘nonlocal’, ‘not’, ‘or’, ‘pass’, ‘raise’, ‘return’, ‘try’, ‘while’, ‘with’, ‘yield’]

8. Escape - последовательности (управляющие последовательности) используются для описания определённых специальных символов внутри строковых литералов, то есть внутри ограничителей ‘‘. Вот некоторые из них:

\’ – одинарная кавычка

\’ – двойная кавычка

\? – вопросительный знак

\\ – обратный слеш

\n – новая строка

\t – горизонтальная табуляция

\v – вертикальная табуляция

\0 – нулевой символ.

Кроме перечисленных, в набор основных символов входит *пробел*. Пробелы нельзя использовать внутри сдвоенных символов и зарезервированных слов.

Следующая табл. 3.1 демонстрирует приоритетность операторов от высокой до самой низкой.

*Таблица 3.1*

|  |  |
| --- | --- |
| **Операторы** | **Применение** |
| { } | Скобки (объединение) |
| f(args…) | Вызов функции |
| x[index:index] | Срез |
| x[index] | Получение по индексу |
| x.attribute | Ссылка на атрибут |
| \*\* | Возведение в степень |
| ~x | Побитовое нет |
| +x, -x | Положительное, отрицательное число |
| \*, /, % | Умножение, деление, остаток |
| +, — | Сложение, вычитание |
| <<, >> | Сдвиг влево/вправо |
| & | Побитовое И |
| ^ | Побитовое ИЛИ НЕ |
| | | Побитовое ИЛИ |
| in, not in, is, is not, <, <=, >, >=, <>, !=, == | Сравнение, принадлежность, тождественность |
| not x | Булево НЕ |
| and | Булево И |
| or | Булево ИЛИ |
| lambda | Лямбда-выражение |

***Идентификатор*** − это имя любого объекта языка. Он может состоять из латинских букв (*a*…*z*), цифр (0...9) и знака подчеркивания и не должен начинаться с цифры. Прописные и строчные буквы в идентификаторах и зарезервированных словах считаются различными.

Разделители используются для отделения друг от друга идентификаторов, чисел и зарезервированных слов. К разделителям относятся, например, пробел и комментарий. В любом месте программы, где разрешается один пробел, их можно вставить любое количество.

В Python все комментарии начинаются с символа **#**и все они являются однострочными. В качестве многострочного комментария будем пользоваться строками документирования т.е. разместим код в утроенных кавычках.

*Пример 3.1:* Виды комментариев

#Обычный текст комментария

my\_var = ‘Hi!’ #Описываем переменную

#your\_var = ‘read’ - в данном случае код не выполняется т.к. символ #в начале строки

```

комментарии

расположенные на

несколько строк

```

”””

Другой способ

комментария состоящего

из нескольких строк

”””

3.2. Структура программы

Любая Python-программа состоит из последовательности лексем (допустимых символов), записанных в определенном порядке и по определенным правилам.

Лексемы включают в себя:

комментарии;

литералы;

знаки пунктуации;

идентификаторы;

ключевые слова.

Для составления программы необходимо объединить лексемы в синтаксические конструкции. Примеры конструкций приведены в примере 3.2.

*Пример 3.2:* Конструкции Python-программы

#1. Простейшие синтаксические конструкции - литералы

#простые

5 #Литерал - целое число

‘это строка’ #Литерал - строка

#составные

[1, 2, 3] #Литерал - список чисел

#с использованием знаков операций

x = 1 #Присваивание

y += 1 #Увеличение на 1

#2. Операторы

if x > 10: #Если выражение истинно

x += 1 #Увеличить x на 1

Контрольные вопросы

1. Сформулируйте определение алфавита языка программирования.
2. Перечислите основные символы языка программирования Python.
3. Сформулируйте определение идентификатора.
4. Дайте понятие разделителей языка программирования Python.
5. Определите структуру программы на языке программирования Python.

Задания для самопроверки

1. Какие бывают булевые переменные?

2. Может ли быть у условия if несколько аргументов, если да то, как их можно записать?

3. Можно ли к условию else написать аргумент, если да, то как?

4. Есть ли у переменных int ограничени?

4. Типы данных

Тип данных (англ. Data type) - характеристика, определяющая множество значений и набор операций на этих значениях:

* множество допустимых значений, которые могут принимать данные, принадлежащие к этому типу (например, объект типа. Целое число может принимать только целочисленные значение в определенном диапазоне);
* набор операций, которые можно осуществлять над данными, принадлежащими к этому типу (например, объекты типа Целое число умеют складываться, умножаться и т.д.).

Все типы в Python являются объектами (в отличие, например, от C++). При создании объекта вызывается специальная функция - конструктор. Конструктор - специальный блок инструкций, вызываемый при создании объекта.

Тип данных характеризует внутреннее представление, множество допустимых значений для этих данных, а также совокупность операций над ними. Среди типов данных различают стандартные (предопределенные разработчиками языка) и пользовательские (определяемые программистом в своей программе).

В Python встроенные типы данных подразделяются на 2 группы:

I.Скалярные (неделимые).

1.Числа (целое, вещественное).

2.Логический тип.

3.NoneType.

II.Структурированные (составные) / коллекции.

1.Последовательности: строка, список, кортеж, числовой диапазон.

2.Множества.

3.Отображения: словарь.

Рассмотрим сначала простые типы данных, каждый из которых определяет упорядоченное множество значений: целые типы, вещественные типы. Все эти типы, кроме вещественных, являются порядковыми.

4.1. Целые типы

Целые числа в Python представлены типом int. Числа в Python 3 ничем не отличаются от обычных чисел. Они поддерживают набор самых обычных арифметических операций табл. 4.1:

*Таблица 4.1*

**Арифметические операции**

|  |  |  |
| --- | --- | --- |
| № п.п. | Запись операции | Функциональность операций |
| 1 | x + y | Сложение |
| 2 | x – y | Вычитание |
| 3 | x \* y | Умножение |
| 4 | x / y | Деление |
| 5 | x // y | Получение целочисленного остатка от деления |
| 6 | x % y | от деления |
| 7 | –x | Унарный минус |
| 8 | x \*\* y | Возведение в степень |

*Пример 4.1:* Арифметические операции

>>> 255 + 34

289

>>> 5 \* 2

10

>>> 20 / 3

6.666666666666667

>>> 20 // 3

6

>>> 20 % 3

2

>>> 3 \*\* 4

81

>>> 3 \*\* 100

515377520732011331036461129765621272702107522001

В Python нет максимального целого числа. Максимум определяется объемом памяти.

4.2. Вещественные типы

Вещественные числа в Python имеют три типа значений:

* float (двойная точность)
* complex (комплексные числа вида3.5+5j);
* decimal.Decimal (большая точность, по умолчанию 28 знаков после запятой).

Наиболее часто используемый тип float представляет числа с плавающей точкой двойной точности, диапазон значений которых зависит от компилятора, применявшегося для компиляции интерпретатора Python. Числа типа float записываются с десятичной точкой или в экспоненциальной форме записи:

>>> 5.7 #Точка -разделитель целой и дробной части 5.7

>>> 4. #Если дробной части нет, ее можно опустить4.0

>>> -2.5 #Отрицательное вещественное число -2.5

>>> 8e-4 #Экспоненциальная форма записи 0.0008

>>> 0.1 + 0.2 #Проблема потери точности актуальна для вещественных чисел 0.30000000000000004

Для чисел с плавающей точкой существует ряд нюансов:

* в машинном представлении такие числа хранятся как двоичные числа. Это означает, что одни дробные значения могут быть представлены точно (такие как 0.5), а другие - только приблизительно (такие как 0.1и 0.2, например, их сумма будет равна не 0.3, а 0.30000000000000004);
* для представления используется фиксированное число битов, поэтому существует ограничение на количество цифр в представлении таких чисел.

В связи с этим числа типа float не могут надежно сравниваться на равенство значений, т.к. имеют ограниченную точность. Проблема потери точности — это не проблема, свойственная только языку Python, а особенность компьютерного представления чисел. Вещественные числа поддерживают те же операции, что и целые. Однако (из-за представления чисел в компьютере) вещественные числа неточны, и это может привести к ошибкам:

*Пример 4.2*: Ошибки при работе с вещественными числами

>>> a = 3 \*\* 1000

>>> a + 0.1

Traceback (most recent call last):

File ‘‘, line 1, in

OverflowError: int too large to convert to float

>>> c = 150

>>> d = 12.9

>>> c + d

162.9

>>> p = abs(d -c) #Модуль числа

>>> print(p)

137.1

>>> round(p) #Округление

137

Для арифметических операций тип результата операции определяется типом аргументов. Если тип результата явно не предусмотрен при вычислении (например, округление до целых подразумевает получение результата типа int), действуют следующие правила:

* float: если хотя бы один аргумент имеет тип float;
* int: если все аргументы имеют тип int.

Вещественные числа хранятся неточно. Каждый из имеющихся вещественных типов гарантирует правильное хранение только определенного количества значащих цифр, их называют верными цифрами. С математической точки зрения, из-за особенностей внутреннего представления речь идет об относительной погрешности.

Контрольные вопросы

1. Сформулируйте понятие типа данных.
2. Перечислите скалярные числовые типы.
3. Укажите особенности хранения вещественных чисел.
4. Приведите перечень арифметических операций.
5. Сформулируйте понятие приоритета операций

Задания для самопроверки

1. Сколько знаков у переменной типа float после точки?

2. Как можно сократить количество знаков после точки?

3. Что будет если превратить int в float или наоборот?

4. Можно ли превратить str в int, если да то как?

5. Как можно избавится от отрицательного числа без abs и -argument?

5. Линейные алгоритмы

Алгоритмические действия над исходными данными и рабочими объектами языка, необходимые для решения поставленной задачи описываются при помощи операторов. Наиболее простой случай представляют собой линейные алгоритмы. При выполнении линейных участков алгоритма операторы выполняются последовательно друг за другом в том порядке, в котором они перечислены в программе. При этом могут использоваться операторы присваивания, операции ввода и вывода.

5.1. Оператор присваивания

Для связывания (и при необходимости предварительного создания) объекта и переменной используется оператор присваивания (рис. 5.1.) и имеет следующий синтаксис:

переменная

**=**

выражение

Рис. 5.1. Синтаксическая диаграмма оператора присваивания

Присваивание выполняется ‘справа налево’ и подразумевает шаги:

* если справа от оператора находится литерал (например, строка или число) в операнд слева записывается ссылка, которая указывает на объект в памяти, хранящий значение литерала:

a =100 #Создание объекта 100 и запись ссылки на него в переменную ‘a’

* если справа находится ссылка на объект, в левый операнд записывается ссылка, указывающая на тот же самый объект, на который ссылается правый операнд;

a =100

b =a #В переменную ‘b’ копируется ссылка из ‘a’ –

#они будут указывать на один и тот же объект.

Переменная лишь указывает на данные - хранит ссылку, а не сами данные. В виду того, что копирования данных при этом не происходит, операция присваивания выполняется с высокой скоростью. В связи с этим целесообразнее в Python в качестве метафоры рассматривать переменные как стикеры (этикетки), цепляемые к данным, и следует говорить, что ‘переменная присвоена объекту’, а не привычное ‘переменной присвоен объект’.

1. Простое присваивание использует одно l-значение и одно r-значение.

#

#Объект 5 целого типа связывается с переменной ‘a’

>>> a =5

>>> a

5

1. Сокращенная запись присваивания часто применяется, когда нужно обновить значение переменной по отношению к текущему значению.

#Сокращенная запись образуется для всех операторов (+, -, \*, \*\*, / и т.п.) одинаково, например

#’\*=‘ для умножения и т.д.

#

#Увеличиваем значение связанного целого объекта ‘a’ на 1, эквивалентно a = a + 1

>>> a +=1

>>> a

6

1. Параллельное присваивание, где выражение присваивания содержит больше одного l-и r-значений.

#

#Связывание значений 1, ‘aaa’, 3 с переменными ‘x’, ‘y’, ‘z’ соответственно

>>> x, y, z =1, ‘aaa’, 3

>>> x, y, z

(1, ‘aaa’, 3)

1. Оператор присваивание выполняется справа налево, поэтому также можно образовывать цепочки присваивания.

#

#0 связывается с переменной ‘y’, а затем и с переменной ‘x’

>>> x = y = 0

>>> x, y

(0, 0)

Переменная должна быть проинициализирована перед использованием в выражении. Например, код a=b+2 или b+=1, вызовет ошибку, если идентификатор b не был предварительно определен.

Создание объекта любого типа подразумевает выделение памяти для размещения данных об этом объекте. Когда объект больше не нужен - его необходимо удалить, очистив занимаемую память.

5.3. Простейший ввод и вывод

Для вывода значения переменной на экран используют оператор print(). Это встроенная функция. Команда вывода сначала автоматически преобразует числовое значение в цепочку символов (цифр), а затем выводит эти символы так же, как обычный текст. Текст заключается в кавычки.

Например:

c= 5

print(c) #будет выведено число 5

print(‘c’) #будет выведен символ ‘с’

Вывод целых чисел

Функция print() вставляет по одному пробелу между элементами списка вывода:

a = 12

b = 5

c = 155

print(a, b, c) #12 5 155

Иногда требуется выводить данные в виде таблицы, выравнивая значения в каждом столбце по правой границе:

12 5 155

211 315 8

Предположим, есть необходимость работы с натуральными числами, которые меньше 1000. Тогда на каждое число можно выделить 4 позиции на экране: три на запись числа и ещё один пробел слева, разделяющий числа. Запись в следующем формате:

print( ‘{:4}{:4}{:4}’.format(a, b, c) )

>>> 5 5 5

Это форматный вывод: строка для вывода строится с помощью встроенной функции format().Аргументы этой функции – a, b и с в скобках – это те данные, которые выводятся. Символьная строка слева от точки – это форматная строка, которая определяет, как именно данные будут представлены на экране. Фигурные скобки обозначают место для вывода очередного элемента: на первом месте выводится значение a, на втором – значение b и на третьем – с, в порядке их перечисления в списке аргументов функции format().Число после двоеточия – это количество позиций, которые отводятся на запись числа. В пределах этого поля число прижимается к правой границе.

Например, числа 12, 5 и 155 будут выведены так: ◦◦12◦◦◦5◦155

Количество позиций можно не указывать: print( ‘{}{}{}’.format(a, b, c)) тогда данные выводятся вплотную друг к другу: 125155

Между данными из списка можно выводить и другие символы. Например, программа:

num1 = 12; num2 = 13

print( ‘{}+{}={}’.format(num1, num2, num1+num2) )

выведет:12+13=25

Как видно из первой строчки программы, в одной строке можно записывать несколько операторов, разделяя их точками с запятой.

print

**(**

переменная

**)**

**,**

Рис. 5.2. Синтаксическая диаграмма оператора вывода

**Вывод вещественных чисел**. При выводе вещественных значений по умолчанию (то есть, если не сказано делать иначе) выводится 16 значащих цифр.

Например, команда: print(16/7) выводит:2.2857142857142856

Если такой вариант не устраивает, применяют форматный вывод. Например:

x = 16/7

print(‘x={:f}’.format(x))

>>>x=2.285714.

Здесь после двоеточия указан формат f, по умолчанию он оставляет 6 цифр в дробной части числа.

Перед буквой f можно записать два числа через точку. Примеры: Результат вывода print(‘x={:10.3f}’.format(x))

x=◦◦◦◦12.346

print(‘x={:8.2f}’.format(x))

x=◦◦12.35

print(‘x={:2.2f}’.format(x))

x=12.35

print(‘x={:0.2f}’.format(x))

x=12.35 print(‘x={:0.1f}’.format(x))

x=12.3

Первое число задаёт общее количество позиций, отведённое на вывод значения, а второе – количество цифр в дробной части.

Если пробелы слева от числа не нужны, а требуется только ограничить количество знаков в дробной части, вместо первого числа пишут 0 или вообще ничего не пишут: print(‘x={:.2f}’.format(x)) #x=12.35.

Для очень больших или очень маленьких чисел используют научный формат (стандартный вид числа). Он обозначается буквой e внутри фигурных скобок:

x = 1e10/7

print(‘x={:12.4e}’.format(x)) #x=1.4286e+09 Синтаксическая диаграмма вызова стандартной функции print (рис. 6.3.):

**(**

выражение

**)**

**,**

формат

print

Рис. 5.3. Синтаксическая диаграмма функции вывода

Число слева от точки в строке формата – это общее количество позиций для вывода числа, а второе число – количество знаков в дробной части мантиссы (для всех чисел, кроме числа 0, она больше или равна 1 и меньше 10). Если первое число не указывать, будет использовано наименьшее возможное место.

Ввод чисел. Для ввода данных с клавиатуры используется встроенная функция input(). Введём два числа и запишем их в переменные num1 и num2:

num1 = input()

num2 = input()

Заметим, что числа нужно вводить по одному в строчке, нажимая клавишу Enter после каждого введённого значения. Можно также вводить данные из файла или принимать через компьютерную сеть, но пока мы не будем обсуждать эти довольно сложные способы. Теперь вычислим сумму и запишем её в переменную summa: summa = num1 + num2.

Выведем результат на экран:

print( summa ).

Вот полная программа сложения двух чисел:

num1 = input()

num2 = input()

summa = num1 + num2

print( summa )

Запустив эту программу, увидим неожиданный результат: если ввести, например, числа 12 и 13, то получим не 25, а 1213.

Дело в том, что функция input не знает заранее, значение какого типа нужно ввести. Поэтому она считает всё, что введено, символьной строкой, то есть цепочкой символов. Операция сложения для символьных строк существует, но работает иначе, чем для чисел: вторая строка дописывается в конец первой. Таким образом, проблема в том, что программа воспринимает введённые нами данные не как числа, а как цепочки символов. Чтобы работать именно с числами, необходимо явно сказать, что введенные строки нужно преобразовать в числа.

Для ввода целых чисел это делает встроенная функция int(), а для вещественных – встроенная функция float():

x = int(input())

y = float(input())

Тогда программа примет вид:

num1 = int(input())

num2 = int(input())

summa = num1 + num2

print(summa)

Обратите внимание, что в начале каждой строки программы не должно быть пробелов. Недостаток этой программы – плохой диалог с пользователем:

* при вводе данных программа просто ждёт ввода, но что именно нужно вводить – не ясно;
* в конце работы программа выводит какое-то число, что оно означает – не ясно.

Программу можно легко доработать. Добавим в самом начале приглашение к вводу print( ‘Введите два целых числа:’ ) и оформим вывод, заменив последнюю строчку:

print( num1, ‘+’, num2, ‘=‘, summa )

Здесь все выводимые данные объединены в один список вывода, элементы в котором разделены запятыми. Обратите внимание, что имена переменных записаны без кавычек, а все символы, которые нужно вывести, – в кавычках (или в апострофах). Если в списке вывода указано имя переменной num1 без кавычек, программа выведет не символы ‘num1’, а значение, которое хранится в переменной num1. Теперь при вводе чисел 12 и 13 программа выведет 12+13=25.Вот текст измененной программы:

print( ‘Введите два целых числа:’ )

num1 = int( input() )

num2 = int( input() )

summa = num1 + num2

print( num1, ‘+’, num2, ‘=‘, summa )

Вывод этой программы:

Введите два целых числа:

12

14

12+14=26

В программе, которая показана выше, мы вводили числа по одному: сначала значение переменной num1, затем, после нажатия на клавишу Enter, значение переменной num2. Иногда нужно вводить несколько значений в одной строке, разделенных пробелом. Рассмотрим случай, когда нужно ввести два целых числа в одной строке и записать их в переменные num1 и num2. В этом случае программа должна:

* ввести символьную строку, содержащую запись двух чисел;
* выделить части, разделённые пробелами;
* каждую часть преобразовать в целое число. Мы уже знаем, как решить первую задачу: s = input(). Введённая строка записывается в переменную s. Применять к ней сразу функцию int( )нельзя, потому что она содержит не одно, а два числа. Чтобы выделить две части, применяется функция split() (от англ. split –расщепить), которая запишет значения в переменные num1 и num2: num1, num2 = s.split(. )Здесь используется так называемое множественное присваивание – в одном операторе присваивания задаются значения двух переменных. Можно обойтись и без переменной s: num1, num2 = input().split(). Теперь нужно применить функцию int к переменным num1 и num2 – преобразовать строки в целые числа: num1 = int(num1) num2 = int(num2). Все эти операции можно заменить одной строчкой: num1, num2 = map( int, input().split() ) Здесь вызывается функция map(), которая применяет другую функцию (в нашем случае для целых чисел–int) к каждой части, полученной после разбиения введённой строки на части.

При вводе нескольких вещественных чисел оператор будет выглядеть так:num1, num2 = map( float, input().split() ). Обратите внимание, что количество имён переменных слева от оператора присваивания должно точно соответствовать количеству введённых чисел: если их будет больше или меньше, программа завершится с ошибкой.

**Модуль math.** При работе с числами можно пользоваться стандартными функциями, которые объединены в модуль math. Для того, чтобы вызывать эти функции из своей программы надо подключить (импортировать) модуль с помощью команды import:

import math

После этого можно применять функции из этого модуля, например:

x= math.sqrt(5)

Здесь из модуля math вызывается функция sqrt, которая вычисляет квадратный корень из числа 5, то есть находит число, квадрат которого равен 5.

Для обращения к функциям модуля используется точечная запись: сначала записывают имя модуля, а затем через точку –название функции. Возможен и другой вариант, когда подключается не весь модуль, а только некоторые функции из него: from math import sqrt, piЭтой строчкой к программе подключены из модуля mathуже знакомая вам функция sqrtи константа (постоянная) pi, равная иррациональному числу (3,1415626...). В этом случае для обращения к ним уже не нужно будет указывать имя модуля:

x = sqrt(5)

R = 12

circleLen = 2\*pi\*R

Можно подключить сразу все функции из модуля, если написать знак \*вместо списка функций: from math import \* Чтобы вывести весь список функций, которые входят в модуль math, надо выполнить следующие действия:

>>> import math

>>> math

<module ‘math’ (built-in)>

>>> dir(math)

[‘\_\_doc\_\_’, ‘\_\_loader\_\_’, ‘\_\_name\_\_’, ‘\_\_package\_\_’, ‘\_\_spec\_\_’, ‘acos’, ‘acosh’, ‘asin’, ‘asinh’, ‘atan’, ‘atan2’, ‘atanh’, ‘ceil’, ‘copysign’, ‘cos’, ‘cosh’, ‘degrees’, ‘e’, ‘erf’, ‘erfc’, ‘exp’, ‘expm1’, ‘fabs’, ‘factorial’, ‘floor’, ‘fmod’, ‘frexp’, ‘fsum’, ‘gamma’, ‘gcd’, ‘hypot’, ‘inf’, ‘isclose’, ‘isfinite’, ‘isinf’, ‘isnan’, ‘ldexp’, ‘lgamma’, ‘log’, ‘log10’, ‘log1p’, ‘log2’, ‘modf’, ‘nan’, ‘pi’, ‘pow’, ‘radians’, ‘remainder’, ‘sin’, ‘sinh’, ‘sqrt’, ‘tan’, ‘tanh’, ‘tau’, ‘trunc’]

Список математических функций, не требующих подключения модуль math, приведен в табл. 5.2.

*Таблица 5.2*

**Список математических функций**

|  |  |  |
| --- | --- | --- |
| **№ п.п.** | **Запись функции** | **Назначение функции** |
| 1 | 2 | 3 |
| 1 | int(x) | Округляет число |
| 2 | round(x) | Округляет число до ближайшего целого. Если дробная часть числа равна 0.5, то число округляется до ближайшего четного числа. |
| 3 | floor(x) | Округляет число вниз (“пол’), при этом floor(1.5)==1, floor(-1.5)==-2 |
| 4 | trunc(x) | Округляет число |
| 5 | abs(x) | Модуль (абсолютная величина) |
| 6 | fabs(x) | Модуль (абсолютная величина). Эта функция всегда возвращает значение типа float. |
| 7 | sqrt(x) | Квадратный корень. |
| 8 | pow(a, b) | Возведение в степень, возвращает ab. |
| 9 | exp(x) | Экспонента, возвращает ex. |
| 10 | log(x) | Натуральный логарифм. При вызове в виде log(x,b) возвращает логарифм по основаниюb. |
| 11 | log10(x) | Десятичный логарифм |
| 12 | sin(x) | Синус угла, задаваемого в радианах |
| 13 | cos(x) | Косинус угла, задаваемого в радианах |
| 14 | tan(x) | Тангенс угла, задаваемого в радианах |
| 15 | asin(x) | Арксинус, возвращает значение в радианах |
| 16 | acos(x) | Арккосинус, возвращает значение в радианах |
| 17 | atan(x) | Арктангенс, возвращает значение в радианах |
| 18 | atan2(y, x) | Полярный угол (в радианах) точки с координатами (x, y) |
| 19 | pi | Константа π |

**Арифметические выражения**. Арифметические выражения обычно записываются в одну строчку. Они могут содержать константы (постоянные значения), имена переменных, знаки арифметических операций, круглые скобки (для изменения порядка действий).

Например: a = (c + b -1)/2\*d

Контрольные вопросы

1. Сформулируйте определение линейного алгоритма.
2. Сформулируйте назначение оператора присваивания.
3. Перечислите операторы ввода и вывода.
4. Дайте определение форматированного вывода.

Задания для самопроверки

1. Как взять корень третей степени?

2. Что будет если сложить 0.1 + 0.2?

3. Как ввести несколько переменных на 1 строке?

4. Опешите все параметры функции среза.

6. Разветвляющиеся алгоритмы

В этом разделе рассмотрим условную инструкцию if, которая является основной инструкцией, используемой для выбора среди альтернативных операций на основе результатов проверки.

6.1. Простой условный оператор

Условная инструкция if в языке Python – это типичная условная инструкция, которая присутствует в большинстве процедурных языков программирования. Синтаксически сначала записывается часть if с условным выражением, далее могут следовать одна или более необязательных частей elif (‘else if’) с условными выражениями и, наконец, необязательная часть else. Условные выражения и часть else имеют ассоциированные с ними блоки вложенных инструкций, с отступом относительно основной инструкции. Во время выполнения условной инструкции if интерпретатор выполняет блок инструкций, ассоциированный с первым условным выражением, только если оно возвращает истину, в противном случае выполняется блок инструкций else.

оператор1

:

if

Лог.выражение

оператор2

else

Рис. 6.1. Синтаксическая диаграмма условного оператора

Общая форма записи условной инструкции if выглядит следующим образом:

if <test1>: *#Инструкция if с условным выражением test1*

<statements1> *#Ассоциированный блок*

elif <test2>: *#Необязательные части elif*

<statements2>

else: *#Необязательный блок else*

<statements3>

6.2. Множественное ветвление

В языке Python *множественное ветвление* оформляется либо в виде последовательности проверок if/elif, как в предыдущем инструкции, либо индексированием словарей, либо поиском в списках. Поскольку словари и списки могут создаваться во время выполнения, они иногда способны обеспечить более высокую гибкость, чем жестко заданная логика инструкции if:

>>> choice = ‘ham’

>>> print {‘spam’: 1.25, *#Инструкция ‘switch’ на базе словаря*

... ‘ham’: 1.99, *#Используйте has\_key или get для*

... ‘eggs’: 0.99, *#значения по умолчанию*

... ‘bacon’: 1.10}[choice]

1.99

Эквивалентная, но менее компактная инструкция if в языке Python выглядит,

как показано ниже:

>>> if choice == ‘spam’:

... print 1.25

... elif choice == ‘ham’:

... print 1.99

... elif choice == ‘eggs’:

... print 0.99

... elif choice == ‘bacon’:

... print 1.10

... else:

... print ‘Bad choice’

**...**

Контрольные вопросы

1. Сформулируйте определение разветвляющегося алгоритма.
2. Перечислите виды алгоритмов ветвления.
3. Дайте определение условного оператора.
4. Охарактеризуйте работу оператора выбора.
5. Приведите пример использования операторов ветвления.

Задания для самопроверки

1. Для чего нужен is и is not?

2. что значат операции +=, -=, \*=, /=, //=, %=

3. Для чего нужны break, continue?

4. лучше записывать выражение в одну строку или в несколько? Свой ответ поясните.

7. Циклические алгоритмы

В этом разделе представлены две основными конструкциями организации *циклов* в языке Python – инструкциями, которые выполняют одну и ту же последовательность действий снова и снова. Первая из них, инструкция while, обеспечивает способ организации универсальных циклов; вторая, инструкция for, предназначена для обхода элементов в последовательностях и выполнения.

7.1. Циклы с параметром

Циклы for в языке Python начинаются со строки заголовка, где указывается переменная для присваивания (или – цель), а также объект, обход которого будет выполнен. Вслед за заголовком следует блок (обычно с отступами) инструкций, которые требуется выполнить:

for <target> in <object>: *#Связывает элементы объекта с переменной цикла*

<statements> *#Повторяющееся тело цикла: использует переменную цикла*

Когда интерпретатор выполняет цикл for, он поочередно, один за другим, присваивает элементы объекта последовательности переменной цикла и выполняет тело цикла для каждого из них. Для обращения к текущему элементу последовательности в теле цикла обычно используется переменная цикла, как если бы это был курсор, шагающий от элемента к элементу.

Имя, используемое в качестве переменной цикла (возможно, новой), которое указывается в заголовке цикла for, обычно находится в области видимости, где располагается сама инструкция for. О ней почти нечего сказать; хотя она может быть изменена в теле цикла, тем не менее ей автоматически будет присвоен следующий элемент последовательности, когда управление вернется в начало цикла. После выхода из цикла эта переменная обычно все еще ссылается на последний элемент последовательности, если цикл не был завершен инструкцией break (рис. 8.1).

for

оператор

оператор

else

:

in

объект

идент.

Рис. 7.1. Синтаксическая диаграмма цикла с параметром

7.2. Циклы с условием

В своей наиболее сложной форме инструкция while состоит из строки заголовка с условным выражением, тела цикла, содержащего одну или более инструкций с отступами, и необязательной части else, которая выполняется, когда управление передается за пределы цикла без использования инструкции break.

Интерпретатор продолжает вычислять условное выражение в строке заголовка и выполнять вложенные инструкции в теле цикла, пока условное выражение не вернет ложное значение:

while <test>: *#Условное выражение test*

<statements1> *#Тело цикла*

else: *#Необязательная часть else*

<statements2> *#Выполняется, если выход из цикла производится не*

*#инструкцией break*

Синтаксическая диаграмма имеет вид (рис. 8.2):

оператор

else

:

логическое

выражение

оператор

while

:

Рис. 7.2. Синтаксическая диаграмма цикла с условием

Цикл **while** организует выполнение одного (возможно составного) оператора пока истинно логическое выражение, стоящее в заголовке цикла. Поскольку значение логического выражения проверяется в начале каждой итерации, то тело цикла может не выполниться ни разу. Таким образом, в этом цикле логическое выражение – это условие продолжения работы в цикле.

*Пример 7.1.*: Дано натуральное число. Найти сумму и произведение его цифр.

a = abs(int(input()))

suma = 0

mult = 1

while a > 0:

digit = a % 10

suma += digit

mult \*= digit

a = a // 10

print(‘Сумма:’, suma)

print(‘Произведение:’, mult)

Контрольные вопросы

1. Перечислите виды операторов цикла.
2. Охарактеризуйте оператор цикла for.
3. Укажите требования к управляющей переменной.
4. Охарактеризуйте оператор цикла while.
5. В чем заключаются основные различия между циклами while и for?
6. Когда выполняется блок else в циклах?

Задания для самопроверки

1. Как будет выглядеть цикл if в блок-схеме?

2. Как сделать обратный цикл for от 100 до 1?

3. Как выполняется цикл for-else?

8. ПОСЛЕДОВАТЕЛЬНОСТИ

8.1. СТРОКИ

**Строки.** Строка – это последовательность символов с произвольным доступом. Строки в языке Python невозможно изменить – в этом случае говорят, что это immutable тип. Попытка изменить символ в определенной позиции или подстроку вызовет ошибку:

>>> word = ‘strength’

>>> word[2] = ‘y’

TypeError: ‘str’ object does not support item assignment

Строку можно изменить, например:

>>> word = word[:3] + ‘!’ + word[4:]

‘str!ngth’

Или так:

>>> word = word.replace(‘!’,’e’)

‘strength’

Индексы могут иметь отрицательные значения для отсчета с конца – отсчет начинается с -1:

>>> word[-1]

h

Строки могут быть заключены как в одинарные, так и в двойные кавычки, причем кавычки одного типа могут быть произвольно вложены в кавычки другого типа:

>>> ‘123’

‘123’

>>> ‘7’8’’9’

‘7’8’’9’

Срезы. Срез – это механизм гибкого управления строкой на основе индексации. Можно получить любой символ строки по его индексу. Первый символ имеет индекс 0. Подстрока может быть определена с помощью среза – двух индексов, разделенных двоеточием:

word = ‘strength’

>>> word[4]

n

>>> word[0:2]

st

>>> word[:3]

str

>>> word[5:]

gth

Можно выбирать последовательность символов из строки с определенной цикличностью:

>>> s = ‘1234567890’

>>> s[::2]

‘13579’

>>> s[1:10:2]

‘13579’

>>> s[::-1]

‘0987654321’

**Операции со строками.** Строки можно склеивать с помощью оператора + :

>>> var = ‘Moscow’ + ‘city’

Между двух строк подряд вообще можно ничего не ставить, и они будут сконцентрированы.

Строки можно умножать с помощью оператора \*:

>>> ‘123’ \* 3

‘123123123’

Пример 1. Работа со строкой.

Задание: в каждом слово в строке изменить порядок следования цифр и букв (цифры спереди, буквы сзади).

I. Постановка задачи

Даны переменные a и b. Выполнить проверку на максимальное число.

II. x – исходные (ввод)

f – результирующие

III. Создание блок-схемы



Рис. 8.1 Блок-схема алгоритма из примера

IV. Код программы

x = ‘12sd54 sedfg321f4f ‘

c, v = ‘‘, ‘‘

f= ‘‘

for i in range(0, len(x)):

if x[i]!=‘ ‘:

if x[i].isdigit():

c += x[i]

else:

v += x[i]

else:

f += c+v+’ ‘

c, v = ‘‘, ‘‘

print(f’{f}’)

8.2. СПИСКИ

Список (list) – это структура данных для хранения объектов различных типов. Список похож на массив, только, в нем можно хранить объекты различных типов. Размер списка не статичен, его можно изменять. Список по своей природе является изменяемым типом данных.

Переменная, определяемая как список, содержит ссылку на структуру в памяти, которая в свою очередь хранит ссылки на какие-либо другие объекты или структуры.

При создании в памяти резервируется область, которую можно условно назвать некоторым “контейнером’, в котором хранятся ссылки на другие элементы данных в памяти. В отличии от таких типов данных как число или строка, содержимое “контейнера’ списка можно менять. На рис. 9.1. представлен процесс создания списка. Изначально был создан список содержащий ссылки на объекты 1 и 2, после операции *a[1] = 3*, вторая ссылка в списке стала указывать на объект 3.

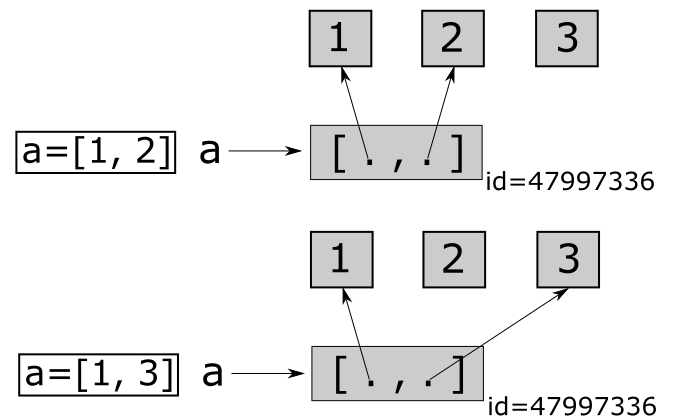


Рис. 9.1. Механизм создания списка

*Пример 8.2*: Создание списка

>>> a = []

>>> type(a)

<class ‘list’>

>>> a = [1, 2, 3]

>>> type(a)

<class ‘list’>

>>> a = [1, 3, 5, 7]

>>> b = a[:]

>>> print(b)

[1, 3, 5, 7]

Добавление элемента в список осуществляется с помощью метода append().

*Пример 8.3*: Добавление элемента в список

>>> a = []

>>> a.append(3)

>>> a.append(‘hello’)

>>> print(a)

[3, ‘hello’]

Для удаления элемента из списка, в случае, если вы знаете его значение, используйте метод remove(x), при этом будет удалена первая ссылка на данный элемент.

*Пример 8.4*: Удаление элемента

>>> b = [2, 3, 5]

>>> print(b)

[2, 3, 5]

>>> b.remove(3)

>>> print(b)

[2, 5]

Удаление элемента по его индексу, выполняется командой del имя\_списка[индекс].

*Пример 8.5*: Удаление элемента по его индексу

>>> c = [3, 5, 1, 9, 6]

>>> print(c)

[3, 5, 1, 9, 6]

>>> del c[2]

>>> print(c)

[3, 5, 9, 6]

Изменить значение элемента списка, зная его индекс, можно напрямую к нему обратившись.

Для получения из списка некоторого подсписка в определенном диапазоне индексов, укажите начальный и конечный индекс в квадратных скобках, разделив их двоеточием.

*Пример 8.6*: Получения из списка подсписка

>>> a[1:4]

[5, 7, 10]

*Пример 8.7*: Найти расстояние между точками с заданными координатами в n-мерном пространстве.

import math

n = input(‘Количество измерений пространства: ‘)

print(‘Через пробел’)

a = input(‘введите ‘ + n + ‘ координаты 1-ой точки: ‘)

b = input(‘введите ‘ + n + ‘ координаты 2-ой точки: ‘)

n = int(n)

a = a.split()

b = b.split()

if len(a) != n or len(b) != n:

print(‘Неверный ввод!’)

exit()

sum\_sqr = 0

for i, j in zip(a, b):

sum\_sqr += (int(i)-int(j))\*\*2

distance = math.sqrt(sum\_sqr)

print(‘Расстояние между точками: %.2f’ % distance)

Методы списков представлены в табл. 8.1.

*Таблица 8.1.*

**Методы списков**

|  |  |  |
| --- | --- | --- |
| № п.п. | Имя метода | Назначение метода |
| 1 | list.append(x) | Добавляет элемент в конец списка. Ту же операцию можно сделать так a[len(a):] = [x]  a.append(3) |
| 2 | list.extend(L) | Расширяет существующий список за счет добавления всех элементов из списка L. Эквивалентно команде a[len(a):] = L.  a.extend(b) |
| 3 | list.insert(i, x) | Вставить элемент x в позицию i. Первый аргумент – индекс элемента после которого будет вставлен элемент x. a.insert(0, 5) |
| 4 | list.remove(x) | Удаляет первое вхождение элемента x из списка. |
| 5 | list.pop([i]) | Удаляет элемент из позиции i и возвращает его. Если использовать метод без аргумента, то будет удален последний элемент из списка.  >>> print(a.pop(2))  3 |

*Пример 8.8*: Реализация алгоритма сортировки выбором

#Заполняем список из 10 элементов

#случайными числами от 1 до 99 и

#выводим неотсортированный список на экран.

from random import randint

N = 10

arr = []

for i in range(N):

arr.append(randint(1, 99))

print(arr)

#В цикле переменная i хранит индекс ячейки,

#в которую записывается минимальный элемент.

#Сначала это будет первая ячейка.

i = 0

#N - 1, так как последний элемент

#обменивать уже не надо.

while i < N - 1:

#ПОИСК МИНИМУМА

#Сначала надо найти минимальное значение

#на срезе от i до конца списка.

#Переменная m будет хранить индекс ячейки

#с минимальным значением.

#Сначала предполагаем, что

#в ячейке i содержится минимальный элемент.

m = i

#Поиск начинаем с ячейки следующей за i.

j = i + 1

#Пока не дойдем конца списка,

while j < N:

#будем сравнивать значение ячейки j,

#со значением ячейки m.

if arr[j] < arr[m]:

#Если в j значение меньше, чем в m,

#сохраним в m номер найденного

#на данный момент минимума.

m = j

#Перейдем к следующей ячейке.

j += 1

#ОБМЕН ЗНАЧЕНИЙ

#В ячейку i записывается найденный минимум,

#а значение из ячейки i переносится

#на старое место минимума.

arr[i], arr[m] = arr[m], arr[i]

#ПЕРЕХОД К СЛЕДУЮЩЕЙ НЕОБРАБОТАННОЙ ЯЧЕЙКЕ

i += 1

#Вывод отсортированного списка

print(arr)

*Пример 8.9*: работа со списком.

Задание: на вход поступает список x, создать новый список z, где zi элемент является максимальным начиная от i до конца.

I. Постановка задачи

Создать список от i до кона списка, найти максимум и записать в новый список на i место.

II. x – исходные (ввод)

z – результирующие

III. Создание блок-схемы



Рис. 8.2 Блок-схема алгоритма из примера

IV. Код программы

x, z = [-5, 65,1,3,14,5,6,7,8], []

for i in range(0, len(x)):

u = [x[y] for y in range(i, len(x))]

z.append(max(u))

print(z)

8.3. КОРТЕЖИ (tuple)

Кортеж (tuple) – это **неизменяемая** структура данных, которая по своему подобию очень похожа на список.

Создание кортежей. Для создания пустого кортежа можно воспользоваться одной из следующих команд.

*Пример* 8.8*:* Создание кортежей

>>> a = ()

>>> print(type(a))

<class ‘tuple’>

>>> a = (1, 2, 3, 4, 5)

>>> print(type(a))

<class ‘tuple’>

>>> print(a)

(1, 2, 3, 4, 5)

Доступ к элементам кортежа. Доступ к элементам кортежа осуществляется также как к элементам списка – через указание индекса. Но, как уже было сказано – изменять элементы кортежа нельзя.

Контрольные вопросы

1. Назовите четыре базовых типа данных в языке Python.
2. Почему они называются базовыми?
3. Назовите два способа создания списка, содержащего пять целочисленных значений, равных нулю.
4. Назовите два способа создания словаря с двумя ключами ‘a’ и ‘b’, каждый из которых ассоциирован со значением 0.
5. Назовите четыре операции, которые изменяют непосредственно объект списка.

Задания для самопроверки

1. Можно ли умножать сроку на число?

2. Как обратится к элементу словаря, не зная ключа?

3. Как поменять значения 2 элементов в 1 строку?

4. Как создать многомерный список?

5. Какие генераторы списков вы знатете?

9. Функция

Функция в программировании представляет собой обособленный участок кода, который можно вызывать, обратившись к нему по имени, которым он был назван. При вызове происходит выполнение команд тела функции. Внедрение функций позволяет решить проблему дублирования кода в разных местах программы. Благодаря им можно исполнять один и тот же участок кода не сразу, а только тогда, когда он понадобится.

**Оператор def.** В языке программирования Python функции определяются с помощью оператора def. Ключевое слово def сообщает интерпретатору, что перед ним определение функции. За def следует имя функции.

def function\_name([parameters]): #`parameters`: параметры функции (через запятую)

suite #Тело функции

Оно может быть любым, также, как и всякий идентификатор, например, переменная. В программировании весьма желательно давать всему осмысленные имена. Польза функций не только в возможности многократного вызова одного и того же кода из разных мест программы. Не менее важно, что, благодаря им программа обретает истинную структуру. Функции как бы разделяют ее на обособленные части, каждая из которых выполняет свою конкретную задачу.

Пусть надо написать программу, вычисляющую площади разных фигур. Пользователь указывает, площадь какой фигуры он хочет вычислить. После этого вводит исходные данные. Например, длину и ширину в случае прямоугольника. Чтобы разделить поток выполнения на несколько ветвей, следует использовать оператор if-elif-else:

*Пример 9.1*: Программа вычисления площади фигур

def rectangle():

a = float(input(‘Ширина: ‘))

b = float(input(‘Высота: ‘))

print(‘Площадь: %.2f’ % (a\*b))

def triangle():

a = float(input(‘Основание: ‘))

h = float(input(‘Высота: ‘))

print(‘Площадь: %.2f’ % (0.5 \* a \* h))

def circle():

r = float(input(‘Радиус: ‘))

print(‘Площадь: %.2f’ % (3.14 \* r\*\*2))

figure = input(‘1-прямоугольник, 2-треугольник, 3-круг: ‘)

if figure == ‘1’:

rectangle()

elif figure == ‘2’:

triangle()

elif figure == ‘3’:

circle()

else:

print(‘Ошибка ввода’)

**Локальные и глобальные переменные.** В программировании особое внимание уделяется концепции о локальных и глобальных переменных, а также связанное с ними представление об областях видимости. Соответственно, локальные переменные видны только в локальной области видимости, которой может выступать отдельно взятая функция. Глобальные переменные видны во всей программе. ‘Видны’ – значит, известны, доступны. К ним можно обратиться по имени и получить связанное с ними значение.

К глобальной переменной можно обратиться из локальной области видимости. К локальной переменной нельзя обратиться из глобальной области видимости, потому что локальная переменная существует только в момент выполнения тела функции. При выходе из нее, локальные переменные исчезают. Компьютерная память, которая под них отводилась, освобождается. Когда функция будет снова вызвана, локальные переменные будут созданы заново.

**Возврат значений из функции. Оператор return.** Функции могут передавать какие-либо данные из своих тел в основную ветку программы. Говорят, что функция возвращает значение. В большинстве языков программирования, в том числе Python, выход из функции и передача данных в то место, откуда она была вызвана, выполняется оператором return.

Если интерпретатор Питона, выполняя тело функции, встречает return, то он ‘забирает’ значение, указанное после этой команды, и ‘уходит’ из функции. Или при встрече оператора return в коде Python немедленно завершает выполнение функции, аналогично break для циклических конструкций.

*Пример 9.2*: Применение оператора return

def cylinder():

r = float(input())

h = float(input())

#площадь боковой поверхности цилиндра:

side = 2 \* 3.14 \* r \* h

#площадь одного основания цилиндра:

circle = 3.14 \* r\*\*2

#полная площадь цилиндра:

full = side + 2 \* circle

return full

square = cylinder()

print(square)

В функции может быть несколько операторов return. Однако всегда выполняется только один из них. Тот, которого первым достигнет поток выполнения.

**Возврат нескольких значений.** В Питоне позволительно возвращать из функции несколько объектов, перечислив их через запятую после команды return:

*Пример 9.3*:Возврат нескольких значений

def cylinder():

r = float(input())

h = float(input())

side = 2 \* 3.14 \* r \* h

circle = 3.14 \* r\*\*2

full = side + 2 \* circle

return side, full

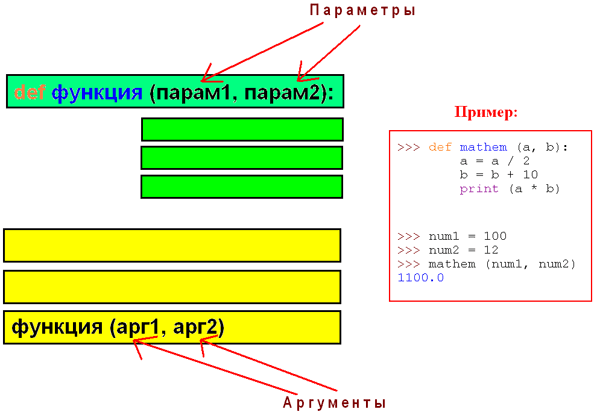
sCyl, fCyl = cylinder()

print(‘Площадь боковой поверхности %.2f’ % sCyl’)

print(‘Полная площадь %.2f’ % fCyl’)

**Параметры и аргументы функции.** В программировании функции могут не только возвращать данные, но также принимать их, что реализуется с помощью так называемых параметров, которые указываются в скобках в заголовке функции. Количество параметров может быть любым.

Параметры представляют собой локальные переменные, которым присваиваются значения в момент вызова функции. Конкретные значения, которые передаются в функцию при ее вызове, будем называть аргументами. Следует иметь в виду, что встречается иная терминология. Например, формальные параметры и фактические параметры. В Python же обычно все называют аргументами.



Когда функция вызывается, то ей передаются аргументы. В примере указаны глобальные переменные num1 и num2. Однако на самом деле передаются не эти переменные, а их значения. В данном случае числа 100 и 12.

Когда интерпретатор переходит к функции, чтобы начать ее исполнение, он присваивает переменным-параметрам переданные в функцию значения-аргументы. В примере переменной a будет присвоено 100, b будет присвоено 12.

Изменение значений a и b в теле функции никак не скажется на значениях переменных num1 и num2. Они останутся прежними. В Python такое поведение характерно для неизменяемых типов данных, к которым относятся, например, числа и строки. Говорят, что в функцию данные передаются по значению. Так, когда a присваивалось число 100, то это было уже другое число, не то, на которое ссылается переменная num1. Число 100 было скопировано и помещено в отдельную ячейку памяти для переменной a.

Существуют изменяемые типы данных. Для Питона, это, например, списки и словари. В этом случае данные передаются по ссылке. В функцию передается ссылка на них, а не сами данные. И эта ссылка связывается с локальной переменной. Изменения таких данных через локальную переменную обнаруживаются при обращении к ним через глобальную. Это есть следствие того, что несколько переменных ссылаются на одни и те же данные, на одну и ту же область памяти.

Необходимость передачи по ссылке связана в первую очередь с экономией памяти. Сложные типы данных, по сути представляющие собой структуры данных, обычно копировать не целесообразно. Однако, если надо, всегда можно сделать это принудительно.

Обратим внимание еще на один момент. Количество аргументов и параметров совпадает. Нельзя передать три аргумента, если функция принимает только два. Нельзя передать один аргумент, если функция требует два обязательных. В рассмотренном примере они обязательные.

Однако в Python у функций бывают параметры, которым уже присвоено значение по умолчанию. В таком случае, при вызове можно не передавать соответствующие этим параметрам аргументы. Хотя можно и передать. Тогда значение по умолчанию заменится на переданное.

*Пример 9.4*: Присвоение значений по умолчанию

def cylinder(h, r = 1):

side = 2 \* 3.14 \* r \* h

circle = 3.14 \* r\*\*2

full = side + 2 \* circle

return full

figure1 = cylinder(4, 3)

figure2 = cylinder(5)

print(figure1)

print(figure2)

Python, как и многие другие языки, позволяет создавать собственные (пользовательские) функции, среди которых можно выделить четыре типа (пример 9.5):

* глобальные(1). Доступны из любой точки программного кода в том же модуле или из других модулей;
* локальные (вложенные)(2). Объявляются внутри других функций и видны только внутри них: используются для создания вспомогательных функций, которые нигде больше не используются.
* анонимные (3). Не имеют имени и объявляются в месте использования. В Python и представлены лямбда-выражениями;
* методы (4). Функции, ассоциированные с каким-либо объектом (например, list.append(), где append() - метод объекта list).

*Пример 9.5*: Четыре типа функций в Python

class Car:

def move(self, x): #Метод (4)

self.x += x

def sum\_of\_cubes(x, y): #Глобальная функция (1)

#Локальная функция (2) (ее ‘видит’ только код внутри sum\_of\_cubes())

def cube(a):

return a\*\*3

return cube(x) + cube(y) #return возвращает результат выполнения тому,

#кто вызвал эту функцию

players = [{‘name’: ‘Юрий’, ‘rank’: 5},

{‘name’: ‘Сергей’, ‘rank’: 3},

{‘name’: ‘Максим’, ‘rank’: 4}]

#Анонимная функция (3) (лямбда-выражение)

#В функции sorted() используется для определения порядка сортировки

print(sorted(players, key=lambda player: player[‘name’])) #Сортировка по name

#[{‘rank’: 4, ‘name’: ‘Максим’}, {‘rank’: 3, ‘name’: ‘Сергей’}, {‘rank’: 5, ‘name’: ‘Юрий’}]

print(sorted(players, key=lambda player: player[‘rank’])) #Сортировка по rank

#[{‘rank’: 3, ‘name’: ‘Сергей’}, {‘rank’: 4, ‘name’: ‘Максим’}, {‘rank’: 5, ‘name’: ‘Юрий’}]

*Пример 9.6*: Функция сортировки выбором

from random import randint

def sel\_sort(array):

for i in range(len(array) - 1):

m = i

j = i + 1

while j < len(array):

if array[j] < array[m]:

m = j

j = j + 1

array[i], array[m] = array[m], array[i]

a = []

for i in range(10):

a.append(randint(1, 99))

print(a)

sel\_sort(a)

print(a)

Глобальные и локальные функции. Позиционные и ключевые параметры/аргументы. Все параметры, указываемые в Python при объявлении и вызове функции, делятся на:

* позиционные: указываются простым перечислением:

def function\_name(a, b, c): #a, b, c - 3 позиционных параметра

pass

* ключевые: указываются перечислением ключ=значение:

def function\_name(key=value, key2=value2): #key, key2 - 2 позиционных аргумента

pass #value, value2 - их значения по умолчанию

Позиционные и ключевые аргументы могут быть скомбинированы. Синтаксис объявления и вызова функции зависит от типа параметра, однако позиционные параметры (и соответствующие аргументы) всегда идут перед ключевыми:

* объявление функции:

def example\_func(a, b, c): #можно : ‘a’, ‘b’, ‘c’ - позиционные параметры

pass

def example\_func(a, b, c=3): #можно : ‘a’, ‘b’ - позиционные параметры,

pass # ‘c’ - ключевой параметр

def example\_func(a=1, b=2, c=3): #можно : ‘a’, ‘b’, ‘c’ - ключевые параметры

pass

def example\_func(a=1, с, b=2): #нельзя: ключевой параметр ‘a’

pass #идет раньше позиционнных

**Упаковка и распаковка аргументов.** В ряде случаев бывает полезно определить функцию, способную принимать любое число аргументов. Так, например, работает функция print(), которая может принимать на печать различное количество объектов и выводить их на экран.

Достичь такого поведения можно, используя механизм упаковки аргументов, указав при объявлении параметра в функции один из двух символов:

\*: все позиционные аргументы начиная с этой позиции и до конца будут собраны в кортеж;

\*\*: все ключевые аргументы начиная с этой позиции и до конца будут собраны в словарь.

*Пример 9.7*: Позиционные аргументы

#При упаковке аргументов все переданные позиционные аргументы

#будут собраны в кортеж ‘order’, а ключевые - в словарь ‘info’

def print\_order(\*order, \*\*info):

print(‘Музыкальная библиотека №1\n’)

#Словарь ‘infos’ должен содержать ключи ‘author’ и ‘birthday’

for key, value in sorted(info.items()):

print(key, ‘:’, value)

#Кортеж ‘order’ содержит все наименования произведений

print(‘Вы выбрали:’)

for item in order:

print(‘ -’, item)

print(‘\nПриходите еще!’)

print\_order(‘Славянский марш’, ‘Лебединое озеро’, ‘Спящая красавица’,

‘Пиковая дама’, ‘Щелкунчик’,

author=‘П.И. Чайковский’, birthday=‘07/05/1840’)

Пример 9.8: работа со вложенной подпрограммой.

Задание: найти среднее арифметическое n последовательностей.

I. Постановка задачи

Создать подпрограмму для подсчета среднего арифметического неограниченного количества последовательностей.

II. print\_mean\_sequences()– исходные (ввод)

k, mean(v)– результирующие

Вид подпрограммы – функция print\_mean\_sequences

Назначение – нахождение среднего арифметического

Входные параметры – \*\*kwargs

Выходные параметры – список и название списка

Вид подпрограммы – функция mean

Назначение – сумма всех элементов списка

Входные параметры – data

Выходные параметры – число типа float

III. Создание блок-схемы



Рис. 9.1 Блок-схема алгоритма из примера

IV. Код программы

def print\_mean\_sequences(\*\*kwargs):

def mean(data):

return sum(data) / float(len(data))

for k, v in kwargs.items():

print(k, mean(v))

print\_mean\_sequences(x=[1, 8, 3], y=[3, 3, 0])

Контрольные вопросы

1. Перечислите возможности объявления функции.
2. Дайте определение функции.
3. Сформулируйте определение локальной переменной
4. Дайте определение глобальной переменной
5. Сформулируйте назначение слова return.

Задание для самопроверки

1. Как создать класс?

2. Как вызвать класс?

3. можно ли создать функцию в функции в классе?

4. Можно ли обратится к функции в функции в классе, если да, то как?

5. Как в классе задаются входные данные

6. Для чего нужен метод \_\_init\_\_

10. Файлы

Большие объемы данных имеет смысл хранить не в списках или словарях, а в файлах. Поэтому в языках программирования предусмотрена возможность работы с файлами. В Python файлы рассматриваются как объекты файловых классов, то есть, например, текстовый файл – это тип данных наряду с типами списка, словаря, целого числа и др.

Обычно файлы делят на текстовые и байтовые (бинарные). Первые рассматриваются как содержащие символьные данные, строки. Вторые – как поток байтов. Побайтово считываются, например, файлы изображений.

Работа с бинарными файлами несколько сложнее. Нередко их обрабатывают с помощью специальных модулей Python (pickle, struct). В этом уроке будут рассмотрены базовые приемы чтения текстовых файлов и записи в них.

**Функция open() – открытие файла.** Открытие файла выполняется с помощью встроенной в Python функции open(). Обычно ей передают один или два аргумента. Первый – имя файла или имя с адресом, если файл находится не в том каталоге, где находится скрипт. Второй аргумент – режим, в котором открывается файл.

Обычно используются режимы чтения (‘r’) и записи (‘w’). Если файл открыт в режиме чтения, то запись в него невозможна. Можно только считывать данные из него. Если файл открыт в режиме записи, то в него можно только записывать данные, считывать нельзя.

Если файл открывается в режиме ‘w’, то все данные, которые в нем были до этого, стираются. Файл становится пустым. Если не надо удалять существующие в файле данные, тогда следует использовать вместо режима записи, режим дозаписи (‘a’).

Если файл отсутствует, то открытие его в режиме ‘w’ создаст новый файл. Бывают ситуации, когда надо гарантировано создать новый файл, избежав случайной перезаписи данных существующего. В этом случае вместо режима ‘w’ используется режим ‘x’. В нем всегда создается новый файл для записи. Если указано имя существующего файла, то будет выброшено исключение. Потери данных в уже имеющемся файле не произойдет.

Если при вызове open() второй аргумент не указан, то файл открывается в режиме чтения как текстовый файл. Чтобы открыть файл как байтовый, дополнительно к букве режима чтения/записи добавляется символ ‘b’. Буква ‘t’ обозначает текстовый файл. Поскольку это тип файла по умолчанию, то обычно ее не указывают.

Нельзя указывать только тип файла, то есть open(‘имя\_файла’, ‘b’) есть ошибка, даже если файл открывается на чтение. Правильно – open(‘имя\_файла’, ‘rb’). Только текстовые файлы можно открыть командой open(‘имя\_файла’), потому что и ‘r’ и ‘t’ подразумеваются по-умолчанию.

Функция open() возвращает объект файлового типа. Его надо либо сразу связать с переменной, чтобы не потерять, либо сразу прочитать.

**Чтение файла.** С помощью файлового метода read() можно прочитать файл целиком или только определенное количество байт. Пусть имеется файл data.txt с таким содержимым:

one - 1 - I

two - 2 - II

three - 3 - III

four - 4 - IV

five - 5 – V

Откроем файл и почитаем:

>>> f1 = open(‘data.txt’)

>>> f1.read(10)

‘one - 1 - ‘

>>> f1.read()

‘I\ntwo - 2 - II\nthree - 3 - III\nfour - 4 - IV\nfive - 5 - V\n’

>>> f1.read()

‘‘

>>> type(f1.read())

<class ‘str’>

Сначала считываются первые десять байтов, которые равны десяти символам. Это не бинарный файл, но файл можно читать по байтам. Последующий вызов read() считывает весь оставшийся текст. После этого объект файлового типа f1 становится пустым.

Заметим, что метод read() возвращает строку, и что конец строки считывается как ‘\n’.

Для того, чтобы читать файл построчно существует метод readline():

>>> f1 = open(‘data.txt’)

>>> f1.readline()

‘one - 1 - I\n’

>>> f1.readline()

‘two - 2 - II\n’

>>> f1.readline()

‘three - 3 — III\n’

Метод readlines() считывает сразу все строки и создает список:

>>> f1 = open(‘data.txt’)

>>> f1.readlines()

[‘one - 1 - I\n’, ‘two - 2 - II\n’, ‘three - 3 - III\n’, ‘four - 4 - IV\n’, ‘five - 5 - V\n’]

**Запись в файл.** Запись в файл выполняется с помощью методов write() и writelines(). Во второй можно передать структуру данных:

>>> l = [‘tree’, ‘four’]

>>> f2 = open(‘newdata.txt’, ‘w’)

>>> f2.write(‘one’)

3

>>> f2.write(‘ two’)

4

>>> f2.writelines(l)

Метод write() возвращает количество записанных символов.

**Закрытие файла.** После того как работа с файлом закончена, важно не забывать его закрыть, чтобы освободить место в памяти. Делается это с помощью файлового метода close(). Свойство файлового объекта closed позволяет проверить закрыт ли файл.

>>> f1.close()

>>> f1.closed

True

>>> f2.closed

False

Пример 10.1: работа с файлом.

Задание: Целочисленная матрица, в которой число строк n вдвое меньше числа столбцов, хранится в текстовом файле. Сохранить в этом файле матрицу [n\*n], удалив последние n столбцов.

I. Постановка задачи

Целочисленная матрица, в которой число строк n вдвое меньше числа столбцов,

хранится в текстовом файле. Сохранить в этом файле матрицу [n\*n], удалив последние n

столбцов.

II. x– исходные (ввод)

файл– результирующие

Вид подпрограммы – функция sew

Назначение – нахождение всех чисел с последующим занесением их в список

Входные параметры – a

Выходные параметры – список (b)

Вид подпрограммы – метод rerait

Назначение – создание матрицы n\*n в этом же файле

Входные параметры – функция sew

Выходные параметры – отсутствуют

III. Создание блок-схемы

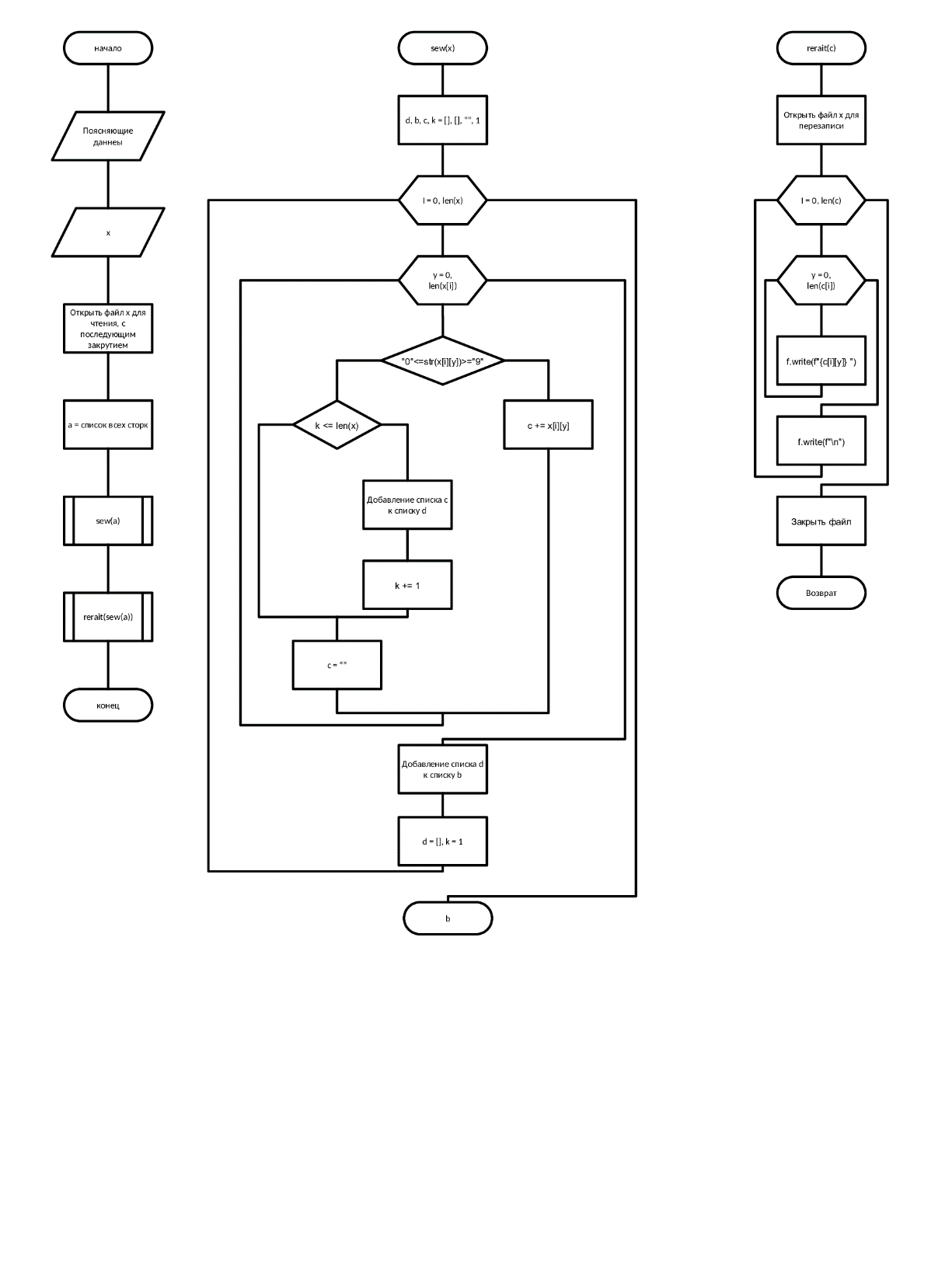


Рис. 10.1 Блок-схема алгоритма из примера

IV. Код программы

x = input(f’Введите название файла, c его расширение:’)  
with open(f’{x}’) as f:  
 a = f.readlines()  
  
  
 def sew(x):  
 d = []  
 b = []  
 c = ‘‘  
 k = 1  
 for i in range(0, len(x)):  
 for y in range(0, len(x[i])):  
 if x[i][y].isdigit():  
 c += x[i][y]  
 else:  
 if k <= len(x):  
 d.append(c)  
 k += 1  
 c = ‘‘  
 b.append(d)  
 d = []  
 k = 1  
 return b  
  
  
def rerait(c):  
 f = open(f’{x}’, ‘w’)  
 for i in range(0, len(c)):  
 for y in range(0, len(c[i])):  
 f.write(f’{c[i][y]} ‘)  
 f.write(f’\n’)  
 f.close()  
  
  
rerait(sew(a))

3) k, mean(v) – вывод

III. Создание блок-схемы

Контрольные вопросы

1. Сформулируйте определение файла.
2. Укажите различия между текстовым и байтовым (бинарным) файлом.
3. Перечислите операции, выполняемые над файлами.
4. Назовите стандартные процедуры и функции применимые к файлам.
5. Приведите пример использования файлов.

Задания для самопроверки

1. Как открыть файл для прочтения и записи

2. Что обязательно нужно писать в конце при простом открытии файла( а = open(“test.txt’r)

3. Что при прочтении файла, состоящего из нескольких строк, находится в конце строки?

11. Модули и пакеты

Модули и пакеты являются неотъемлемой частью модульного программирования - организации программы как совокупности небольших независимых блоков, структура и поведение которых подчиняются определенным правилам.

Разработка программы как совокупности модулей позволяет:

* упростить задачи проектирования программы и распределения процесса разработки между группами разработчиков;
* предоставить возможность обновления (замены) модуля, без необходимости изменения остальной системы;
* упростить тестирование программы;
* упростить обнаружение ошибок.

Программный код часто разбивается на несколько файлов, каждый из которых используется отдельно от остальных. Одним из методов написания модульных программ является объектно-ориентированное программирование.

11.1. Основные понятия

Модуль (англ. Module) - специальное средство языка программирования, позволяющее объединить вместе данные и функции и использовать их как одну функционально-законченную единицу (например, математический модуль, содержащий тригонометрические и прочие функции, константы π, ϵ и т.д.).

Пакеты (англ. Package) являются еще более крупной единицей и представляют собой набор взаимосвязанных модулей, предназначенных для решения задач определенного класса некоторой предметной области (например, пакет для решения систем уравнений, который может включать математический модуль, модуль со специальными типами данных и т.д.).

11.2. Модули и пакеты в Python

Модуль - отдельный файл с кодом на Python, содержащий функции и данные:

* имеет расширение \*.py (имя файла является именем модуля);
* может быть импортирован (подключен) (директива import ...);
* может быть многократно использован.

Пакеты в Python — это способ структуризации модулей. Пакет представляет собой папку, в которой содержатся модули и другие пакеты, и обязательный файл \_\_init.py\_\_, отвечающий за инициализацию пакета.

Так, например, пакет xml имеет следующую структуру:

C:\USERS\YURI\APPDATA\LOCAL\PROGRAMS\PYTHON\PYTHON35\LIB\xml

│ \_\_init\_\_.py Файл инициализации пакета xml

│

├───dom Вложенный пакет xml.dom

│ domreg.py

│ expatbuilder.py

│ minicompat.py

│ minidom.py

│ NodeFilter.py

│ pulldom.py

│ xmlbuilder.py

│ \_\_init\_\_.py

│

├───etree Вложенный пакет xml.etree

│ cElementTree.py

│ ElementInclude.py

│ ElementPath.py

│ ElementTree.py

│ \_\_init\_\_.py

│

├───parsers Вложенный пакет xml.parsers

│ expat.py

│ \_\_init\_\_.py

│

└───sax Вложенный пакет xml.sax

expatreader.py

handler.py

saxutils.py

xmlreader.py

\_exceptions.py

\_\_init\_\_.py

11.3. Подключение и использование

Для использования модуля или пакета в коде необходимо его предварительно подключить (импортировать - import).

Импорт модуля или пакета выполняется единожды инструкцией [import](https://www.yuripetrov.ru/edu/python/ch_09_01.html#import), располагаемой, как правило, в начале файла.

Выполнить подключение модуля можно несколькими способами:

#**Способ** №1

#В данном способе обратиться к члену модуля можно с указанием модуля,

#например, module\_1.object\_1

#Импортирует модуль ‘module\_1’

import module\_1

#Импортирует модули ‘module\_1’, ‘module\_2’,..., ‘module\_n’

import module\_1, module\_2, ..., module\_n

#Импортирует модуль ‘module\_1’ под псевдонимом ‘preferred\_name’

import module\_1 as preferred\_name

#**Способ** №2

#В данном способе обратиться к члену модуля можно без указания модуля,

#например, object\_1

#Импортирует ‘object’ под псевдонимом ‘preferred\_name’ из модуля ‘module\_1’

from module\_1 import object as preferred\_name

#Импортирует объекты ‘object\_1’, ..., ‘object\_n’ из модуля ‘module\_1’

from module\_1 import object\_1, object\_2, ..., object\_n

#Импортирует все объекты из модуля ‘module\_1’

from module\_1 import \*

Контрольные вопросы

1. Модульное программирование. Понятие модуля и пакета, их реализация в языке Python.
2. Классификация модулей/пакетов в Python.
3. Подключение и использование модулей/пакетов. Инструкция import.
4. Область поиска модулей. Общая схема импорта.
5. Специальные и дополнительные атрибуты модуля.

Задание для самопроверки

1. Как установить пакет?

2. Как создать свой пакет?

3. Можно ли взять все/одну функци(и/ю) из пакета?

4. Можно ли изменить существующий пакет?

12. Графика

Черепашья графика - популярный способ познакомить детей с программированием. Это была часть оригинального языка программирования логотипов, разработанного Уолли Ферзейгом, Сеймуром Папертом и Синтией Соломон в 1967 году.

Представьте себе роботизированную черепаху, начинающуюся с (0, 0) в плоскости x-y. После импорта turtle дайте ему команду turtle.forward (15), и он перемещается (на экране!) на 15 пикселей в направлении, в котором он обращен, рисуя линию по мере движения. Дайте ему команду turtle.right(25), и он повернется на месте на 25 градусов по часовой стрелке.

Комбинируя вместе эти и подобные команды, можно легко рисовать замысловатые формы и рисунки.

Модуль turtle — это расширенная повторная реализация одноименного модуля из стандартного дистрибутива Python вплоть до версии Python 2.5.

Он пытается сохранить достоинства старого модуля turtle и быть (почти) на 100% совместимым с ним. Это означает, в первую очередь, дать возможность обучающемуся программисту использовать все команды, классы и методы в интерактивном режиме при использовании модуля из IDLE run с переключателем -n.

Модуль turtle предоставляет графические примитивы turtle как объектно-ориентированными, так и процедурно-ориентированными способами. Поскольку он использует tkinter для базовой графики, ему нужна установленная версия Python с поддержкой Tk.

Объектно-ориентированный интерфейс использует, по существу, два + два класса:

Класс TurtleScreen определяет графические окна как игровую площадку для рисования черепах. Его конструктору нужен tkinter.Canvas или ScrolledCanvas в качестве аргумента. Его следует использовать, когда черепаха используется как часть какого-либо приложения.

Функция Screen() возвращает одноэлементный объект подкласса TurtleScreen. Эту функцию следует использовать, когда turtle используется в качестве автономного инструмента для создания графики. Как одноэлементный объект, наследование от его класса невозможно.

Все методы TurtleScreen/Screen также существуют как функции, то есть как часть процедурно-ориентированного интерфейса.

RawTurtle (псевдоним: RawPen) определяет объекты Turtle, которые рисуются на TurtleScreen. Его конструктору нужен Canvas, ScrolledCanvas или TurtleScreen в качестве аргумента, чтобы объекты RawTurtle знали, где рисовать.

Производным от RawTurtle является подкласс Turtle (псевдоним: Pen), который рисует на “экранном’ экземпляре, который создается автоматически, если он еще не присутствует.

Все методы RawTurtle/Turtle также существуют как функции, т.е. часть процедурно-ориентированного интерфейса.

Процедурный интерфейс предоставляет функции, которые являются производными от методов классов Screen и Turtle. Они имеют те же имена, что и соответствующие методы. Экранный объект автоматически создается всякий раз, когда вызывается функция, производная от экранного метода. (Безымянный) объект turtle автоматически создается всякий раз, когда вызывается любая из функций, производных от метода Turtle.

Чтобы использовать несколько черепах на экране, необходимо использовать объектно-ориентированный интерфейс.

Пример работы с графикой: исследовать область определения и выбрав расположение координатных осей.

Пример 11.1: построить на экране график функции используя процедуру построения графика функции y = F(x) точечным методом.

II. Исходные данные — отсутствуют

Результирующие — график

Вид подпрограммы – Class CoordinateLine

Назначение – создание координатной плоскости

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – функция \_\_init\_\_

Назначение – инициализация переменных

Входные параметры – self, segments, distance\_pixel

Выходные параметры – отсутствуют

Вид подпрограммы – функция line

Назначение – создание координатных линий и отсечек

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – функция coordinates

Назначение – добавление координат на осях

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – функция x\_segments

Назначение – добавление координат и имени направления

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – функция unt\_x\_segments

Назначение – добавление координат и имени направления

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – функция y\_segments

Назначение – добавление координат и имени направления

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – функция unt\_y\_segments

Назначение – добавление координат и имени направления

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – class DotInLine

Назначение – добавление точек согласно уравнению

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – функция \_\_init\_\_

Назначение – инициализация переменных

Входные параметры – self, segments, distance\_pixel

Выходные параметры – отсутствуют

Вид подпрограммы – функция float\_range

Назначение – диапазон переменных с плавающей точкой

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

Вид подпрограммы – функция step\_dot

Назначение – добавление точек согласно уравнению

Входные параметры – отсутствуют

Выходные параметры – отсутствуют

III. Создание блок-схемы 

Рис. 11.1 Блок-схема алгоритма из примера



Рис.11.2 Блок-схема алгоритма из примера

IV. Код программы

import turtle

t = turtle.Turtle()

window = turtle.Screen()

t.speed(0)

class CoordinateLine:

‘‘‘Создание декартовой системы координат’’’

def \_\_init\_\_(self, segments: int = 12, distance\_pixel: int = 25):

self.segments = segments

self.distance\_pixel = distance\_pixel

‘‘‘Создание осей координат и отступа для осей’’’

def line(self):

for x in range(4):

for y in range(self.segments):

t.forward(self.distance\_pixel)

t.right(90)

t.forward(5)

t.backward(10)

t.forward(5)

t.left(90)

t.home()

t.right(90 \* x + 90)

‘‘‘Наименование осей и интервал между отрезками’’’

def coordinates(self):

t.pencolor(‘blue’)

def null():

t.penup()

t.setpos(4, 1)

t.pendown()

t.write(‘0’, font=(‘Arial’, 16, ‘bold’))

null()

def x\_segments():

t.penup()

t.setpos(self.segments \* self.distance\_pixel, 4)

t.pendown()

t.write(‘X’, font=(‘Arial’, 16, ‘bold’))

t.penup()

t.setpos(self.distance\_pixel - 3, -20)

b = 1

t.pendown()

for i in range(self.segments):

t.write(b)

t.penup()

t.forward(self.distance\_pixel)

t.pendown()

b += 1

x\_segments()

def unt\_x\_segments():

t.penup()

t.setpos(-self.segments \* self.distance\_pixel, 4)

t.pendown()

t.write(‘-X’, font=(‘Arial’, 16, ‘bold’))

t.penup()

t.setpos(-self.distance\_pixel - 5, -20)

b = -1

t.pendown()

for i in range(self.segments):

t.write(b)

t.penup()

t.backward(self.distance\_pixel)

t.pendown()

b -= 1

unt\_x\_segments()

def y\_segments():

t.penup()

t.setpos(4, self.segments \* self.distance\_pixel)

t.pendown()

t.write(‘Y’, font=(‘Arial’, 16, ‘bold’))

t.penup()

t.setpos(-20, self.distance\_pixel - 8)

b = 1

t.pendown()

t.left(90)

for i in range(self.segments):

t.write(b)

t.penup()

t.forward(self.distance\_pixel)

t.pendown()

b += 1

y\_segments()

def unt\_y\_segments():

t.penup()

t.setpos(4, -self.segments \* self.distance\_pixel)

t.pendown()

t.write(‘-Y’, font=(‘Arial’, 16, ‘bold’))

t.penup()

t.setpos(-20, -self.distance\_pixel - 8)

b = -1

t.pendown()

for i in range(self.segments):

t.write(b)

t.penup()

t.backward(self.distance\_pixel)

t.pendown()

b -= 1

unt\_y\_segments()

‘‘‘Диапазон чисел с плавающей точкой’’’

def float\_range(a, k=None, d=None):

if k is None:

k = a + 0.0

a = 0.0

if d is None:

d = 1.0

while True:

if d > 0 and a >= k:

break

elif d < 0 and a <= k:

break

yield ‘%g’ % a

a = a + d

class DotInLine:

‘‘‘Расстановка точек согласно уравнению y=F(x)’’’

def \_\_init\_\_(self, segments: int = 12, distance\_pixel: int = 25):

self.segments = segments

self.distance\_pixel = distance\_pixel

‘‘‘Создание точек согласно уравнению’’’

def step\_dot(self):

t.penup()

v = float(self.segments)

for x in float\_range(-v \* 5, 0, 0.9):

y = 1 / (float(x) + 1)

t.goto(5 \* float(x), y \* self.distance\_pixel)

t.dot(4, ‘Black’)

for x in float\_range(-5, 1, 0.06):

y = 1 / (float(x) + 1)

t.goto(5 \* float(x), y \* self.distance\_pixel)

t.dot(4, ‘Black’)

for x in float\_range(0, v \* 5, 0.9):

y = 1 / (float(x) + 1)

t.goto(5 \* float(x), y \* self.distance\_pixel)

t.dot(4, ‘Black’)

lines = CoordinateLine()

lines.line()

lines.coordinates()

coord = DotInLine()

coord.step\_dot()

window.mainloop()

Контрольные вопросы

1. Можно ли в turtle самостоятельно рисовать на экране
2. Есть ли в turtle предустановленные фигуры(круг, квадрат, треугольник)
3. Почему turtle не эффективен по сравнению с другими пакетами для рисования

Какое ограничение имеет turtle на размер экрана, в заключение ответ

ЗАКЛЮЧЕНИЕ

Разработка любой вычислительной системы вне зависимости от уровня ее сложности ставит разработчика перед необходимостью выбора тех или иных структур данных, в рамках которых осуществляется отображение соответствующей предметной области.

Описанный в пособии язык программирования предоставляет программисту возможность определять необходимые типы данных, наиболее полно соответствующие решаемой задаче. Это позволяет осуществлять ее моделирование на том уровне абстракции, который соответствует специфике задачи.

Пособие посвящено этапам разработки вычислительной системы, структурной организации данных и методам, лежащим в основе их обработки. Рассматриваются фундаментальные приемы и методы программирования, рассмотрены некоторые методы сортировок.

Излагаются практически хорошо обоснованные методы обработки данных. Ответ (10^154 пикселей)

Учебное пособие предлагает инструмент для применения основ алгоритмизации и программирования в практической деятельности.

Библиографический список

1. Северенс, Ч. Введение в программирование на Python / Ч. Северенс. -2-е изд., испр. -Москва : Национальный Открытый Университет ‘ИНТУИТ’, 2016. -231 с. : схем., ил. ; То же [Электронный ресурс]. -URL: <http://biblioclub.ru/index.php?page=book&id=429184>
2. Златопольский Д.М. Основы программирования на языке Python: Учебники/ Д.М. Златопольский - Издательство ‘ДМК Пресс’, 2017 То же [Электронный ресурс]. - URL: <https://e.lanbook.com/book/97359>
3. Хахаев И. А. Практикум по алгоритмизации и программированию на Python: курс / И. А. Хахаев - М.: Национальный Открытый Университет ‘ИНТУИТ’, 2016//ЭБС ‘Университетская библиотека online’ То же [Электронный ресурс]. -URL: <http://biblioclub.ru/index.php?page=book&id=429256&sr=1>
4. Тузовский А.Ф. Объектно-ориентированное программирование: Учебное пособие для прикладного бакалавриата / А.Ф. Тузовский - М.: Издательство Юрайт, 2018 // ЭБС ‘Юрайт’ <https://www.biblio-online.ru/book/obektno-orientirovannoe-programmirovanie-414163>
5. Митчелл Р. Скрапинг веб-сайтов с помощю Python: Самоучители и руководства / Р. Митчелл - Издательство ‘ДМК Пресс’, 2016 То же [Электронный ресурс]. - URL: https://e.lanbook.com/book/100903

Учебное издание

**Стативко** Роза Усмановна

Алгоритмы и структуры данных

Учебное пособие

Подписано в печать 20.03.20. Формат 60x84/16. Усл. печ. л. \_\_ . Уч.-изд. л.

Тираж 45 экз. Заказ Цена

Отпечатано в Белгородском государственном технологическом университете им. В. Г. Шухова

308012, г. Белгород, ул. Костюкова, 46