НАЧИНАЕМ ПРОГРАММИРОВАТЬ НА >>> PYTHON®

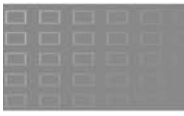
4-Е ИЗДАНИЕ

ТОНИ ГЭДДИС

bhv®

STARTING OUT WITH

PYTHON®



FOURTH EDITION

STARTING OUT WITH

PYTHON®

FOURTH EDITION

Tony Gaddis
Haywood Community College

330 Hudson Street, New York, NY 10013

ТОНИ ГЭДДИС

НАЧИНАЕМ ПРОГРАММИРОВАТЬ НА >>> PYTHON®

4-Е ИЗДАНИЕ

Санкт-Петербург
«БХВ-Петербург»
2019

УДК 004.438 Python
ББК 32.973.26-018.1
Г98

Гэддис Т.

Г98 Начинаем программировать на Python. — 4-е изд.: Пер. с англ. — СПб.: БХВ-Петербург, 2019. — 768 с.: ил.

ISBN 978-5-9775-4002-5

Изложены принципы программирования, с помощью которых читатель приобретет навыки алгоритмического решения задач на языке Python, даже не имея опыта программирования. Дано краткое введение в компьютеры и программирование. Рассмотрен ввод, обработка и вывод данных, управляющие структуры и булева логика, структуры с повторением, функции, файлы и исключения, списки и кортежи, строковые данные, словари и множества, классы и ООП, наследование, рекурсия, программирование интерфейса, функциональное программирование и др.

Для облегчения понимания сути алгоритмов широко использованы блок-схемы, псевдокод и другие инструменты. Приведено большое количество сжатых и практичных примеров программ. В каждой главе предложены тематические задачи с пошаговым анализом их решения.

Для начинающих программистов,
старших школьников и студентов первых курсов

УДК 004.438 Python
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Оформление обложки	*Марины Дамбиевой*

Authorized translation from the English language edition, entitled STARTING OUT WITH PYTHON, 4th Edition;
ISBN: 978-0-13-444432-1: by GADDIS, TONY; published by Pearson Education, Inc. Copyright © 2018 by Pearson Education, Inc.
All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. RUSSIAN language edition published by LLC "BHV". Copyright © 2018.

Авторизованный перевод с английского на русский язык произведения STARTING OUT WITH PYTHON, 4th Edition;
ISBN: 978-0-13-444432-1; GADDIS TONY, опубликованного Pearson Education, Inc. Copyright © 2018 by Pearson Education, Inc.
Все права защищены. Никакая часть этой книги не может быть воспроизведена или передана в любой форме или любыми средствами, будь то электронными или механическими, включая фотокопирование, запись или иное использование любой информационно-поисковой системы без разрешения Pearson Education, Inc. Русский перевод опубликован ООО "БХВ".

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-0-13-444432-1 (англ.)
ISBN 978-5-9775-4002-5 (рус.)

© 2018 by Pearson Education, Inc.
© Перевод на русский язык, оформление.
ООО "БХВ-Петербург", ООО "БХВ", 2019

Краткое оглавление

Оглавление ..... **XI**

Предисловие **1**
Прежде всего управляющие структуры и только потом классы 1
Краткий обзор глав 1
Организация учебного материала 4
Структурные элементы и условные обозначения книги 5
Дополнительные материалы 6
Электронный архив 7
Об авторе 7
Комментарий переводчика 7

Глава 1. Введение в компьютеры и программирование **13**
1.1 Введение 13
1.2 Аппаратное и программное обеспечение 14
1.3 Как компьютеры хранят данные 20
1.4 Как программа работает 24
1.5 Использование языка Python 32
Вопросы для повторения 36
Задачи 40

Глава 2. Ввод, обработка и вывод **43**
2.1 Проектирование программы 43
2.2 Ввод, обработка и вывод 48
2.3 Вывод данных на экран при помощи функции *print* 48
2.4 Комментарии 51
2.5 Переменные 52
2.6 Чтение входных данных с клавиатуры 62
2.7 Выполнение расчетов 66
2.8 Подробнее о выводе данных 79
2.9 Именованные константы 87
2.10 Введение в черепашью графику 88
Вопросы для повторения 114
Задачи по программированию 119

Глава 3. Структуры принятия решения и булева логика **125**
3.1 Инструкция *if* 125
3.2 Инструкция *if-else* 133
3.3 Сравнение строковых значений 137

3.4 Вложенные структуры принятия решения и инструкция *if-elif-else* 141
3.5 Логические операторы 149
3.6 Булевы переменные 156
3.7 Черепашья графика: определение состояния черепахи 157
Вопросы для повторения 165
Задачи по программированию 169

Глава 4. Структуры с повторением 177
4.1 Введение в структуры повторения 177
4.2 Цикл *while*: цикл с условием повторения 178
4.3 Цикл *for*: цикл со счетчиком повторений 185
4.4 Вычисление нарастающего итога 196
4.5 Сигнальные метки 200
4.6 Циклы валидации входных данных 202
4.7 Вложенные циклы 207
4.8 Черепашья графика: применение циклов для рисования узоров 214
Вопросы для повторения 217
Задачи по программированию 220

Глава 5. Функции 225
5.1 Введение в функции 225
5.2 Определение и вызов функции без возврата значения 228
5.3 Проектирование программы с использованием функций 233
5.4 Локальные переменные 239
5.5 Передача аргументов в функцию 241
5.6 Глобальные переменные и глобальные константы 251
5.7 Введение в функции с возвратом значения: генерирование случайных чисел 255
5.8 Написание функций с возвратом значения 265
5.9 Математический модуль *math* 277
5.10 Хранение функций в модулях 280
5.11 Черепашья графика: модуляризация кода при помощи функций 284
Вопросы для повторения 291
Задачи по программированию 296

Глава 6. Файлы и исключения 303
6.1 Введение в файловый ввод и вывод 303
6.2 Применение циклов для обработки файлов 320
6.3 Обработка записей 327
6.4 Исключения 340
Вопросы для повторения 353
Задачи по программированию 357

Глава 7. Списки и кортежи 361
7.1 Последовательности 361
7.2 Введение в списки 361
7.3 Нарезка списка 369
7.4 Поиск значений в списках при помощи инструкции *in* 372
7.5 Списковые методы и полезные встроенные функции 373
7.6 Копирование списков 381
7.7 Обработка списков 382

7.8 Двумерные списки ... 395
7.9 Кортежи ... 399
7.10 Построение графиков с данными списков при помощи пакета *matplotlib* ... 401
Вопросы для повторения ... 418
Задачи по программированию ... 421

Глава 8. Подробнее о строковых данных ... 427
8.1 Базовые строковые операции ... 427
8.2 Нарезка строковых значений ... 434
8.3 Проверка, поиск и манипуляция строковыми данными ... 439
Вопросы для повторения ... 451
Задачи по программированию ... 454

Глава 9. Словари и множества ... 459
9.1 Словари ... 459
9.2 Множества ... 483
9.3 Сериализация объектов ... 496
Вопросы для повторения ... 502
Задачи по программированию ... 507

Глава 10. Классы и объектно-ориентированное программирование ... 513
10.1 Процедурное и объектно-ориентированное программирование ... 513
10.2 Классы ... 516
10.3 Работа с экземплярами ... 534
10.4 Приемы конструирования классов ... 556
Вопросы для повторения ... 567
Задачи по программированию ... 571

Глава 11. Наследование ... 575
11.1 Введение в наследование ... 575
11.2 Полиморфизм ... 589
Вопросы для повторения ... 596
Задачи по программированию ... 598

Глава 12. Рекурсия ... 601
12.1 Введение в рекурсию ... 601
12.2 Решение задач на основе рекурсии ... 604
12.3 Примеры алгоритмов на основе рекурсии ... 607
Вопросы для повторения ... 615
Задачи по программированию ... 618

Глава 13. Программирование графического пользовательского интерфейса ... 619
13.1 Графические интерфейсы пользователя ... 619
13.2 Использование модуля *tkinter* ... 621
13.3 Вывод текста при помощи элемента интерфейса *Label* ... 624
13.4 Упорядочение элементов интерфейса при помощи рамок *Frame* ... 628
13.5 Элементы интерфейса *Button* и информационные диалоговые окна ... 630
13.6 Получение входных данных при помощи элемента интерфейса *Entry* ... 634
13.7 Применение элементов *Label* в качестве полей вывода ... 637
13.8 Радиокнопки и флаговые кнопки ... 644
13.9 Рисование фигур при помощи элемента интерфейса *Canvas* ... 651

Вопросы для повторения ... 671
Задачи по программированию ... 675

Глава 14. Основы функционального программирования ... 681
14.1 Принципы функционального программирования ... 681
14.2 Оператор *lambda*, функции *map*, *filter*, *reduce* и другие ... 682
14.3 Включение в последовательность ... 687
14.4 Замыкание ... 689
14.5 Функциональное ядро программы на основе конвейера ... 691

Приложение 1. Установка языка Python ... 699
Скачивание языка Python ... 699
Установка Python 3.x в Windows ... 699

Приложение 2. Введение в среду IDLE ... 701
Запуск среды IDLE и использование оболочки Python ... 701
Написание программы Python в редакторе IDLE ... 703
Цветная разметка ... 704
Автоматическое выделение отступом ... 704
Сохранение программы ... 705
Выполнение программы ... 705
Другие ресурсы ... 706

Приложение 3. Набор символов ASCII ... 707

Приложение 4. Предопределенные именованные цвета ... 709

Приложение 5. Подробнее об инструкции *import* ... 715
Импортирование конкретной функции или класса ... 715
Импорт с подстановочным символом ... 716
Использование псевдонимов ... 716

Приложение 6. Установка модулей при помощи менеджера пакетов *pip* ... 719

Приложение 7. Ответы на вопросы в *Контрольных точках* ... 721
Глава 1 ... 721
Глава 2 ... 722
Глава 3 ... 724
Глава 4 ... 725
Глава 5 ... 727
Глава 6 ... 729
Глава 7 ... 730
Глава 8 ... 732
Глава 9 ... 733
Глава 10 ... 735
Глава 11 ... 736
Глава 12 ... 736
Глава 13 ... 736

Предметный указатель ... 739

Оглавление

Краткое оглавление ..... **VII**

Предисловие **1**
Прежде всего управляющие структуры и только потом классы 1
Краткий обзор глав 1
Организация учебного материала 4
Структурные элементы и условные обозначения книги 5
Дополнительные материалы 6
Онлайновые учебные ресурсы 6
Ресурсы для преподавателя 6
Электронный архив 7
Об авторе 7
Комментарий переводчика 7
Представление чисел 7
Базовый набор сторонних библиотек 8
Порядок установки библиотек 9
Установка библиотек Python из whl-файлов 9
Установка и настройка инструментальной среды Spyder 10
Блокноты Jupyter 10

Глава 1. Введение в компьютеры и программирование **13**
1.1 Введение 13
1.2 Аппаратное и программное обеспечение 14
Аппаратное обеспечение 14
Центральный процессор 15
Основная память 17
Вторичные устройства хранения 17
Устройства ввода 18
Устройства вывода 18
Программное обеспечение 18
Системное программное обеспечение 18
Прикладное программное обеспечение 19
1.3 Как компьютеры хранят данные 20
Хранение чисел 21
Хранение символов 22
Хранение чисел повышенной сложности 23
Другие типы данных 23

1.4 Как программа работает 24
От машинного языка к языку ассемблера 27
Высокоуровневые языки 28
Ключевые слова, операторы и синтаксис: краткий обзор 29
Компиляторы и интерпретаторы 30
1.5 Использование языка Python 32
Установка языка Python 32
Интерпретатор языка Python 32
Интерактивный режим 33
Написание программ Python и их выполнение в сценарном режиме 34
Среда программирования IDLE 35
Вопросы для повторения 36
Множественный выбор 36
Истина или ложь 40
Короткий ответ 40
Задачи 40

Глава 2. Ввод, обработка и вывод 43
2.1 Проектирование программы 43
Цикл проектирования программы 43
Подробнее о процессе проектирования 44
Понять задачу, которую программа должна выполнить 44
Определить шаги, необходимые для выполнения задачи 45
Псевдокод 46
Блок-схемы 46
2.2 Ввод, обработка и вывод 48
2.3 Вывод данных на экран при помощи функции *print* 48
Строковые данные и строковые литералы 49
2.4 Комментарии 51
2.5 Переменные 52
Создание переменных инструкцией присваивания 53
Правила именования переменных 55
Вывод многочисленных значений при помощи функции *print* 57
Повторное присваивание значений переменным 58
Числовые типы данных и числовые литералы 58
Хранение строковых данных с типом *str* 60
Повторное присвоение переменной значения другого типа 60
2.6 Чтение входных данных с клавиатуры 62
Чтение чисел при помощи функции *input* 63
2.7 Выполнение расчетов 66
В центре внимания Вычисление процентов 68
Деление с плавающей точкой и целочисленное деление 69
Приоритет операторов 70
Группирование при помощи круглых скобок 71
В центре внимания Вычисление среднего арифметического значения 71
Оператор возведения в степень 73
Оператор остатка от деления 73
Преобразование математических формул в программные инструкции 74
В центре внимания Преобразование математической формулы в программную инструкцию 75

Смешанные выражения и преобразование типов данных 77
Разбиение длинных инструкций на несколько строк 78
2.8 Подробнее о выводе данных 79
Подавление концевого символа новой строки в функции *print* 79
Задание символа-разделителя значений 80
Экранированные символы 80
Вывод многочисленных значений при помощи оператора + ...81
Форматирование чисел 82
Форматирование в экспоненциальной записи 83
Вставка запятой в качестве разделителя 84
Определение минимальной ширины поля 84
Форматирование числа с плавающей точкой в виде процента 86
Форматирование целых чисел 86
2.9 Именованные константы 87
2.10 Введение в черепашью графику 88
Рисование отрезков прямой при помощи черепахи 89
Поворот черепахи 91
Установка углового направления черепахи в заданный угол 94
Получение текущего углового направления черепахи 94
Поднятие и опускание пера 95
Рисование кругов и точек 96
Изменение размера пера 97
Изменение цвета рисунка 97
Изменение цвета фона 98
Возвращение экрана в исходное состояние 98
Установление размера графического окна 98
Перемещение черепахи в заданную позицию 99
Получение текущей позиции черепахи 100
Управление скоростью анимации черепахи 101
Сокрытие черепахи 101
Вывод текста в графическое окно 101
Заполнение геометрических фигур 103
Применение команды *turtle.done()* для сохранения графического окна открытым 105
В центре внимания Программа "Созвездие Ориона" 105
Вопросы для повторения 114
Множественный выбор 114
Истина или ложь 117
Короткий ответ 118
Алгоритмический тренажер 118
Задачи по программированию 119

Глава 3. Структуры принятия решения и булева логика 125
3.1 Инструкция *if* 125
Булевы выражения и операторы сравнения 127
Операторы >= и <= 129
Оператор == 129
Оператор != 130
Собираем все вместе 130
В центре внимания Применение инструкции *if* 131

3.2 Инструкция *if-else* 133
Выделение отступом в инструкции *if-else* 135
В центре внимания Применение инструкции *if-else* 135
3.3 Сравнение строковых значений 137
Другие способы сравнения строковых значений 138
3.4 Вложенные структуры принятия решения и инструкция *if-elif-else* 141
Проверка серии условий 145
В центре внимания Многочисленные вложенные структуры принятия решения 145
Инструкция *if-elif-else* 147
3.5 Логические операторы 149
Оператор *and* 150
Оператор *or* 150
Вычисление по укороченной схеме 151
Оператор *not* 151
Пересмотренная программа одобрения на получение ссуды 152
Еще одна программа об одобрении ссуды 153
Проверка числовых диапазонов при помощи логических операторов 154
3.6 Булевы переменные 156
3.7 Черепашья графика: определение состояния черепахи 157
Определение позиции черепахи 157
Определение углового направления черепахи 157
Определение положения пера над холстом 157
Определение видимости черепахи 158
Определение текущего цвета 158
Определение размера пера 159
Определение скорости анимации черепахи 159
В центре внимания Игра "Порази цель" 160
Вопросы для повторения 165
Множественный выбор 165
Истина или ложь 167
Короткий ответ 167
Алгоритмический тренажер 168
Задачи по программированию 169

Глава 4. Структуры с повторением 177
4.1 Введение в структуры повторения 177
Циклы с условием повторения и со счетчиком повторений 178
4.2 Цикл *while*: цикл с условием повторения 178
Цикл *while* как цикл с предусловием 181
В центре внимания Проектирование программы с циклом *while* 183
Бесконечные циклы 184
4.3 Цикл *for*: цикл со счетчиком повторений 185
Применение функции *range* с циклом *for* 188
Использование целевой переменной в цикле 189
В центре внимания Проектирование цикла со счетчиком повторений на основе инструкции *for* 192
Пользовательский контроль итераций цикла 193
Порождение итерируемой последовательности в диапазоне от максимального до минимального значений 195

4.4 Вычисление нарастающего итога 196
Расширенные операторы присваивания 198
4.5 Сигнальные метки 200
В центре внимания Применение сигнальной метки 200
4.6 Циклы валидации входных данных 202
В центре внимания Написание цикла валидации входных данных 204
4.7 Вложенные циклы 207
В центре внимания Применение вложенных циклов для печати комбинаций символов 210
4.8 Черепашья графика: применение циклов для рисования узоров 214
Вопросы для повторения 217
Множественный выбор 217
Истина или ложь 219
Короткий ответ 219
Алгоритмический тренажер 220
Задачи по программированию 220

Глава 5. Функции 225
5.1 Введение в функции 225
Преимущества модуляризации программы на основе функций 226
Функции без возврата значения и с возвратом значения 227
5.2 Определение и вызов функции без возврата значения 228
Определение и вызов функции 228
Вызов функции 229
Выделение отступом в Python 232
5.3 Проектирование программы с использованием функций 233
Составление блок-схемы программы с использованием функций 234
Нисходящая разработка алгоритма 234
Иерархические схемы 235
В центре внимания Определение и вызов функций 236
Приостановка исполнения до тех пор, пока пользователь не нажмет клавишу <Enter> 238
5.4 Локальные переменные 239
Область действия и локальные переменные 240
5.5 Передача аргументов в функцию 241
Область действия параметрической переменной 243
В центре внимания Передача аргумента в функцию 243
Передача многочисленных аргументов 245
Внесение изменений в параметры 247
Именованные аргументы 248
Смешивание именованных и позиционных аргументов 250
5.6 Глобальные переменные и глобальные константы 251
Глобальные константы 252
В центре внимания Применение глобальных констант 253
5.7 Введение в функции с возвратом значения: генерирование случайных чисел 255
Функции стандартной библиотеки и инструкция *import* 255
Генерирование случайных чисел 256
Эксперименты с использованием случайных чисел в интерактивном режиме 260
В центре внимания Применение случайных чисел 260
В центре внимания Применение случайных чисел для представления других значений 262
Функции *randrange*, *random* и *uniform* 263
Начальные значения случайного числа 264

5.8 Написание функций с возвратом значения 265
Использование инструкции *return* по максимуму 267
Как использовать функции с возвратом значения? 268
Использование таблиц "ввод-обработка-вывод" 270
В центре внимания Модуляризация функций 270
Возвращение строковых значений 274
Возвращение булевых значений 274
Использование булевых функций в программном коде валидации входных данных 275
Возвращение многочисленных значений 276
5.9 Математический модуль *math* 277
Значения *math.pi* и *math.e* 279
5.10 Хранение функций в модулях 280
Программы, управляемые из меню 284
5.11 Черепашья графика: модуляризация кода при помощи функций 284
Хранение собственных графических функций в модуле 289
Вопросы для повторения 291
Множественный выбор 291
Истина или ложь 294
Короткий ответ 294
Алгоритмический тренажер 295
Задачи по программированию 296

Глава 6. Файлы и исключения 303
6.1 Введение в файловый ввод и вывод 303
Типы файлов 305
Методы доступа к файлам 305
Имена файлов и файловые объекты 305
Открытие файла 306
Указание места расположения файла 307
Запись данных в файл 308
Чтение данных из файла 310
Конкатенация символа новой строки в строковое значение 313
Чтение строкового значения и удаление из него символа новой строки 314
Дозапись данных в существующий файл 315
Запись и чтение числовых данных 316
6.2 Применение циклов для обработки файлов 320
Чтение файла в цикле и обнаружение конца файла 321
Применение цикла *for* для чтения строк 323
В центре внимания Работа с файлами 324
6.3 Обработка записей 327
В центре внимания Добавление и вывод записей на экран 331
В центре внимания Поиск записи 333
В центре внимания Изменение записей 334
В центре внимания Удаление записей 337
6.4 Исключения 340
Обработка многочисленных исключений 346
Использование одного выражения *except* для отлавливания всех исключений 348
Вывод заданного по умолчанию сообщения об ошибке при возникновении исключения 349
Выражение *else* 350

Выражение *finally* 352
Что если исключение не обработано? 352
Вопросы для повторения 353
Множественный выбор 353
Истина или ложь 355
Короткий ответ 356
Алгоритмический тренажер 356
Задачи по программированию 357

Глава 7. Списки и кортежи 361
7.1 Последовательности 361
7.2 Введение в списки 361
Оператор повторения 363
Последовательный обход списка в цикле *for* 364
Индексация 364
Функция *len* 365
Списки — это мутирующие последовательности 365
Конкатенация списков 367
7.3 Нарезка списка 369
7.4 Поиск значений в списках при помощи инструкции *in* 372
7.5 Списковые методы и полезные встроенные функции 373
Метод *append()* 374
Метод *index()* 376
Метод *insert()* 377
Метод *sort()* 378
Метод *remove()* 378
Метод *reverse()* 379
Инструкция *del* 379
Функции *min* и *max* 380
7.6 Копирование списков 381
7.7 Обработка списков 382
В центре внимания Применение элементов списка в математическом выражении 382
Суммирование значений в списке 384
Усреднение значений в списке 385
Передача списка в функцию в качестве аргумента 386
Возвращение списка из функции 386
В центре внимания Обработка списка 388
Работа со списками и файлами 391
7.8 Двумерные списки 395
7.9 Кортежи 399
В чем их смысл? 400
Преобразование между списками и кортежами 400
7.10 Построение графиков с данными списков при помощи пакета *matplotlib* 401
Импорт модуля *pyplot* 402
Построение линейного графика 402
Добавление заголовка, меток осей и сетки 404
Индивидуализация настроек осей *x* и *y* 405
Вывод маркеров в точках данных 408
Построение столбчатой диаграммы 410

Индивидуальная настройка ширины столбика 412
Изменение цвета столбиков 413
Добавление заголовка, меток осей и индивидуальная настройка подписей меток делений 413
Построение круговой диаграммы 415
Вывод меток долей и заголовка диаграммы 416
Изменение цвета долей 417
Вопросы для повторения 418
Множественный выбор 418
Истина или ложь 420
Короткий ответ 420
Алгоритмический тренажер 421
Задачи по программированию 421

Глава 8. Подробнее о строковых данных 427
8.1 Базовые строковые операции 427
Доступ к отдельным символам в строковом значении 427
Обход строкового значения в цикле *for* 428
Индексация 430
Исключения *IndexError* 431
Функция *len* 431
Конкатенация строковых данных 432
Строковые данные — это немутирующие последовательности 433
8.2 Нарезка строковых значений 434
В центре внимания Извлечение символов из строкового значения 436
8.3 Проверка, поиск и манипуляция строковыми данными 439
Проверка строковых значений при помощи *in* и *not in* 439
Строковые методы 439
Методы проверки строковых значений 440
Методы модификации 442
Поиск и замена 443
В центре внимания Анализ символов в пароле 445
Оператор повторения 448
Разбиение строкового значения 449
Вопросы для повторения 451
Множественный выбор 451
Истина или ложь 453
Короткий ответ 453
Алгоритмический тренажер 454
Задачи по программированию 454

Глава 9. Словари и множества 459
9.1 Словари 459
Создание словаря 460
Получение значения из словаря 460
Применение операторов *in* и *not in* для проверки на наличие значения в словаре 461
Добавление элементов в существующий словарь 462
Удаление элементов 463
Получение количества элементов в словаре 464
Смешивание типов данных в словаре 464

Создание пустого словаря 466
Применение цикла *for* для последовательного обхода словаря 466
Несколько словарных методов 467
Метод *clear()* 467
Метод *get()* 468
Метод *items()* 468
Метод *keys()* 469
Метод *pop()* 470
Метод *popitem()* 471
Метод *values()* 472
В центре внимания Применение словаря для имитации карточной колоды 472
В центре внимания Хранение имен и дней рождения в словаре 475
9.2 Множества 483
Создание множества 483
Получение количества элементов в множестве 484
Добавление и удаление элементов 484
Применение цикла *for* для последовательного обхода множества 486
Применение операторов *in* и *not in* для проверки на принадлежность значения множеству 487
Объединение множеств 488
Пересечение множеств 488
Разность множеств 489
Симметрическая разность множеств 490
Подмножества и надмножества 490
В центре внимания Операции над множествами 492
9.3 Сериализация объектов 496
Вопросы для повторения 502
Множественный выбор 502
Истина или ложь 504
Короткий ответ 505
Алгоритмический тренажер 506
Задачи по программированию 507

Глава 10. Классы и объектно-ориентированное программирование 513
10.1 Процедурное и объектно-ориентированное программирование 513
Возможность многократного использования объекта 514
Пример объекта из повседневной жизни 515
10.2 Классы 516
Определения классов 518
Сокрытие атрибутов 523
Хранение классов в модулях 526
Класс *BankAccount* 528
Метод *__str__()* 531
10.3 Работа с экземплярами 534
В центре внимания Создание класса *CellPhone* 536
Методы-получатели и методы-модификаторы 538
В центре внимания Хранение объектов в списке 539
Передача объектов в качестве аргументов 542
В центре внимания Консервация собственных объектов 543
В центре внимания Хранение объектов в словаре 546

10.4 Приемы конструирования классов 556
Унифицированный язык моделирования 556
Идентификация классов в задаче 557
Составление описания предметной области задачи 557
Идентификация всех именных групп 558
Уточнение списка именных групп 559
Идентификация обязанностей класса 562
Класс *Customer* 563
Класс *Car* 564
Класс *ServiceQuote* 566
Это только начало 567
Вопросы для повторения 567
Множественный выбор 567
Истина или ложь 569
Короткий ответ 570
Алгоритмический тренажер 570
Задачи по программированию 571

Глава 11. Наследование 575
11.1 Введение в наследование 575
Обобщение и конкретизация 575
Наследование и отношение "род — вид" 575
Наследование в диаграммах UML 584
В центре внимания Применение наследования 585
11.2 Полиморфизм 589
Функция *isinstance* 592
Вопросы для повторения 596
Множественный выбор 596
Истина или ложь 597
Короткий ответ 597
Алгоритмический тренажер 597
Задачи по программированию 598

Глава 12. Рекурсия 601
12.1 Введение в рекурсию 601
12.2 Решение задач на основе рекурсии 604
Применение рекурсии для вычисления факториала числа 604
Прямая и косвенная рекурсия 607
12.3 Примеры алгоритмов на основе рекурсии 607
Последовательность Фибоначчи 608
Нахождение наибольшего общего делителя 610
Ханойские башни 611
Рекурсия против циклов 614
Вопросы для повторения 615
Множественный выбор 615
Истина или ложь 616
Короткий ответ 617
Алгоритмический тренажер 617
Задачи по программированию 618

Глава 13. Программирование графического пользовательского интерфейса 619
13.1 Графические интерфейсы пользователя 619
Программы с GUI событийно-управляемы 620
13.2 Использование модуля *tkinter* 621
13.3 Вывод текста при помощи элемента интерфейса *Label* 624
13.4 Упорядочение элементов интерфейса при помощи рамок *Frame* 628
13.5 Элементы интерфейса *Button* и информационные диалоговые окна 630
Создание кнопки выхода из программы 632
13.6 Получение входных данных при помощи элемента интерфейса *Entry* 634
13.7 Применение элементов *Label* в качестве полей вывода 637
В центре внимания Создание программы с GUI 640
13.8 Радиокнопки и флаговые кнопки 644
Радиокнопки 644
Использование функций обратного вызова с радиокнопками 647
Флаговые кнопки 648
13.9 Рисование фигур при помощи элемента интерфейса *Canvas* 651
Экранная система координат элемента интерфейса *Canvas* 651
Рисование прямых: метод *create_line()* 653
Рисование прямоугольников: метод *create_rectangle()* 655
Рисование овалов: метод *create_oval()* 657
Рисование дуг: метод *create_arc()* 659
Рисование многоугольников: метод *create_polygon()* 664
Рисование текста: метод *create_text()* 666
Настройка шрифта 669
Вопросы для повторения 671
Множественный выбор 671
Истина или ложь 674
Короткий ответ 674
Алгоритмический тренажер 674
Задачи по программированию 675

Глава 14. Основы функционального программирования 681
14.1 Принципы функционального программирования 681
14.2 Оператор *lambda*, функции *map*, *filter*, *reduce* и другие 682
Оператор *lambda* 682
Функция *map* 683
Функция *filter* 684
Функция *reduce* 685
Функция *zip* 686
Функция *enumerate* 687
14.3 Включение в последовательность 687
14.4 Замыкание 689
14.5 Функциональное ядро программы на основе конвейера 691
В центре внимания Функциональная реализация вычисления факториала числа 693
В центре внимания Функциональная реализация вычисления последовательности Фибоначчи 696
В центре внимания Функциональная реализация суммирования диапазона значений последовательности 698

Приложение 1. Установка языка Python 699
Скачивание языка Python 699
Установка Python 3.x в Windows 699

Приложение 2. Введение в среду IDLE 701
Запуск среды IDLE и использование оболочки Python 701
Написание программы Python в редакторе IDLE 703
Цветная разметка 704
Автоматическое выделение отступом 704
Сохранение программы 705
Выполнение программы 705
Другие ресурсы 706

Приложение 3. Набор символов ASCII 707

Приложение 4. Предопределенные именованные цвета 709

Приложение 5. Подробнее об инструкции *import* 715
Импортирование конкретной функции или класса 715
Импорт с подстановочным символом 716
Использование псевдонимов 716

Приложение 6. Установка модулей при помощи менеджера пакетов *pip* 719

Приложение 7. Ответы на вопросы в *Контрольных точках* 721
Глава 1 721
Глава 2 722
Глава 3 724
Глава 4 725
Глава 5 727
Глава 6 729
Глава 7 730
Глава 8 732
Глава 9 733
Глава 10 735
Глава 11 736
Глава 12 736
Глава 13 736

Предметный указатель 739

Предисловие

Добро пожаловать в четвертое издание книги "Все начинается с Python". В книге изложены принципы программирования, с помощью которых вы приобретете навыки алгоритмического решения задач на языке Python, даже если у вас нет опыта программирования. На доступных для понимания примерах, псевдокоде, блок-схемах и других инструментах вы научитесь разрабатывать алгоритмы и реализовывать их в программах. Книга идеально подходит для вводного курса по программированию или курса логики и разработки программного обеспечения на основе языка Python.

Отличительным признаком издания является его ясное, дружественное и легкое для понимания изложение материала. Помимо этого, оно располагает большим количеством сжатых и практичных примеров программ. Программы, в свою очередь, содержат краткие примеры, посвященные конкретным темам программирования, а также более развернутые примеры, направленные на решение задачи. В каждой главе предложено одно или несколько тематических исследований, которые обеспечивают пошаговый анализ конкретной задачи и демонстрируют обучаемому ее решение.

Прежде всего управляющие структуры и только потом классы

Python — это полностью объектно-ориентированный язык программирования, однако, чтобы начать программировать на Python, обучаемому не обязательно разбираться в понятиях. Книга сначала знакомит с основными принципами хранения данных, консольного и файлового ввода-вывода, управляющими структурами, функциями, последовательностями и списками, а также объектами, которые создаются из классов стандартной библиотеки. Затем обучаемый учится программировать классы, изучает темы наследования и полиморфизма и учится писать рекурсивные функции. И наконец, он учится разрабатывать простые событийно-управляемые приложения с графическим пользовательским интерфейсом (graphical user interface, GUI).

Краткий обзор глав

Глава 1. Введение в компьютеры и программирование

Глава начинается с объяснения в очень конкретной и легкой для понимания форме принципов работы компьютеров, способов хранения и обработки данных, а также причин написания программ на высокоуровневых языках. В ней предлагается введение в язык Python, интерактивный и сценарный режимы работы и среду разработки IDLE.

Глава 2. Ввод, обработка и вывод

Глава знакомит с циклом разработки программ, переменными, типами данных и простыми программами, которые пишутся как последовательности структур. Обучаемый учится писать простые программы, способные считывать данные с клавиатуры, выполнять математические операции и выводить результаты на экран. Кроме того, он знакомится с такими инструментами разработки программ, как псевдокод и блок-схемы. В этой главе также представлен дополнительный материал о библиотеке черепашьей графики.

Глава 3. Структуры принятия решения и булева логика

В данной главе обучаемый знакомится с операторами сравнения и булевыми выражениями и учится управлять потоком выполнения программы при помощи управляющих структур. Рассматриваются инструкции `if`, `if-else` и `if-elif-else`, обсуждаются вложенные управляющие структуры и логические операторы. Данная глава также включает дополнительный материал, посвященный черепашьей графике с обсуждением приемов применения управляющих структур для проверки состояния черепахи.

Глава 4. Структуры с повторением

В главе демонстрируются способы создания структур повторения на основе циклов `while` и `for`. Рассматриваются счетчики, накопители, нарастающие итоги и сигнальные метки, а также приемы написания циклов проверки допустимости вводимых данных. Глава содержит дополнительный материал, посвященный применению циклов для рисования узоров с использованием библиотеки черепашьей графики.

Глава 5. Функции

В данной главе обучаемый сначала учится писать и вызывать функции без возврата значения — так называемые пустые функции. Показаны преимущества использования функций в плане модуляризации программ, и обсуждается нисходящий подход к их разработке. Затем обучаемый учится передавать в функции аргументы. Рассматриваются общие библиотечные функции, в частности, функции генерации случайных чисел. Научившись вызывать библиотечные функции и использовать возвращаемые ими значения, обучаемый учится писать и вызывать собственные функции и применять модули для упорядочения функций. В качестве дополнительного материала обсуждаются темы модуляризации программного кода с инструкциями черепашьей графики на основе функций.

Глава 6. Файлы и исключения

Глава знакомит с последовательным файловым вводом и выводом. Читатель учится считывать и записывать большие наборы данных и хранить данные в виде полей и записей. Глава заканчивается рассмотрением темы исключений и знакомит с приемами написания программного кода для обработки исключений.

Глава 7. Списки и кортежи

В данной главе вводится понятие последовательности в языке Python и рассматривается применение двух часто встречающихся последовательностей Python: списков и кортежей. Обучаемый учится применять списки для массивоподобных операций, таких как хранение объектов в списке, последовательный обход списка, поиск значений в списке и вычисление суммы и среднего арифметического значения в списке. В этой главе обсуждается тема на-

резки списка и многие другие списковые методы. Рассматриваются одно- и двумерные списки. Представлен анализ пакета `matplotlib` и приемов его применения для построения диаграмм и графиков из списков.

Глава 8. Подробнее о строковых данных

В данной главе читатель учится обрабатывать строковые данные в развернутом виде. Обсуждаются нарезка строковых данных и алгоритмы, которые выполняют последовательный перебор отдельных символов в строковом значении. Глава также знакомит с несколькими встроенными функциями и строковыми методами обработки символов и текста.

Глава 9. Словари и множества

Данная глава знакомит с такими структурами данных, как словарь и множество. Читатель учится хранить данные в словарях в виде пар "ключ-значение", отыскивать значения, изменять существующие значения, добавлять новые пары "ключ-значение" и удалять такие пары. Объясняется, как хранить значения в множествах в виде уникальных элементов и выполнять широко применяемые операции над множествами, такие как объединение, пересечение, разность и симметрическая разность. Данная глава заканчивается обсуждением темы сериализации объектов и знакомит читателя с Python-модулем `pickle` для консервации данных.

Глава 10. Классы и объектно-ориентированное программирование

В данной главе сравниваются методы процедурного и объектно-ориентированного программирования. В ней раскрываются фундаментальные понятия классов и объектов. Обсуждается тема атрибутов, методов, инкапсуляции и сокрытия данных, а также рассматриваются функции инициализации `__init__` (аналогичные конструкторам), методы-получатели и методы-модификаторы. Читатель учится моделировать классы при помощи унифицированного языка моделирования (UML) и идентифицировать классы в конкретной задаче.

Глава 11. Наследование

В данной главе изучение классов продолжится, и на этот раз темами изучения станут понятия наследования и полиморфизма. Будут затронуты темы надклассов, подклассов, роли функции `__init__` в наследовании, а также темы переопределения методов и полиморфизма.

Глава 12. Рекурсия

В данной главе рассматриваются рекурсия и ее применение в решении задач. Будет представлена визуальная трассировка рекурсивных вызовов и рассмотрены рекурсивные приложения. Здесь показаны рекурсивные алгоритмы для ряда задач, таких как нахождение факториалов, нахождение наибольшего общего знаменателя, суммирование диапазона значений в списке и классический пример головоломки "Ханойские башни".

Глава 13. Программирование графического пользовательского интерфейса

В данной главе рассматриваются основные аспекты разработки приложения с графическим интерфейсом пользователя (GUI) на языке Python с применением модуля `tkinter`. Обсуждаются фундаментальные элементы визуального интерфейса, такие как метки, кнопки, поля ввода данных, переключатели, флаговые кнопки и диалоговые окна. Обучаемый также узнает, как в приложении с GUI работают события и как для обработки событий писать функции

обратного вызова. В главе обсуждаются элемент интерфейса `Canvas` и приемы его использования для рисования линий, прямоугольников, овалов, дуг, многоугольников и текста.

Глава 14. Основы функционального программирования

Эта глава добавлена в русское издание настоящей книги. В ней дается краткое изложение материала о встроенных функциях и расширенных возможностях языка Python по работе с последовательностями. На основе простых примеров рассматриваются возможности применения в Python парадигмы функционального программирования.

Приложение 1. Установка языка Python

В приложении даются разъяснения по поводу скачивания и установки интерпретатора Python 3.

Приложение 2. Введение в среду IDLE

Приводится обзор интегрированной среды разработки IDLE, которая поставляется вместе с Python.

Приложение 3. Набор символов ASCII

В качестве справочного материала приводится набор символов ASCII.

Приложение 4. Предопределенные именованные цвета

Представлен перечень предопределенных названий цветов, которые могут использоваться вместе с библиотекой черепашьей графики, пакетами `matplotlib` и `tkinter`.

Приложение 5. Подробнее об инструкции `import`

Рассматриваются различные способы применения инструкции импорта `import`. Например, эту инструкцию можно использовать для импортирования модуля, класса, функции либо присвоения модулю псевдонима.

Приложение 6. Установка модулей при помощи менеджера пакетов `pip`

Обсуждаются способы применения менеджера пакетов `pip` для установки сторонних модулей из каталога пакетов Python, или PyPI.

Приложение 7. Ответы на вопросы в *Контрольнах точках*

Представлены ответы на вопросы из разделов "Контрольная точка", которые встречаются в конце почти каждого раздела книги.

Организация учебного материала

Представленный в книге тренировочный материал учит программировать в пошаговом режиме. Каждая глава освещает перечисленные ранее в кратком обзоре темы и наращивает знание по мере продвижения от темы к теме. Главы учебника можно изучать в существующей последовательности. Вместе с тем имеется определенная гибкость относительно порядка следования учебного материала. На рис. 1 показана зависимость между главами.

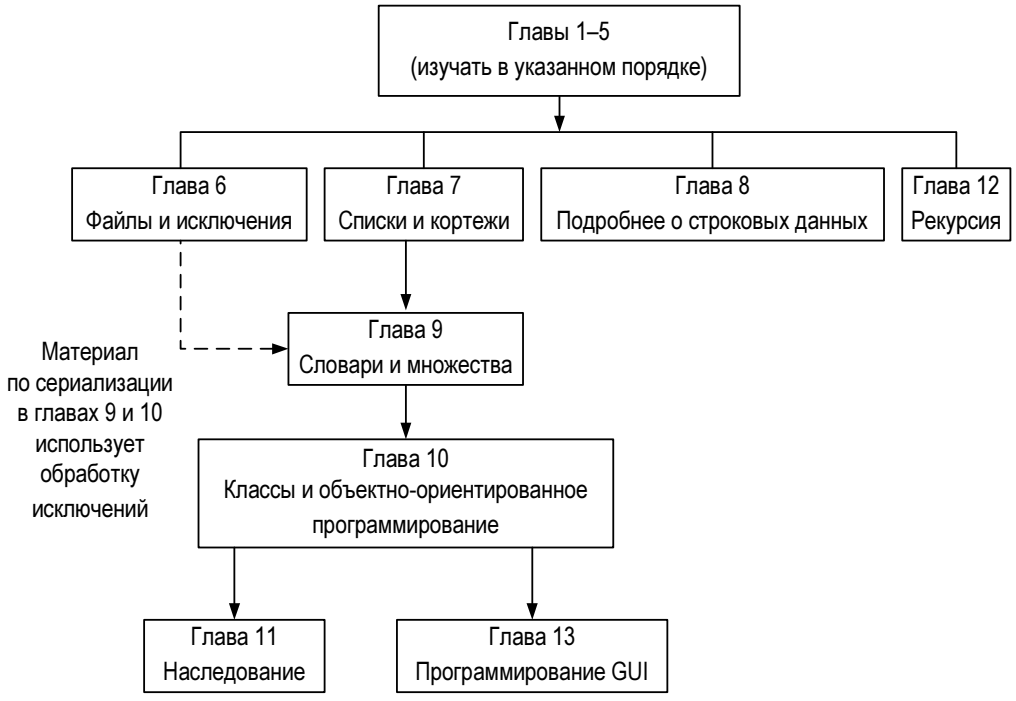


Рис. 1. Зависимости между главами

Структурные элементы и условные обозначения книги

Ключевые положения — Все главные разделы книги начинаются с объяснения принципов и понятий.

Объяснения — Данное объяснение в сжатом виде обобщает главную мысль раздела.

Примеры программ — Главы содержат многочисленные примеры законченных программ и фрагменты кода — все они предназначены для демонстрации текущей темы.

Тематическое исследование "В центре внимания" — Каждая глава содержит одно или несколько тематических исследований под рубрикой "В центре внимания", в которых обучаемому предоставляется подробный пошаговый анализ и демонстрация решения задач.

Видеозаписи — Онлайновые видеоматериалы, разработанные специально для настоящей книги, доступны для просмотра на **www.pearsonhighered.com/cs-resources**[1]. Соответствующие значки появляются на протяжении всего учебника, напоминая обучаемому о видеоматериалах по конкретным темам.

[1] Видеоматериалы доступны после регистрации пользователя по указанному адресу. В раскрывающемся списке **Browse by Author** надо выбрать **Gaddis**, а затем на появившейся веб-странице найти издание **Starting Out with Python, 4th Edition** и щелкнуть по его обложке. На странице издания следует найти горизонтальное меню, щелкнуть по пункту **Resources** и таким образом получить доступ к ресурсам книги: презентациям, программному коду и др. — *Прим. ред.*

Примечания Примечания представляют собой короткие объяснения интересных или нередко недопонимаемых вопросов, относящихся к рассматриваемой теме.

Советы Советы предоставляют обучаемому консультации относительно оптимальных подходов к решению различных задач по программированию.

Предупреждения Предупреждения предостерегают обучаемого относительно приемов или методов программирования, которые могут привести к неправильному функционированию программ либо потере данных.

Контрольная точка Контрольные точки представляют собой перечень промежуточных вопросов, размещенных в конце разделов книги. Они предназначены для быстрой проверки знаний после изучения новой темы.

Вопросы для повторения Каждая глава содержит глубокий и разнообразный перечень вопросов и упражнений для повторения пройденного материала. Здесь представлены: тест с множественным выбором, тест с определением истинного или ложного утверждения, алгоритмический тренажер и вопросы, требующие короткого ответа.

Задачи по программированию Каждая глава предлагает широкий круг задач по программированию, предназначенных для закрепления знаний, полученных при изучении текущей темы.

Дополнительные материалы

Онлайновые учебные ресурсы

Специально для этой книги издателем предоставлено большое количество учебных ресурсов. В частности, на **www.pearsonhighered.com/cs-resources** можно найти следующие материалы:

- исходный код для каждого приводимого в книге примера программы;
- доступ к сопроводительным видеоматериалам.

Ресурсы для преподавателя

Приведенные ниже дополнительные материалы доступны только квалифицированным преподавателям:

- ответы на все вопросы в разделах "Вопросы для повторения";
- решения к упражнениям;
- слайды презентаций PowerPoint для каждой главы;
- тестовый банк.

По поводу получения информации о доступе к вышеназванным ресурсам посетите Образовательный преподавательский ресурсный центр Pearson (**www.pearsonhighered.com/irc**) или свяжитесь со своим местным представителем Образовательного центра Pearson.

Электронный архив

По ссылке **ftp://ftp.bhv.ru/9785977540025.zip** можно скачать электронный архив с рассмотренными проектами и исходными кодами примеров, ответами на вопросы для повторения, а также решениями задач по программированию. Эта ссылка доступна также со страницы книги на сайте **www.bhv.ru**.

Об авторе

Тони Гэддис — ведущий автор серии книг "Начинаем программировать..." (Starting Out With). У него почти двадцатилетний опыт преподавания курсов информатики в колледже округа Хейвуд, шт. Северная Каролина. Его преподавательские заслуги общепризнаны: ему присвоено звание "Преподаватель года" колледжем Северной Каролины, он удостаивался премии "Педагогическое мастерство", вручаемой Национальным институтом кадрового и организационного развития. В серии "Начинаем программировать..." издаются книги, посвященные языкам программирования C++, Java™, Microsoft® Visual Basic®, Microsoft® C#®, Python®, Alice, среде визуальной разработки android-приложений App Inventor, а также программированию логики и дизайна. Все они были опубликованы в издательстве Pearson. Дополнительную информацию можно найти на сайте **www.pearsonhighered.com/gaddisbooks**.

Комментарий переводчика

Весь материал книги протестирован в среде Windows 10. При тестировании исходного кода за основу взят Python версии 3.6.4 и дополнен материалом по функциональному программированию (*см. главу 14*).

Прилагаемый к книге адаптированный и скорректированный исходный код примеров следует разместить в подпапке домашней папки пользователя (/home/python_projects или C:\Users\[ИМЯ_ПОЛЬЗОВАТЕЛЯ]\python_projects).

Представление чисел

Для представления десятичных чисел в данной книге в качестве разделителя используется точка.

Для форматирования чисел в локальное для России представление (когда в качестве разделителя разрядов используется запятая) в Python имеется встроенная библиотека `locale`, которая позволяет настраивать форму представления чисел и соответствующим образом их форматировать. В общем случае для форматирования чисел следует начинать программу с приведенных ниже строк кода:

```
import locale            # импортировать библиотеку по работе с локалями
loc = locale.getlocale()                      # запомнить текущую локаль
locale.setlocale(locale.LC_ALL, "Russian_Russia.1251") # изменить локаль
```

В последней строке задается форма написания чисел, принятая в России. Далее можно разместить собственный программный код, к примеру:

```
salary = 50000.0          # зарплата в рублях
print('Ваша ежемесячная зарплата составляет ',
       locale.format('%.2f', salary, grouping=True), sep='')
```

А в конце программы нужно вернуть локаль:

```
locale.setlocale(locale.LC_ALL, loc)  # вернуть локаль
```

Если выполнить этот фрагмент кода, то получится результат, в котором представление чисел будет соответствовать принятому у нас стандарту, т. е. с разделением групп разрядов пробелом и использованием в качестве десятичного разделителя запятой:

```
Ваша ежемесячная зарплата составляет 50 000,00
```

Базовый набор сторонних библиотек

Хотя в книге установка и применение сторонних библиотек не рассматриваются, тем не менее, ниже приведена информация о базовом наборе инструментов, необходимых для дальнейшей работы. Эта информация ни к чему не обязывает, но избавит от затруднений и послужит прекрасной отправной точкой для всех, кто интересуется программированием на Python.

В обычных условиях библиотеки Python можно скачать и установить из каталога библиотек Python `PyPi` (**https://pypi.python.org/**). Следует учесть, что для работы некоторых библиотек, в частности SciPy и Scikit-learn, в ОС Windows требуется установленная библиотека Numpy+MKL. Библиотека Numpy+MKL "привязана" к библиотеке Intel® Math Kernel Library и содержит необходимые динамические библиотеки (dll-файлы) в каталоге numpy.core. Библиотеку Numpy+MKL следует скачать из репозитория whl-файлов (**http://www.lfd.uci.edu/~gohlke/pythonlibs/**) и установить при помощи менеджера пакетов `pip` как `whl` (соответствующая процедура установки пакетов в формате WHL описана далее). Например, для 64-разрядной операционной системы Windows и среды Python 3.6 команда будет такой:

```
pip install numpy-1.13.4+mkl-cp36-cp36m-win_amd64.whl
```

Основополагающими являются следующие библиотеки:

- NumPy — основополагающая библиотека, необходимая для научных вычислений на Python;
- Matplotlib — библиотека для работы с двумерными графиками;
- Pandas — инструмент для анализа структурных данных и временных рядов. Требует наличия `numpy` и некоторых других;
- Scikit-learn — интегратор классических алгоритмов машинного обучения. Требует наличия `numpy+mkl`;
- Scikit-image — коллекция алгоритмов для обработки изображений. Требует наличия `numpy+mkl`;
- SciPy — библиотека, используемая в математике, естественных науках и инженерном деле; требует наличия `numpy+mkl`;
- Jupyter — интерактивная онлайновая вычислительная среда;
- PyQt5 — библиотека инструментов для программирования графического интерфейса пользователя, требуется для работы инструментальной среды программирования Spyder;
- Spyder — инструментальная среда программирования на Python.

Порядок установки библиотек

Для того чтобы работать с более свежими версиями библиотек и держать процесс установки под своим контролем, далее предлагается список команд установки библиотек. В случае нескольких библиотек может потребоваться установка из файлов whl, которые можно взять из репозитория whl-файлов (**http://www.lfd.uci.edu/~gohlke/pythonlibs/**).

```
pip3 install --upgrade pip
pip3 install numpy
```

Либо как whl-файл: `pip3 install numpy-1.13.4+mkl-cp36-cp36m-win_amd64.whl`

```
pip3 install pandas
pip3 install scipy
```

Либо как whl-файл: `pip3 install scipy-0.19.0-cp36-cp36m-win_amd64.whl`

```
pip3 install Scikit-learn
```

Либо как whl-файл: `pip3 install scikit_learn-0.18.1-cp36-cp36m-win_amd64.whl`

```
pip install scikit-image
```

Либо как whl-файл: `pip install scikit_image-0.13.1-cp36-cp36m-win_amd64.whl`

```
pip3 install matplotlib
pip install jupyter
pip install pyqt5
pip install spyder
```

ПРИМЕЧАНИЕ

В зависимости от базовой ОС, версий языка Python и версий программных библиотек устанавливаемые вами версии whl-файлов могут отличаться от приведенных выше, где показаны последние на январь 2018 г. версии для 64-разрядной ОС Windows и Python версии 3.6.4.

Установка библиотек Python из whl-файлов

Библиотеки для Python можно разрабатывать не только на чистом Python. Довольно часто библиотеки программируют на C (динамические библиотеки), и для них пишут обертку Python, либо библиотеку пишут на Python, но для оптимизации узких мест часть кода — на C. Такие библиотеки получаются очень быстрыми, однако библиотеки с вкраплениями кода на C программисту на Python тяжелее установить ввиду отсутствия соответствующих знаний либо необходимых компонентов и настроек в рабочей среде (в особенности в Windows). Для решения описанных проблем разработан специальный формат (файлы с расширением whl) для распространения библиотек, который содержит заранее скомпилированную версию библиотеки со всеми ее зависимостями. Формат WHL поддерживается всеми основными платформами (Mac OS X, Linux, Windows).

Установка производится с помощью менеджера библиотек `pip`. В отличие от обычной установки командой `pip install <имя_библиотеки>` вместо имени библиотеки указывается полное имя whl-файла (то есть путь к файлу и собственно имя файла):

```
pip install <путь_к_whl-файлу>
```

Например,

```
pip install C:\temp\scipy-0.19.0-cp36-cp36m-win_amd64.whl
```

Откройте окно командой строки и при помощи команды `cd` перейдите в каталог, где размещен ваш whl-файл. В этом случае путь указывать не понадобится. Например,

```
pip install scipy-0.19.0-cp36-cp36m-win_amd64.whl
```

При выборе библиотеки важно, чтобы разрядность устанавливаемой библиотеки и разрядность интерпретатора совпадали. Пользователи Windows могут брать whl-файлы на вебстранице **http://www.lfd.uci.edu/~gohlke/pythonlibs/** Кристофа Голька из Лаборатории динамики флуоресценции Калифорнийского университета в г. Ирвайн. Библиотеки там постоянно обновляются, и в архиве содержатся все, какие только могут понадобиться.

Установка и настройка инструментальной среды Spyder

Spyder (Scientific PYthon Development EnviRonment) — это инструментальная среда для научных вычислений для языка Python для Windows, Mac OS X и Linux. Это простая, легковесная и бесплатная интерактивная среда разработки на Python, которая предлагает функционал, аналогичный среде разработки на MATLAB, включая готовые к использованию элементы интерфейса PyQt5 и PySide: редактор исходного кода, редактор массивов данных `NumPy`, редактор словарей, а также консоли Python и IPython и многое другое.

Для того чтобы установить среду Spyder в Ubuntu Linux, используя официальный менеджер библиотек, нужна всего одна команда:

```
sudo apt-get install spyder
```

Установка с использованием менеджера библиотек `pip` выполняется так:

```
sudo apt-get install python-qt5 python-sphinx
sudo pip install spyder
```

Обновление:

```
sudo pip install -U spyder
```

Установка среды Spyder в Fedora 27:

```
dnf install python-spyder
```

Установка среды Spyder в Windows:

```
pip install spyder
```

ПРИМЕЧАНИЕ

Среда Spyder требует обязательной установки библиотеки PyQt5 (`pip install pyqt5`).

Блокноты Jupyter

Среди файлов с исходным кодом Python можно найти файлы с расширением html и ipynb. Последние — это файлы блокнотов интерактивной среды программирования Jupyter (**http://jupyter.org/**). Блокноты Jupyter позволяют иметь в одном месте исходный код, результаты выполнения исходного кода, графики данных и документацию, которая поддерживает синтаксис упрощенной разметки Markdown и мощный синтаксис LaTeX.

Интерактивная среда программирования Jupyter — это зонтичный проект, который наряду с Python предназначен для выполнения в обычном веб-браузере небольших программ

и фрагментов программного кода на других языках программирования, в том числе Julia, R и многих других (уже более 40 языков).

Интерактивная среда программирования Jupyter устанавливается, как обычно, при помощи менеджера пакетов `pip`:

```
pip install jupyter
```

Для того чтобы запустить интерактивную среду Jupyter, нужно в командой оболочке или окне терминала набрать и исполнить команду:

```
jupyter notebook
```

Локальный сервер интерактивной среды Jupyter запустится в браузере, заданном по умолчанию (как правило, по адресу **http://localhost:8888/**). После этого нужный блокнот Jupyter можно просто выбрать из меню Jupyter.

Файлы HTML представляют собой простые копии блокнотов Jupyter для просмотра в веб-браузере.

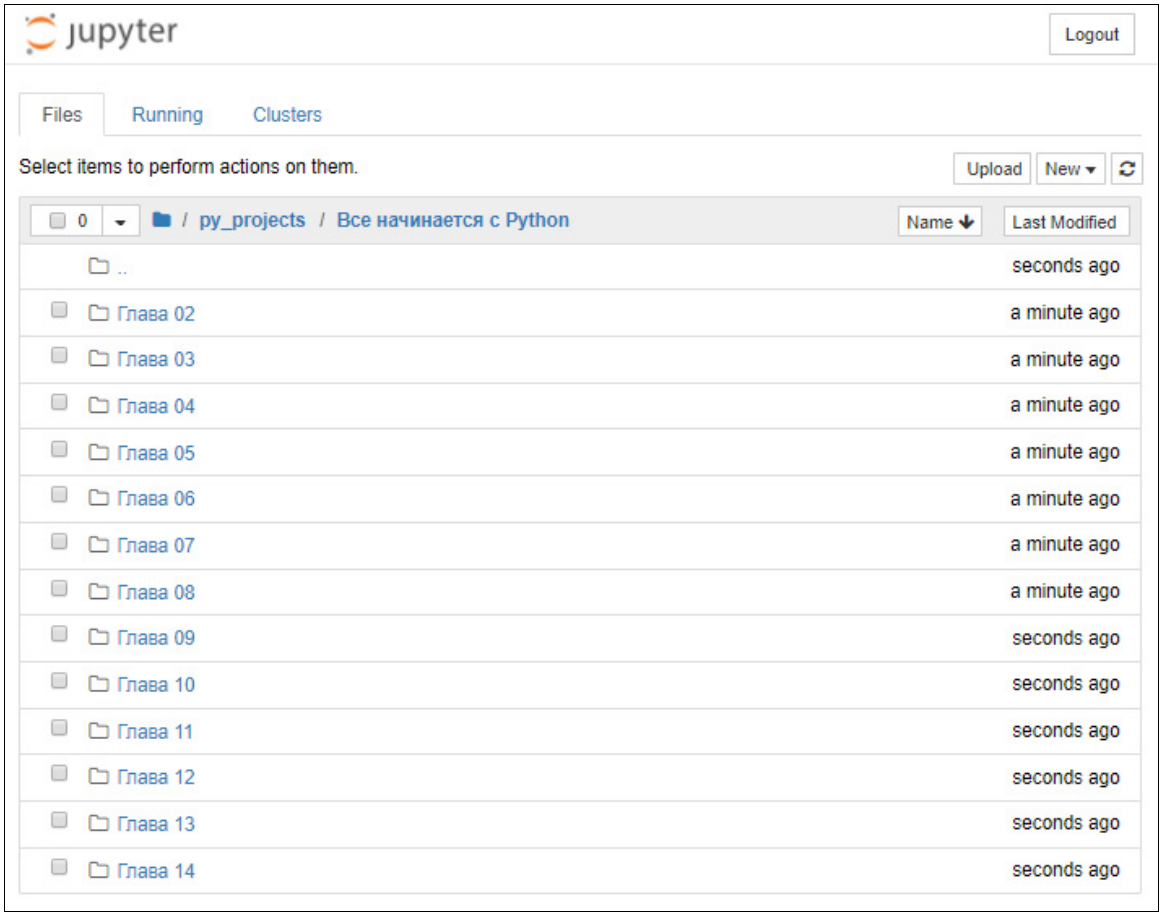


Рис. 2. Рабочее окно Jupyter в браузере

ГЛАВА 1

Введение в компьютеры и программирование

1.1 Введение

Давайте подумаем, каким образом люди используют компьютеры. В учебных заведениях компьютеры нужны для составления докладов и написания сочинений, поиска статей, пересылке электронных писем и участия в онлайновых учебных занятиях. На работе люди используют компьютеры, чтобы анализировать данные, составлять презентации, проводить деловые транзакции, общаться с клиентами и коллегами, управлять машинами на заводах и др. Дома компьютеры используются для оплаты счетов, покупки в онлайновых магазинах, общения с друзьями и семьей и, конечно же, для развлечений. Не забывайте также, что сотовые телефоны, планшеты, смартфоны, автомобильные системы навигации и множество других устройств — это тоже компьютеры. В повседневной жизни разнообразие применений компьютеров почти безгранично.

Компьютеры умеют выполнять такой широкий круг задач, потому что они программируемы. Из этого следует, что компьютеры приспособлены выполнять не одну задачу, а любую задачу, которую их программы предписывают им выполнить. *Программа* — это набор инструкций, которые компьютер исполняет с целью решения задачи. Например, на рис. 1.1 представлены снимки экранов с двумя популярными программами — Microsoft Word и PowerPoint.

Программы принято называть *программным обеспечением* (ПО). Программное обеспечение имеет для компьютера решающее значение, потому что оно управляет всем, что делает компьютер. Все ПО, которое мы используем, чтобы сделать наши компьютеры полезными, создается людьми — программистами, или разработчиками программного обеспечения. *Программист*, или *разработчик программного обеспечения*, — это человек с профессиональной подготовкой и квалификацией, необходимыми для проектирования, создания и тестирования компьютерных программ. Наверняка вы знаете, что результаты работы программистов используются в бизнесе, медицине, правительственных учреждениях, правоохранительных органах, сельском хозяйстве, научных кругах, развлекательных заведениях и многих других сферах человеческой деятельности.

С помощью этой книги вы знакомитесь с фундаментальными понятиями и принципами программирования на языке Python. Язык Python идеально подходит для новичков, потому что он легок в освоении, и программы пишутся на нем очень быстро. К тому же Python — это мощный язык, который популярен среди профессиональных разработчиков программного обеспечения. Не раз сообщалось, что Python используется в Google, NASA, YouTube, на Нью-Йоркской фондовой бирже, различными компаниями, специализирующимися на разработке игр, и многими другими организациями.

Прежде чем приступить к исследованию принципов программирования, необходимо понять несколько основных идей относительно компьютеров и принципов их работы. Эта глава за-

ложит прочный фундамент знаний, на который можно будет постоянно опираться по ходу изучения информатики. Сначала мы обсудим физические компоненты, из которых компьютеры обычно состоят. Затем мы рассмотрим способы хранения компьютерами данных и выполнения ими программ. И наконец, будет предложено краткое введение в программное обеспечение, которое будет использоваться для написания программ на Python.

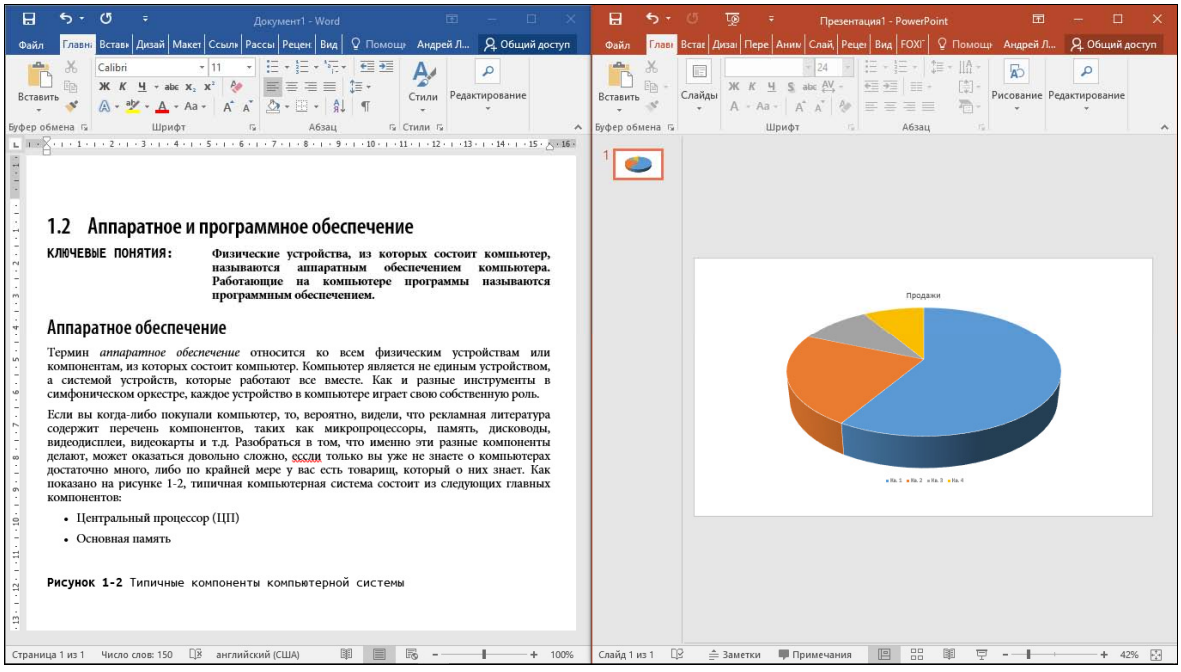


РИС. 1.1. Программы для обработки текстов (*слева*) и редактирования изображений (*справа*)

1.2 Аппаратное и программное обеспечение

КЛЮЧЕВЫЕ ПОЛОЖЕНИЯ

Физические устройства, из которых состоит компьютер, называются аппаратным обеспечением компьютера. Работающие на компьютере программы называются программным обеспечением.

Аппаратное обеспечение

Термин *"аппаратное обеспечение"* (hardware) относится ко всем физическим устройствам или *компонентам*, из которых состоит компьютер. Компьютер представляет собой не единое устройство, а систему устройств, которые работают во взаимодействии друг с другом. Как и разнообразные инструменты в симфоническом оркестре, каждое устройство в компьютере играет собственную роль.

Если вы когда-либо покупали компьютер, то, вероятно, видели, что рекламная литература содержит перечень компонентов, таких как микропроцессоры, память, дисководы, видеодисплеи, видеокарты и т. д. Довольно сложно разобраться в том, что именно делают эти разнообразные компоненты, если только вы уже не знаете о компьютерах достаточно много

либо у вас нет товарища, который о них что-то знает. Как показано на рис. 1.2, типичная компьютерная система состоит из следующих главных компонентов:

- центрального процессора (ЦП);
- основной памяти;
- вторичных устройств хранения данных;
- устройств ввода;
- устройств вывода.

Давайте взглянем на каждый из этих компонентов поближе.

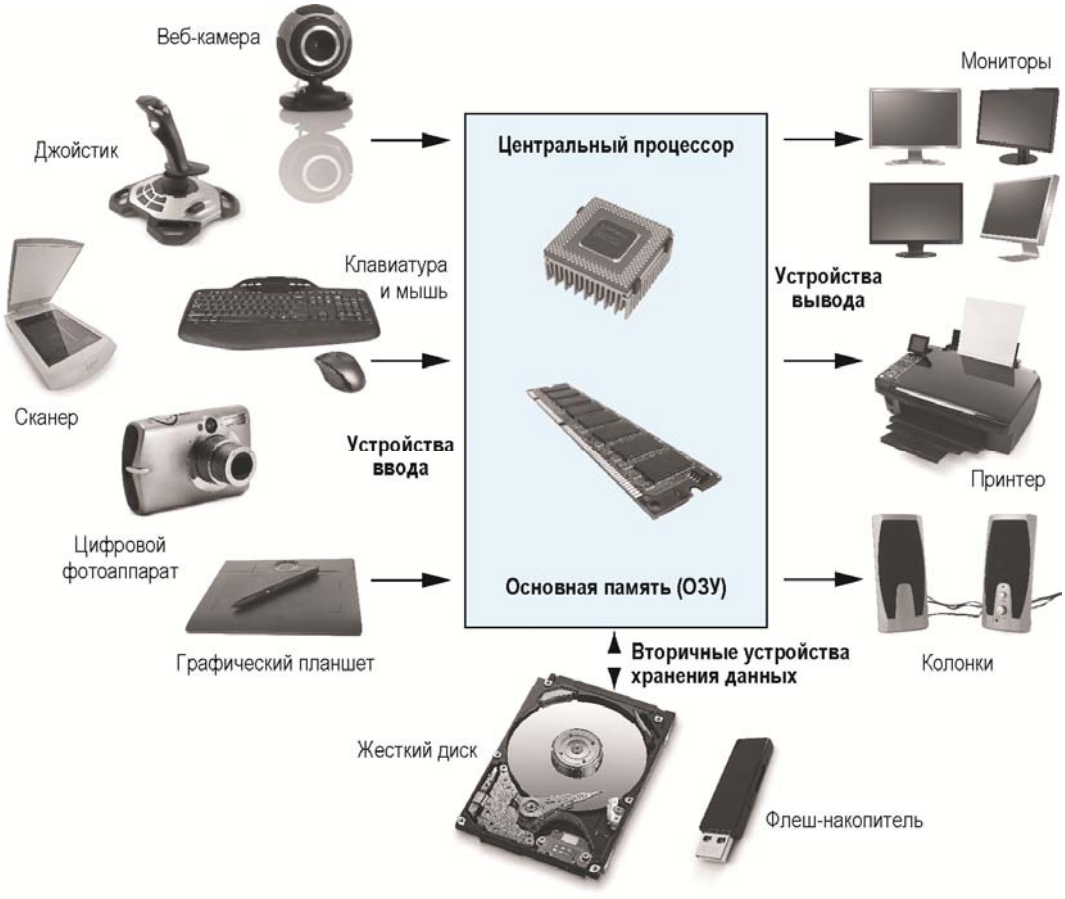


РИС. 1.2. Типичные компоненты компьютерной системы

Центральный процессор

Когда компьютер занят задачами, которые программа поручает ему выполнить, мы говорим, что компьютер *выполняет*, или *исполняет*, программу. *Центральный процессор* (ЦП) является той частью компьютера, которая фактически исполняет программы. Это самый важный компонент в компьютере, потому что без него компьютер не сможет выполнять программы.

В самых ранних компьютерах ЦП представлял собой огромные устройства, состоящие из электрических и механических компонентов, таких как вакуумные лампы и переключатели.

На фотографии рис. 1.3 две женщины работают с историческим компьютером ENIAC. ENIAC, который многими считается первой в мире программируемой электронно-вычислительной машиной, был создан в 1945 г. с целью вычисления орудийных баллистических таблиц для армии США[1]. Эта машина, которая первоначально представляла один большой ЦП, имела высоту 2,5 м, ширину 30,5 м и весила 30 т.

В наше время ЦП представляют собой миниатюрные кристаллы интегральной микросхемы, которые называются *микропроцессорами*. На рис. 1.4 представлена фотография техническо-

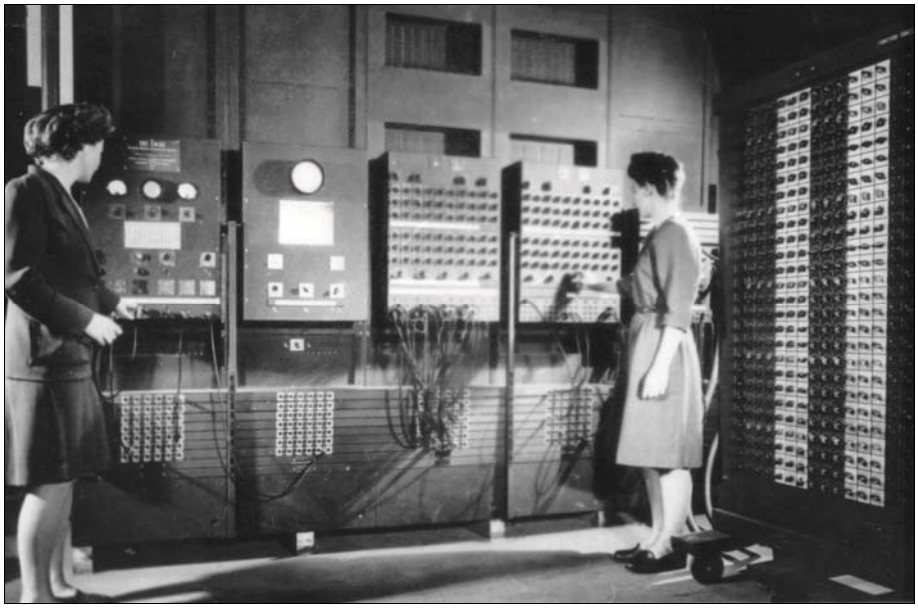


РИС. 1.3. Компьютер ENIAC
(с разрешения фотофонда U.S. Army Historic Computer Images)

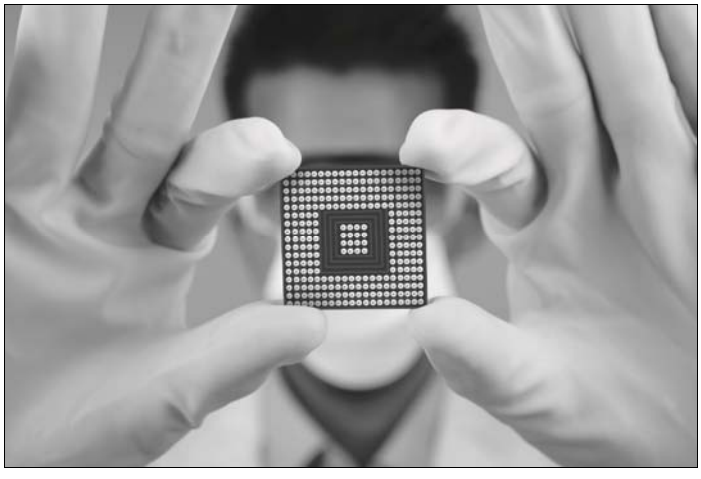


РИС. 1.4. Техник-лаборант держит современный процессор

[1] Первый советский компьютер, МЭСМ, был создан несколько позже под руководством С. А. Лебедева. Соответствующие работы начались только с осени 1948 г. и в конце 1951 г. МЭСМ прошел испытания и был принят в эксплуатацию Комиссией АН СССР во главе с академиком Келдышем. — *Прим. пер.*

го специалиста лаборатории, который держит в руках современный микропроцессор. Помимо того, что они намного меньше старых электромеханических ЦП в ранних компьютерах, микропроцессоры к тому же гораздо мощнее.

Основная память

Основную память можно представить, как рабочую область компьютера. Это то место, где компьютер хранит программу, пока та выполняется, и данные, с которыми программа работает. Например, предположим, вы используете программу обработки текста, чтобы написать работу для одного из своих учебных занятий. Пока вы это делаете, программа обработки текста и ваше сочинение хранятся в основной памяти.

Основную память принято называть *оперативным запоминающим устройством* (ОЗУ), или запоминающим устройством с произвольным доступом (ЗУПД), или просто *оперативной памятью*. Называется она так, потому что ЦП способен получать немедленный доступ к данным, хранящимся в любой произвольной единице хранения в ОЗУ. Как правило, ОЗУ является *энергозависимым* типом памяти, которая используется только для временного хранения, пока программа работает. Когда компьютер выключают, содержимое ОЗУ стирается. В вашем компьютере ОЗУ размещен в кристаллах интегральной микросхемы, подобных тем, которые показаны на рис. 1.5.

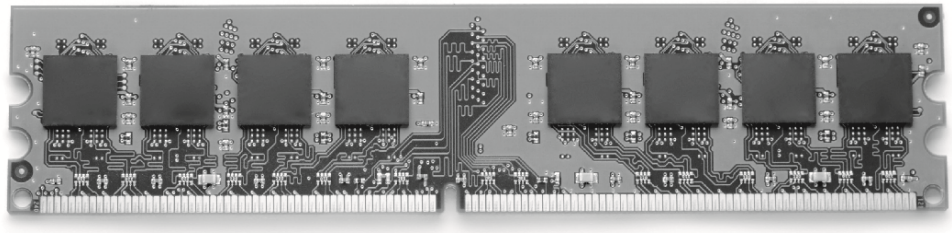


РИС. 1.5. Кристаллы интегральной микросхемы памяти

Вторичные устройства хранения

Вторичное устройство хранения — это тип памяти, который может хранить данные в течение долгих промежутков времени, даже когда к компьютеру не подведено электропитание. В обычных условиях программы хранятся во вторичной памяти и загружаются в основную память по мере необходимости. Важные данные, такие как документы текстового редактора, данные платежных ведомостей и складских запасов, тоже хранятся во вторичной памяти.

Наиболее распространенным типом вторичного устройства хранения является *жесткий диск*. Традиционный жесткий диск сохраняет данные путем их магнитной записи на вращающийся круговой диск. Все бóльшую популярность приобретают *твердотельные диски*, которые сохраняют данные в твердотельной памяти. Твердотельный диск не имеет подвижных частей и работает быстрее, чем традиционный диск. В большинстве компьютеров обязательно имеется какое-то вторичное устройство хранения, традиционный диск или твердотельный диск, смонтированное внутри своего корпуса. Имеются также вторичные устройства хранения, которые присоединяются к одному из коммуникационных портов компьютера. Вторичные устройства хранения используются для создания резервных копий важных данных либо для перемещения данных на другой компьютер.

Для копирования данных и их перемещения на другие компьютеры помимо вторичных устройств хранения были созданы разнообразные типы устройств. Возьмем, к примеру, *USB-диски*. Эти небольшие устройства подсоединяются к порту USB (универсальной последовательной шине) компьютера и определяются в системе как внешний жесткий диск. Правда, эти диски на самом деле не содержат дисковых пластин. Они хранят данные в памяти особого типа — во *флеш-памяти*. USB-диски, именуемые также *картами памяти* и *флеш-накопителями*, имеют небольшую стоимость, надежны и достаточно маленькие, чтобы носить их в кармане одежды.

Оптические устройства, такие как CD (компакт-диск) и DVD (цифровой универсальный диск), тоже широко используются для хранения данных. На оптическом диске данные записываются не магнитным способом, а кодируются как серия углублений на поверхности диска. С целью обнаружения этих углублений и, следовательно, чтения закодированных данных в CD- и DVD-приводах используется лазерный луч. Оптические диски содержат большие объемы данных, и по этой причине, записываемые CD- и DVD-приводы широко используются для создания резервных копий данных.

Устройства ввода

Вводом называются любые данные, которые компьютер собирает от людей и от различных устройств. Компонент, который собирает данные и отправляет их в компьютер, называется *устройством ввода*. Типичными устройствами ввода является клавиатура, мышь, сенсорный экран, сканер, микрофон и цифровая фотокамера. Дисководы и оптические диски можно тоже рассматривать, как устройства ввода, потому что из них извлекаются программы и данные, которые затем загружаются в оперативную память компьютера.

Устройства вывода

Выводом являются любые данные, которые компьютер производит для людей или для различных устройств. Это может быть торговый отчет, список имен или графическое изображение. Данные отправляются в *устройство вывода*, которое их форматирует и предъявляет. Типичными устройствами вывода являются видеодисплеи и принтеры. Дисководы можно тоже рассматривать, как устройства вывода, потому что компьютерная система отправляет на них данные с целью их сохранения.

Программное обеспечение

Для функционирования компьютера нужно программное обеспечение. Все, что компьютер делает с момента нажатия на кнопку включения питания и до момента выключения компьютерной системы, находится под контролем программного обеспечения. Имеются две общие категории программного обеспечения: системное программное обеспечение и прикладное программное обеспечение. Большинство компьютерных программ четко вписывается в одну из этих двух категорий. Давайте взглянем на каждую из них поближе.

Системное программное обеспечение

Программы, которые контролируют и управляют основными операциями компьютера, обычно называются *системным программным обеспечением*. К системному ПО, как правило, относятся следующие типы программ.

Операционная система (ОС) — фундаментальный набор программ на компьютере. Она управляет внутренними операциями аппаратного обеспечения компьютера и всеми подключенными к компьютеру устройствами, позволяет сохранять данные и их получать из устройств хранения, обеспечивает выполнение других программ. Популярные операционные системы для ноутбуков и настольных компьютеров включают Windows, Mac OS X и Linux. Популярные операционные системы для мобильных устройств — Android и iOS.

Обслуживающая программа, или *утилита*, выполняет специализированную задачу, которая расширяет работу компьютера или гарантирует сохранность данных. Примерами обслуживающих, или сервисных, программ являются вирусные сканеры, программы сжатия файлов и программы резервного копирования данных.

Инструменты разработки программного обеспечения — это программы, которые программисты используют для создания, изменения и тестирования программного обеспечения. Ассемблеры, компиляторы и интерпретаторы являются примерами программ, которые попадают в эту категорию.

Прикладное программное обеспечение

Программы, которые делают компьютер полезным для повседневных задач, называются *прикладным программным обеспечением*. На выполнение этих программ на своих компьютерах люди по обыкновению тратят большую часть своего времени. На рис. 1.1 был представлен снимок экрана с двумя популярными приложениями: программой обработки текста Microsoft Word и программой для презентаций PowerPoint. В качестве других примеров прикладного ПО можно назвать программы для работы с электронными таблицами, почтовыми программами, веб-браузерами и компьютерными играми.

Контрольная точка

1.1. Что такое программа?

1.2. Что такое аппаратное обеспечение?

1.3. Перечислите пять главных компонентов компьютерной системы.

1.4. Какая часть компьютера исполняет программы фактически?

1.5. Какая часть компьютера служит рабочей областью для хранения программы и ее данных, пока программа работает?

1.6. Какая часть компьютера содержит данные в течение долгого времени, даже когда к компьютеру не подведено электропитание?

1.7. Какая часть компьютера собирает данные от людей и от различных устройств?

1.8. Какая часть компьютера форматирует и предоставляет данные людям и подключенным к нему устройствам?

1.9. Какой фундаментальный набор программ управляет внутренними операциями аппаратного обеспечения компьютера?

1.10. Как называется программа, которая выполняет специализированную задачу, в частности, такие программы, как вирусный сканер, программа сжатия файлов или программа резервного копирования данных?

1.11. К какой категории программного обеспечения принадлежат программы обработки текста, программы по работе электронными таблицами, почтовые программы, веб-браузеры и компьютерные игры?

1.3 Как компьютеры хранят данные

КЛЮЧЕВЫЕ ПОЛОЖЕНИЯ

Все хранящиеся в компьютере данные преобразуются в последовательности, состоящие из нулей и единиц.

Память компьютера разделена на единицы хранения, которые называются *байтами*. Одного байта памяти достаточно только для того, чтобы разместить одну букву алфавита или небольшое число. Для того чтобы делать что-то более содержательное, компьютер должен иметь очень много байтов. Большинство компьютеров сегодня имеет миллионы или даже миллиарды байтов оперативной памяти.

Каждый байт разделен на восемь меньших единиц хранения, которые называются *битами*, или *разрядами*. Термин *"бит"* происходит от англ. binary digit и переводится как *двоичная цифра*. Программисты обычно рассматривают биты, как крошечные переключатели, которые могут находиться в одном из двух положений: включено или выключено. Однако на самом деле биты не являются «переключателями», по крайней мере не в стандартном смысле. В большинстве компьютерных систем биты — это крошечные электрические компоненты, которые могут содержать либо положительный, либо отрицательный заряд. Программисты рассматривают положительный заряд, как переключатель в положении «Включено», и отрицательный заряд, как переключатель в положении «Выключено». На рис. 1.6 показано, как программист может рассматривать байт памяти: как коллекцию переключателей, каждый из которых установлен в положение «Вкл» либо «Выкл».

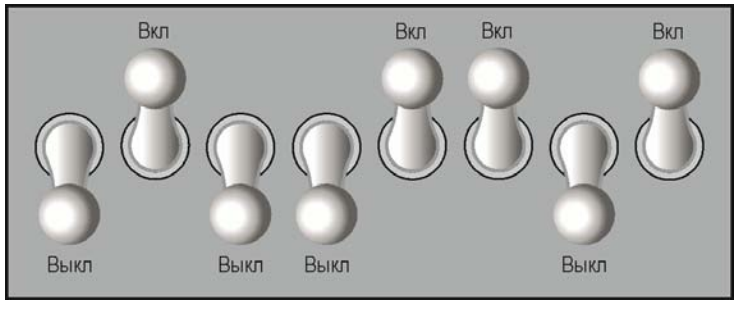


РИС. 1.6. Рассматривайте байт как восемь переключателей

Когда порция данных сохраняется в байте, компьютер размещает восемь битов в двухпозиционной комбинации «включено-выключено», которая представляет данные. Например, на рис. 1.7, слева показано, как в байте будет размещено число 77, а справа — латинская буква A. Далее мы объясним, каким образом эти комбинации определяются.

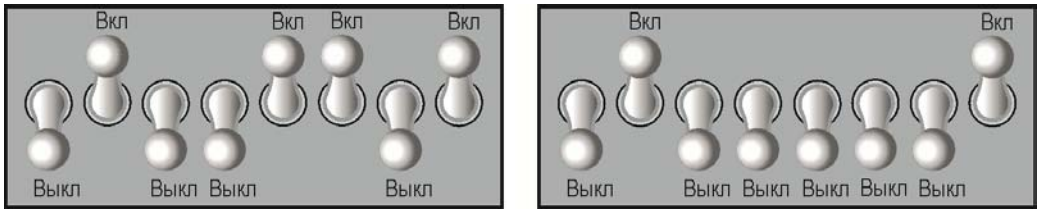


РИС. 1.7. Комбинации двоичных разрядов для числа 77 (*слева*) и латинской буквы A (*справа*)

Хранение чисел

Бит используется для представления чисел в весьма ограниченной форме. В зависимости от того, "включен" он или "выключен", бит может представлять одно из двух разных значений. В компьютерных системах выключенный бит представляет число 0, а включенный — число 1. Это идеально соответствует *двоичной системе исчисления*, в которой все числовые значения записываются как последовательности нулей и единиц. Вот пример числа, которое записано в двоичной системе исчисления:

```
10011101
```

Позиция каждой цифры в двоичном числе соответствует определенному значению. Как показано на рис. 1.8, начиная с самой правой цифры и двигаясь влево, значения позиций равняются 2^0, 2^1, 2^2, 2^3 и т. д. На рис. 1.9 показана та же схема, но здесь позиции вычислены. Начиная с самой правой цифры и двигаясь влево, значения позиций равняются 1, 2, 4, 8 и т. д.

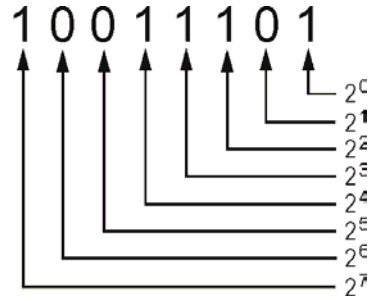


РИС. 1.8. Значения двоичных разрядов как степеней 2

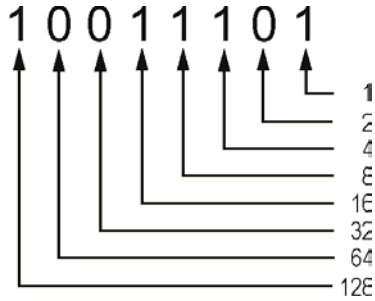


РИС. 1.9. Значения двоичных разрядов

Для того чтобы определить значение двоичного числа, нужно просто сложить позиционные значения всех единиц. Например, в двоичном числе 10011101 позиционные значения единиц равняются 1, 4, 8, 16 и 128 (рис. 1.10). Сумма всех этих позиционных значений равняется 157. Значит, двоичное число 10011101 равняется 157 в десятичной системе исчисления.

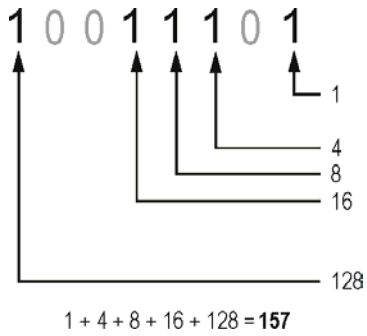


РИС. 1.10. Определение значения 10011101

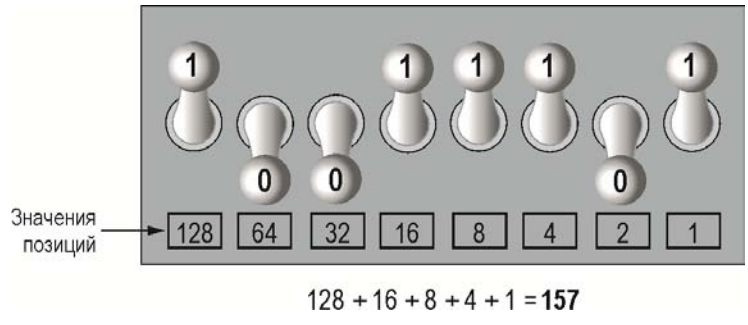


РИС. 1.11. Комбинация двоичных разрядов для числа 157

На рис. 1.11 показано, как можно изобразить хранение числа 157 в байте оперативной памяти. Каждая 1 представлена битом в положении «Вкл», а каждый 0 — битом в положении «Выкл».

Когда всем битам в байте назначены нули (т. е. они выключены), значение байта равняется 0. Когда всем битам в байте назначены единицы (т. е. они включены), байт содержит самое

большое значение, которое в нем может быть размещено. Самое большое значение, которое может быть размещено в байте, равняется 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255. Этот предел является следствием того, что в байте всего восемь бит.

А если нужно сохранить число, которое больше 255? Ответ прост: использовать еще один байт. Например, мы расположили два байта вместе. В итоге получаем 16 бит. Позиционные значения этих 16 бит будут 2^0, 2^1, 2^2, 2^3 и т. д. до 2^{15}. Максимальное значение, которое можно разместить в двух байтах, равно 65 535 (рис. 1.12). Если же нужно сохранить еще большее число, то для этого потребуется больше байтов.

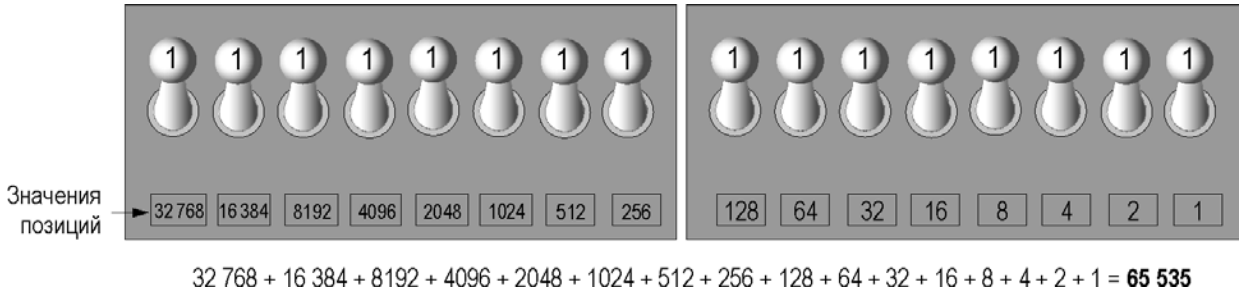


РИС. 1.12. Для большого числа использованы два байта

СОВЕТ

Если вы чувствуете, что этот материал дается с трудом, расслабьтесь! Программируя, вам не придется заниматься преобразованием чисел в двоичную форму. Знание о том, как этот процесс происходит в компьютере, поможет вам в ходе обучения, и в долгосрочной перспективе это знание сделает из вас хорошего программиста.

Хранение символов

Любая порция данных в оперативной памяти компьютера должна храниться как двоичное число. Это относится и к символам, таким как буквы и знаки препинания. Когда символ сохраняется в памяти, он сначала преобразуется в цифровой код. И затем этот цифровой код сохраняется в памяти как двоичное число.

За прошедшие годы для представления символов в памяти компьютера были разработаны различные схемы кодирования. Исторически самой важной из этих схем кодирования является схема кодирования *ASCII* (American Standard Code for Information Interchange — *американский стандартный код обмена информацией*). ASCII представляет собой набор из 128 цифровых кодов, которые обозначают английские буквы, различные знаки препинания и другие символы. Например, код ASCII для прописной английской буквы A (латинской) равняется 65. Когда на компьютерной клавиатуре вы набираете букву A в верхнем регистре, в памяти сохраняется число 65 (как двоичное число, разумеется). Это показано на рис. 1.13.

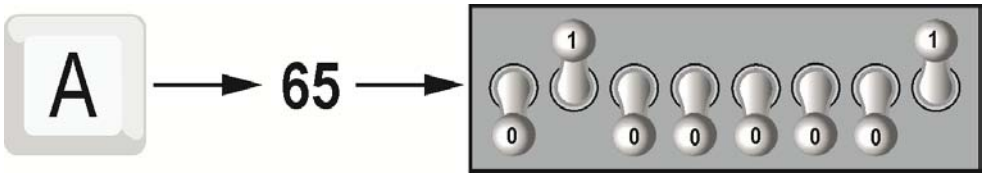


РИС. 1.13. Буква A хранится в памяти как число 65

ПРИМЕЧАНИЕ
Аббревиатура ASCII произносится «аски».

На случай, если вам любопытно, код ASCII для английской B в верхнем регистре равняется 66, для C в верхнем регистре — 67 и т. д. В *приложении 3* приведены все коды ASCII и символы, которые ими представлены.

Набор символов ASCII был разработан в начале 1960-х годов и в конечном счете принят почти всеми производителями компьютеров. Однако схема кодирования ASCII имеет ограничения, потому что она определяет коды только для 128 символов. Для того чтобы это исправить, в начале 1990-х годов был разработан набор символов Юникода (Unicode). Это широкая схема кодирования, совместимая с ASCII, которая может также представлять символы многих языков мира. Сегодня Юникод быстро становится стандартным набором символов, используемым в компьютерной индустрии.

Хранение чисел повышенной сложности

Итак, вы познакомились с числами и их хранением в памяти. Во время чтения вы, возможно, подумали, что двоичная система исчисления может использоваться для представления только целых чисел, начиная с 0. Представить отрицательные и вещественные числа (такие как 3.14159) при помощи рассмотренного нами простого двоичного метода нумерации невозможно.

Однако компьютеры способны хранить в памяти и отрицательные, и вещественные числа, но для этого помимо двоичной системы исчисления в них используются специальные схемы кодирования. Отрицательные числа кодируются с использованием метода, который называется *дополнением до двух*, а вещественные числа кодируются в форме записи с *плавающей точкой*[1]. От вас не требуется знать, как эти схемы кодирования работают, за исключением того, что они используются для преобразования отрицательных и вещественных чисел в двоичный формат.

Другие типы данных

Компьютеры принято называть цифровыми устройствами. Термин *"цифровой"* применяется для описания всего, что использует двоичные числа. *Цифровые данны*е — это данные, которые хранятся в двоичном формате, *цифровое устройство* — любое устройство, которое работает с двоичными данными. Мы уже рассмотрели приемы хранения чисел и символов в двоичном формате, но компьютеры также работают со многими другими типами цифровых данных.

Например, возьмем снимки, которые вы делаете своей цифровой фотокамерой. Эти изображения состоят из крошечных точек цвета, которые называются пикселами. (Термин «пиксел» образован от слов picture element — *элемент изображения*.) Каждый пиксел в изображении преобразуется в числовой код, который представляет цвет пиксела (рис. 1.14). Числовой код хранится в оперативной памяти как двоичное число.

---

[1] Числа с плавающей точкой (или запятой) — это общепринятый способ представления чисел, имеющих десятичный разделитель. В англоязычных странах в качестве десятичного разделителя используется точка (.), в большинстве остальных — запятая (,). — *Прим. пер.*

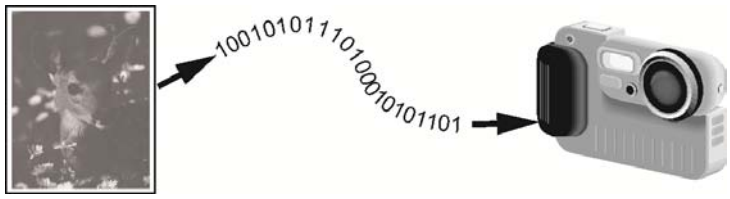


РИС. 1.14. Цифровое изображение, хранимое в двоичном формате

Музыка, которую вы воспроизводите на своем проигрывателе CD, iPod или MP3-плеере, тоже является цифровой. Цифровая песня разбивается на небольшие порции, которые называются *выборками*, или семплами[1]. Каждая выборка конвертируется в двоичное число, которое может быть сохранено в памяти. Чем больше выборок, на которые песня подразделяется, тем ее качество звучания становится ближе к оригиналу музыкальной композиции при ее воспроизведении. Песня с качеством звучания CD подразделяется более чем на 44 000 выборки в секунду!

Контрольная точка

1.12. Какого объема памяти достаточно для хранения буквы алфавита или небольшого числа?

1.13. Как называется крошечный "переключатель", который может быть установлен в положение "включено" или "выключено"?

1.14. В какой системе исчисления все числовые значения записываются как последовательности нулей и единиц?

1.15. Какова задача схемы кодирования ASCII?

1.16. Какая схема кодирования является достаточно широкой, чтобы включать символы многих языков мира?

1.17. Что означают термины "цифровые данные" и "цифровое устройство"?

1.4 Как программа работает

КЛЮЧЕВЫЕ ПОЛОЖЕНИЯ

ЦП компьютера может понимать только те инструкции, которые написаны на машинном языке. Поскольку людям очень сложно писать какие-либо программы на машинном языке, были придуманы другие языки программирования.

Ранее мы установили, что ЦП является самым важным компонентом в компьютере, потому что это та его часть, которая выполняет программы. Иногда ЦП называют "мозгом" компьютера и даже используют эпитет "умный" (smart). Хотя эти метафоры встречаются очень часто, следует понимать, что ЦП не является мозгом, и его нельзя назвать умным. ЦП — это

[1] Здесь речь идет о частоте дискретизации сигнала, т. е. количестве замеров величины сигнала, осуществляемых в одну секунду. Дискретизация сигнала также называется частотой выборки или частотой семплирования. — *Прим. пер.*

электронное устройство, которое предназначено для выполнения конкретных вещей. В частности, ЦП предназначен для выполнения таких операций, как:

- чтение порции данных из оперативной памяти;
- сложение двух чисел;
- вычитание одного числа из другого;
- умножение двух чисел;
- деление одного числа на другое число;
- перемещение порции данных из одной ячейки оперативной памяти в другую;
- определение, равно ли одно значение другому значению.

Как видно из этого списка, ЦП выполняет простые операции с порциями данных. Однако ЦП ничего не делает самостоятельно. Ему нужно сообщить, что именно надо сделать, и это является целью программы. Программа — это не более чем список инструкций, которые заставляют ЦП выполнять операции.

Каждая инструкция в программе — это команда, которая поручает ЦП выполнить конкретную операцию. Вот пример инструкции, которая может появиться в программе:

```
10110000
```

Для нас это всего лишь вереница нулей и единиц. Для ЦП она является инструкцией с указанием выполнить какую-то операцию[1] и записывается в виде нулей и единиц, потому что ЦП понимают только те инструкции, которые написаны на *машинном языке*, а инструкции машинного языка всегда имеют двоичную структуру.

Инструкция на машинном языке существует для каждой операции, которую ЦП способен выполнить. Весь перечень инструкций, которые ЦП может исполнять, называется *набором инструкций* ЦП.

ПРИМЕЧАНИЕ

Сегодня имеется несколько микропроцессорных компаний, которые изготавливают центральные процессоры. Самые известные — Intel, AMD и Motorola. Если вы внимательно посмотрите на свой компьютер, то сможете найти наклейку с логотипом его микропроцессора.

Каждый бренд микропроцессора имеет собственную уникальную систему инструкций, которая, как правило, понятна микропроцессорам только того же бренда. Например, микропроцессоры Intel понимают одинаковые инструкции, но они не понимают инструкции для микропроцессоров Motorola.

Ранее показанная инструкция на машинном языке является примером всего одной инструкции. Однако, для того чтобы компьютер делал что-то содержательное, ему требуется гораздо больше инструкций. Поскольку операции, которые ЦП умеет выполнять, являются элементарными, содержательная задача может быть выполнена, только если ЦП выполняет много операций. Например, если вы хотите, чтобы ваш компьютер вычислил сумму процентного дохода, которую вы получите на своем сберегательном счете в этом году, то ЦП должен будет исполнить большое количество инструкций в надлежащей последовательности. Вполне обычной является ситуация, когда программы содержат тысячи или даже миллионы инструкций на машинном языке.

---

[1] Приведенный пример является фактической командой для микропроцессора Intel. Она поручает микропроцессору переместить значение в ЦП.

Программы обычно хранятся на вторичном устройстве хранения, таком как жесткий диск. Когда вы устанавливаете, или, говоря техническим языком, инсталлируете программу на компьютер, она, как правило, копируется на диск компьютера с диска CD-ROM или скачивается с веб-сайта.

Хотя программа размещается на вторичном устройстве хранения, таком как жесткий диск, тем не менее при каждом ее исполнении центральным процессором она должна копироваться в основную память, или ОЗУ. Например, предположим, что на диске вашего компьютера имеется программа обработки текста. Для ее исполнения вы делаете двойной щелчок мышью по значку программы. Это приводит к тому, что программа копируется с диска в основную память. Затем ЦП компьютера исполняет копию программы, которая находится в основной памяти (рис. 1.15).

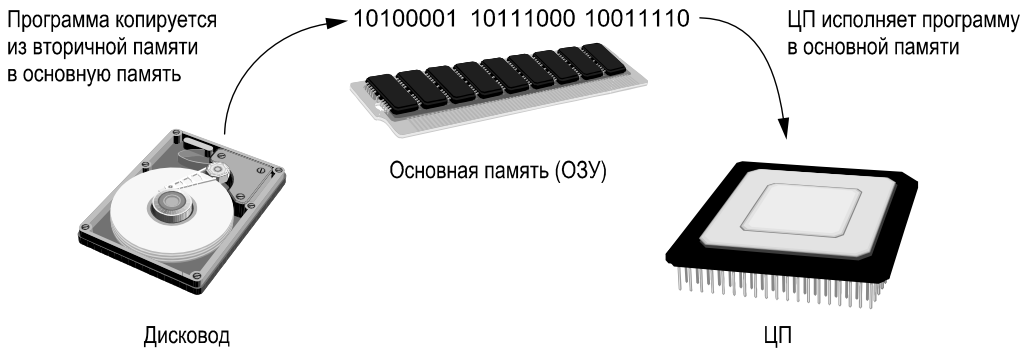


РИС. 1.15. Программа копируется в основную память и затем исполняется

Когда ЦП исполняет инструкции в программе, он участвует в процессе, который называется циклом *выборки-декодирования-исполнения*. Этот цикл состоит из трех шагов и повторяется для каждой инструкции в программе. Эти шаги следующие:

1. **Выборка.** Программа представляет собой длинную последовательность инструкций на машинном языке. Первый шаг цикла состоит в выборке, или чтении, следующей инструкции из памяти и ее доставки в ЦП.

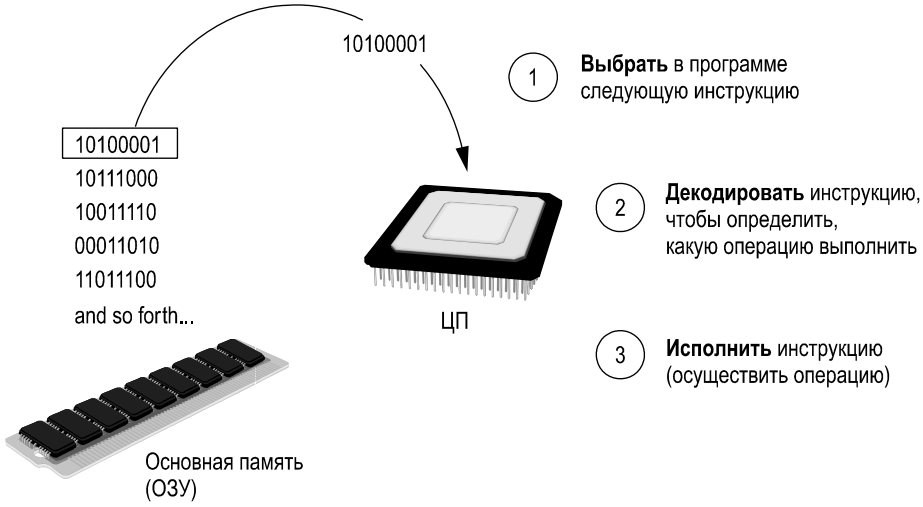


РИС. 1.16. Цикл выборки-декодирования-исполнения

2. **Декодирование.** Инструкция на машинном языке представляет собой двоичное число, обозначающее команду, которая даст указание ЦП выполнить операцию. На этом шаге ЦП декодирует инструкцию, которая только что была выбрана из оперативной памяти, чтобы определить, какую операцию он должен выполнить.
3. **Исполнение.** Последний шаг в цикле состоит в исполнении, или осуществлении, операции.

На рис. 1.16 проиллюстрированы эти шаги.

От машинного языка к языку ассемблера

Компьютеры могут исполнять только те программы, которые написаны на машинном языке. Как упоминалось ранее, программа может иметь тысячи или даже миллионы двоичных инструкций и написание такой программы было бы очень утомительной и трудоемкой работой. Программировать на машинном языке будет тоже очень трудно, потому что если разместить 0 или 1 в неправильном месте, то это вызовет ошибку.

Хотя ЦП компьютера понимает только машинный язык, люди практически неспособны писать программы на нем. По этой причине с первых дней вычислительных систем[1] в качестве альтернативы машинному языку был создан *язык ассемблера* (от англ. *assembly* — сборка, монтаж, компоновка). Вместо двоичных чисел, которые соответствуют инструкциям, в языке ассемблера применяются короткие слова, которые называются *мнемониками*, или мнемокодами. Например, на языке ассемблера мнемоника `add`, как правило, означает сложить числа, `mul` — умножить числа, `mov` — переместить значение в ячейку оперативной памяти. Когда для написания программы программист использует язык ассемблера, вместо двоичных чисел он имеет возможность писать короткие мнемоники.

ПРИМЕЧАНИЕ

Существует много разных версий языка ассемблера. Ранее было упомянуто, что каждый бренд ЦП имеет собственную систему инструкций машинного языка. Как правило, каждый бренд ЦП также имеет свой язык ассемблера.

Вместе с тем программы на языке ассемблера ЦП исполнять не может. ЦП понимает только машинный язык, поэтому для перевода программы, написанной на языке ассемблера, в программу на машинном языке применяется специальная программа, которая называется *ассемблером*. Этот процесс показан на рис. 1.17. Программа на машинном языке, которая создается ассемблером, затем может быть исполнена центральным процессором.

РИС. 1.17. Ассемблер переводит программу на языке ассемблера в программу на машинном языке

[1] Самый первый язык ассемблера, по всей видимости, был разработан в 1940-х годах в Кембриджском университете с целью его применения на историческом компьютере под названием EDSAC.

Высокоуровневые языки

Хотя язык ассемблера делает ненужным написание двоичных инструкций на машинном языке, он имеет свои сложности. Язык ассемблера прежде всего является непосредственной заменой машинного языка и подобно машинному языку он обязывает вас к тому, чтобы вы хорошо разбирались в ЦП. Язык ассемблера также требует, чтобы вы писали большое количество инструкций даже для самой простой программы. Поскольку язык ассемблера по своей природе непосредственно связан с машинным языком, он называется *низкоуровневым языком*.

В 1950-х годах появилось новое поколение языков программирования, которые называются *высокоуровневыми языками*. Высокоуровневый язык позволяет создавать мощные и сложные программы, не требуя знаний особенностей работы ЦП, и написания большого количества низкоуровневых инструкций. Кроме того, в большинстве высокоуровневых языков используются легко понимаемые слова. Например, если бы программист использовал COBOL (который был одним из ранних высокоуровневых языков, созданных в 1950-х годах), то, к примеру, для отображения на мониторе сообщения "Привет, мир!", он написал бы следующую инструкцию:

```
DISPLAY "Привет, мир!"
```

Python — это современный, высокоуровневый язык программирования, который мы будем использовать в этой книге. В Python сообщение "Привет, мир!" выводится на экран при помощи такой инструкции:

```
print('Привет, мир!')
```

Выполнение того же самого на языке ассемблера потребовало бы нескольких инструкций и глубокого знания того, как ЦП взаимодействует с компьютерным устройством вывода. Как видно из этого примера, высокоуровневые языки позволяют программистам сосредотачиваться на задачах, которые они хотят выполнять при помощи своих программ, а не деталях того, как ЦП будет исполнять эти программы.

Начиная с 1950-х годов были созданы тысячи высокоуровневых языков. В табл. 1.1 перечислены некоторые из наиболее известных языков.

Таблица 1.1. Языки программирования

Язык	Описание
Ada	Был создан в 1970-х годах прежде всего для приложений, использовавшихся Министерством обороны США. Язык назван в честь графини Ады Лавлейс, влиятельной исторической фигуры в области вычислительных систем
BASIC	Универсальная система символического кодирования для начинающих (Beginners All-purpose Symbolic Instruction Code) является языком общего назначения, который был первоначально разработан в начале 1960-х годов, как достаточно простой в изучении начинающими программистами. Сегодня существует множество разных версий языка BASIC
FORTRAN	Транслятор формул (FORmula TRANslator) был первым высокоуровневым языком программирования. Он был разработан в 1950-х годах для выполнения сложных математических вычислений
COBOL	Универсальный язык для коммерческих задач (Common Business-Oriented Language) был создан в 1950-х гг. для бизнес-приложений

Таблица 1.1 *(окончание)*

Язык	Описание
Pascal	Был создан в 1970 г. и первоначально был предназначен для обучения программированию. Этот язык был назван в честь математика, физика и философа Блеза Паскаля
C и C++	C и C++ (произносится как "си плюс плюс") являются мощными языками общего назначения, разработанными в компании Bell Laboratories. Язык C был создан в 1972 г., язык C++ был выпущен в 1983 г.
C#	C# произносится "си шарп". Этот язык был создан компанией Microsoft примерно в 2000 г. для разработки приложений на основе платформы Microsoft .NET
Java	Был создан компанией Sun Microsystems в начале 1990-х годов. Он используется для разработки программ, которые работают на одиночном компьютере либо через Интернет с веб-сервера
JavaScript	Был создан в 1990-х годах и используется на веб-страницах. Несмотря на его название JavaScript, с Java никак не связан
Python	Используемый в этой книге язык Python является языком общего назначения, который был создан в начале 1990-х годов. Он стал популярным в коммерческих и научных приложениях
Ruby	Ruby является языком общего назначения, который был создан в 1990-х годах. Он становится все более популярным языком для создания программ, которые работают на веб-серверах
Visual Basic	Широко известный как VB, является языком программирования Microsoft и средой разработки программного обеспечения, позволяет программистам быстро создавать Windows-ориентированные приложения. VB был создан в начале 1990-х годов

Ключевые слова, операторы и синтаксис: краткий обзор

Каждый высокоуровневый язык имеет свой набор предопределенных слов, которые программист должен использовать для написания программы. Слова, которые составляют высокоуровневый язык программирования, называются *ключевыми*, или *зарезервированными*, *словами*. Каждое ключевое слово имеет определенное значение и не может использоваться ни для какой другой цели. В табл. 1.2 перечислены все ключевые слова языка Python.

Таблица 1.2. Ключевые (зарезервированные) слова языка Python

`and`	`def`	`finally`	`in`	`or`	`while`
`as`	`del`	`for`	`is`	`pass`	`with`
`assert`	`elif`	`from`	`lambda`	`raise`	`yield`
`break`	`else`	`global`	`None`	`return`	
`class`	`except`	`if`	`nonlocal`	`True`	
`continue`	`False`	`import`	`not`	`try`	

В дополнение к ключевым словам языки программирования имеют *операторы*, которые выполняют различные операции с данными. Например, все языки программирования имеют математические операторы, которые выполняют арифметические вычисления. В Python,