Оглавление (кратко)

	Введение	23
1	Приступим	37
2	Сначала было значение	69
3	Принятие решений	109
4	Структуры данных	161
5	Функциональный код	215
4а	Наведение порядка в данных	261
6	Сведем все воедино	281
7	Модульное программирование	327
8	И снова об индексах и циклах	377
9	Длительное хранение данных	429
10	Так хочется большего	471
11	Интерактивные возможности	503
12	Путешествие в страну объектов	559
	Приложение: Десять ключевых тем (которые не были рассмотрены)	611

Содержание (подробно)

Введение

Настройте свой мозг на изучение программирования.

Конечно, вам хочется чему-то научиться, но у мозга могут быть свои планы на этот счет. Он контролирует, чтобы вы не забивали себе голову ненужной информацией. Есть ведь вещи и поважнее, например какой фильм пойти посмотреть на выходных или что надеть на вечеринку. Как же убедить мозг, что книга по программированию содержит нужную информацию?

Для кого эта книга	24
Мы знаем, о чем вы думаете	25
Мы считаем читателей книги учениками	26
Метапознание: осознание знаний	27
Вот что сделали мы	28
Вот что можете сделать вы, чтобы настроить свой мозг на обучение	29
Это важно	30
Необходимо установить Python	32
Как работать с исходными кодами	34
Благодарности	35
Команда рецензентов	36

учимся думать как программист

1 Приступим

Чтобы научиться писать программы, нужно думать как программист. В мире современных технологий все вокруг нас становится взаимосвязанным, настраиваемым, программируемым и в каком-то смысле **компьютерным**. Можно оставаться пассивным наблюдателем, а можно *научиться программировать*. Умея писать программный код, вы становитесь творцом, устанавливающим правила игры, ведь именно вы отдаете команды компьютеру. Но как освоить искусство программирования? Самое главное — это начать **думать как программист**. Далее нужно определиться с **языком программирования** — написанные на нем программы должны выполняться на всех целевых устройствах (компьютерах, смартфонах и любых других электронных гаджетах, снабженных процессором). Что это вам даст? Больше свободного времени, больше ресурсов и больше творческих возможностей для воплощения в жизнь задуманного.

Шаг за шагом	38
Как пишут программы	42
Понимают ли нас компьютеры?	43
Мир языков программирования	44
Ввод и выполнение кода Python	49
Краткая история Python	51
Тестируем среду	54
Сохранение кода	56
Первая программа готова!	57
Phrase-O-Matic	61
Запуск кода на выполнение	62

значения, переменные и типы данных

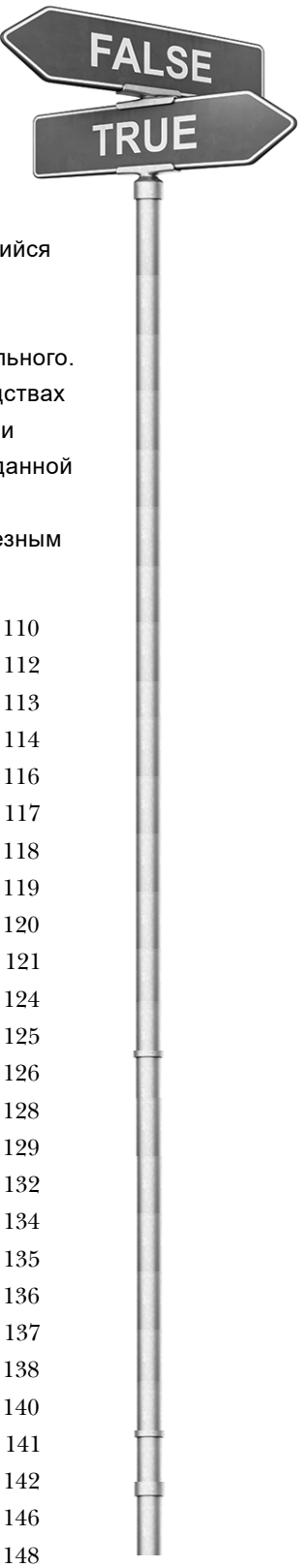
2 Сначала было значение

В реальности компьютеры умеют делать всего две вещи: сохранять значения и выполнять над ними операции. А как же редактирование текста, ретуширование фотографий и онлайн-покупки, спросите вы? Удивительно, но решение любых подобных задач в конечном счете сводится к выполнению **простых операций** над не менее **простыми значениями**. Человек, обладающий вычислительным мышлением, понимает, что это за операции и значения и как с их помощью создать что-то более сложное. В этой главе мы познакомимся с типами значений, узнаем, какие операции можно выполнять над ними и какую роль во всем этом играют переменные.

Калькулятор возраста собаки 70
От псевдокода к программному коду 72
Этап 1: запрос входных данных 73
Как работает функция `input()` 74
Сохранение значений в переменных 74
Сохранение вводимых данных в переменной 75
Этап 2: еще один запрос 75
Пора выполнить код 76
Ввод кода в редакторе 79
Подробнее о переменных 80
Добавляем выражения 81
Изменение переменных 82
Приоритет операторов 83
Вычисление выражений согласно приоритету операторов 84
Руки прочь от клавиатуры! 87
Этап 3: вычисление возраста собаки 88
У нас проблема! 89
Неизбежность ошибок 90
Еще немного отладки 92
Типы данных Python 94
Исправление ошибки 95
Ура, заработало! 96
Этап 4: вывод результата 97

Булевы значения, условия и циклы

3 Принятие решений



Думаю, вы заметили, что наши предыдущие программы были не особо интересными. Содержащийся в них код представлял собой простую последовательность инструкций, выполняемых интерпретатором **сверху вниз**. Никаких тебе поворотов сюжета, внезапных изменений, сюрпризов — в общем, ничего оригинального. Чтобы программа могла стать более интересной, она нуждается в средствах **принятия решений**, позволяющих **управлять ходом ее выполнения** и многократно повторять одни и те же действия. Этим мы и займемся в данной главе. По ходу дела вы научитесь играть в загадочную игру шоушилин, познакомитесь с персонажем по имени Буль и увидите, насколько полезным может быть тип данных, содержащий всего два возможных значения.

Сыграем в игру? 110
Общий алгоритм игры 112
Компьютер делает свой выбор 113
Применение случайных чисел 114
Истина или ложь? 116
Булевы значения 117
Принимаем решения 118
Дополнительные условия 119
Вернемся к игре 120
Пользователь делает свой выбор 121
Проверяем, что выбрал пользователь 124
Код определения ничьей 125
Кто выиграл? 126
Реализация логики игры 128
Подробнее о булевых операторах 129
Вывод имени победителя 132
А где документация? 134
Добавление комментариев в код 135
Завершение игры 136
Проверка правильности введенных данных 137
Проверка и уточнение выражения 138
Повторный вывод запроса 140
Многократное повторение действий 141
Как работает цикл `while` 142
Как использовать цикл `while` для повторного вывода запроса 146
Итак, наша первая игра готова! 148

списки и циклы

4 Структуры данных

Числа, строки и булевы значения — не единственные типы данных в программировании. Пока что мы использовали только **примитивные типы**: числа с плавающей точкой, целые числа, строки и булевы значения (например, `3.14`, `42`, `"имя собаки"` и `True`). И хотя их возможности достаточно велики, время от времени в программах приходится иметь дело с большими наборами данных, такими как все добавленные в корзину покупки, названия всех известных звезд или содержимое всего каталога товаров. Для эффективной работы с подобными наборами нужны **структуры данных**. В этой главе вы познакомитесь с новым типом данных — **списком**, хранящим коллекцию значений. Списки позволяют структурировать данные, вместо того чтобы хранить каждый элемент в отдельной переменной. Вы узнаете о том, как работать со списком в целом и как поочередно перебирать его элементы в цикле `for`. Все это расширит ваши возможности по работе с данными.

Поможете разобраться? 162
Сохранение нескольких значений в одной переменной 163
Работа со списками 164
Размер списка 167
Извлечение последнего элемента списка 168
В Python все намного проще 168
Отрицательные индексы 169
Продолжим подсчет мыльных пузырьков 171
Просмотр элементов списка 174
Устранение ошибки вывода 175
Правильный способ вывода результата 176
Цикл `for` предпочтителен при работе со списками 178
Обработка диапазона чисел в цикле `for` 181
Другие операции с диапазонами 182
Составление отчета 184
Построение списка с нуля 192
Другие операции со списками 193
Тест-драйв готового отчета 197
И победителями становятся... 197
Тестирование самого выгодного раствора 201

Функции и абстракции

5 Функциональный код

Вы уже многому научились. Переменные и типы данных, условные конструкции и циклы — этих инструментов вполне достаточно, чтобы написать почти любую программу. Но зачем останавливаться на достигнутом? Следующий шаг — научиться **создавать абстракции** в коде. Звучит пугающе, хотя ничего страшного в этом нет. Абстракции — ваш спасательный круг. Они упрощают процесс разработки, давая возможность создавать более эффективные и функциональные программы. Абстракции позволяют упаковать код в небольшие блоки, удобные для повторного использования. С ними вы перестанете концентрироваться на низкоуровневых деталях и сможете программировать на высоком уровне.

Что не так с кодом? 217
Превращение блока кода в функцию 219
Функция создана — можно использовать 220
Как работает функция 220
Функции могут возвращать значения 228
Вызов функции, возвращающей значение 229
Улучшение имеющегося кода 231
Выполнение кода 232
Создание абстракций в коде выбора аватарки 233
Тело функции `get_attribute()` 234
Вызов функции `get_attribute()` 235
И снова о переменных 237
Область видимости переменной 238
Передача переменных в функцию 239
Вызов функции `drink_me()` 240
Использование глобальных переменных в функциях 243
Размышления о параметрах: значения по умолчанию и ключевые слова 246
Параметры по умолчанию 246
Первыми всегда указываются обязательные параметры 247
Аргументы с ключевыми словами 248
Как правильно использовать все доступные возможности 248

сортировка и вложенные списки

4 Наведение порядка в данных

(продолжение)

Иногда стандартный порядок сортировки данных нас не устраивает. Например, у вас есть список лучших результатов в аркадных играх, и вы хотите упорядочить его по названиям игр. Или же это может быть список сотрудников, регулярно опаздывающих на работу, и вы хотите узнать, кто из них проштрафился чаще остальных. Для решения подобных задач нужно знать, как сортировать данные и как самостоятельно настраивать порядок сортировки. В этой главе мы также изучим вложенные циклы и выясним, насколько эффективен написанный нами код.

Пузырьковый метод сортировки	264
Псевдокод алгоритма пузырьковой сортировки	267
Реализация пузырьковой сортировки на Python	270
Правильное определение номеров растворов	272

текст, строки и эвристические алгоритмы

6 Сведем все воедино

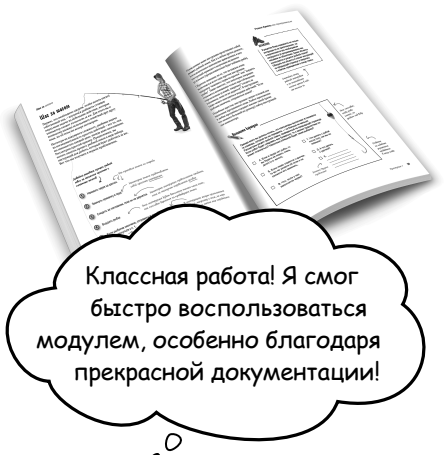
Вы уже многое знаете. Пора задействовать знания по максимуму. В этой главе мы сведем все воедино и начнем писать **сложный код**. В то же время мы продолжим пополнять арсенал знаний и навыков программирования. В частности, будет показано, как написать код, который **загружает текст**, обрабатывает его и выполняет **анализ текстовых данных**. Вы также узнаете, что такое **эвристический алгоритм** и как его реализовать. Приготовьтесь — вас ждет по-настоящему сложная и насыщенная глава.

Добро пожаловать в науку о данных	282
Как вычисляется индекс удобочитаемости	283
План действий	284
Общий псевдокод	285
Нужен текст для анализа	286
Создание основной функции	288
Подсчет числа слов в тексте	289
Подсчет общего числа предложений	293
Функция `count_sentences()`	294
Подсчет слогов: знакомство с эвристическими алгоритмами	300
Создание эвристического алгоритма	303
Начинаем кодировать эвристический алгоритм	304
Подсчет гласных	305
Игнорирование последовательных гласных	305
Код игнорирования последовательных гласных	306
Учет конечной ‘e’, а также знаков препинания	308
Создание срезов (подстрок)	310
Завершение эвристического алгоритма	312
Код вычисления индекса удобочитаемости	314
Дальнейшие улучшения	319

Классы, объекты, методы и модули

7 Модульное программирование

Написанные вами программы становятся все более сложными. Это заставляет тщательнее продумывать структуру и организацию программ. Вы уже знаете, что функции позволяют группировать строки кода в компактные блоки, удобные для повторного использования. Вы также знаете, что наборы функций и переменных можно объединять в модули, подключаемые к программам. В этой главе мы вернемся к теме модулей и рассмотрим, как работать с ними эффективнее (и даже делиться ими с другими пользователями). Кроме того, вас ждет знакомство с ключевыми строительными блоками программ — *объектами*. Вы увидите, что в Python они встречаются повсеместно, даже там, где вы не ожидаете их встретить.



Знакомство с модулями	330
Глобальная переменная `__name__`	332
Обновление файла *analyze.py*	333
Использование файла *analyze.py* в качестве модуля	335
Добавление документирующих строк в файл *analyze.py*	337
Знакомство с другими модулями Python	341
Что еще за черепашки?	342
Создание собственной черепашки	344
Изучение черепашек	345
Еще одна черепашка	347
Кто вы, черепашки?	350
Что такое объект?	351
А что тогда класс?	352
Использование объектов и классов	354
Что такое атрибуты и методы?	355
Объекты и классы повсюду	356
Черепашьи бега	358
Планирование игры	359
Приступаем к написанию кода игры	360
Код настройки игры	361
Спешка ни к чему!	362
Начало забега	364

Рекурсия и словари

8 И снова об индексах и циклах

Настало время перейти на новый уровень. Пока что мы придерживались итеративного стиля программирования, создавая структуры данных наподобие списков, строк и диапазонов чисел и обрабатывая их в циклах поэлементно. В этой главе мы пересмотрим подход к программированию и выбору структур данных. Наш новый стиль программирования будет требовать написания *рекурсивного*, т.е. вызывающего самого себя, кода. Кроме того, вы познакомитесь с новым типом данных — *словарем*, который больше напоминает ассоциативный массив, а не список. Воспользовавшись новыми знаниями, мы научимся эффективно решать множество задач. Заранее предупредим: это достаточно сложные темы. Пройдет какое-то время, прежде чем вы освоите все нюансы, но поверьте, оно того стоит.

Иной стиль программирования 378
Мы пойдем другим путем 379
Код для обоих случаев 380
Давайте попрактикуемся 383
Нахождение палиндромов рекурсивным способом 384
Код рекурсивной функции распознавания палиндромов 385
Антисоциальная сеть 396
Знакомство со словарем 398
Просмотр словаря 400
Словари в антисоциальной сети 402
Добавление новых атрибутов 404
Ключевая особенность антисоциальной сети 406
Определение самого антисоциального пользователя 407
Слово за вами! 408
Можно ли запомнить результаты работы функции? 412
Мемоизация 413
Анализ функции `koch()` 416
Фрактал Коха 418

Чтение и запись файлов

9 Длительное хранение данных

Вы умеете хранить данные в переменных, но как только программа завершается — бах! — они исчезают навсегда. Вот для чего нужно *постоянное* хранилище данных. Большинство компьютерных устройств оснащено такими хранилищами, в частности, жесткими дисками и флеш-накопителями. Кроме того, данные можно хранить в облачной службе. В этой главе вы узнаете, как писать программы, умеющие работать с файлами. Зачем это нужно? Вариантов множество: сохранение конфигурационных настроек, создание файлов отчетов, обработка графических файлов, поиск сообщений электронной почты — список можно продолжать еще долго.

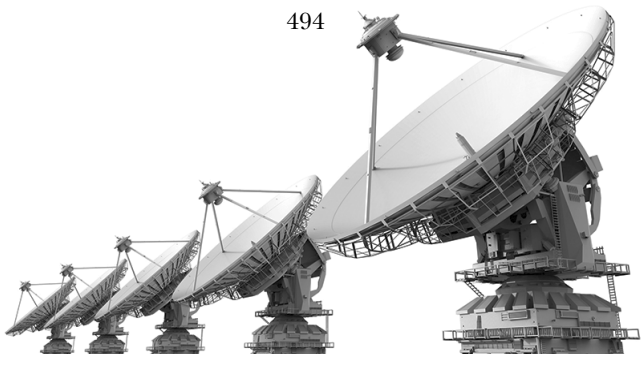
Игра в слова 430
Логика игры 432
Считывание истории из файла 435
Путь к файлу 436
Не забудьте все закрыть в конце 438
Чтение файла в коде 439
Пора прекратить... 442
Вернемся к игре 443
Как узнать, что мы достигли конца? 445
Чтение шаблона игры 446
Обработка текста шаблона 447
Исправление ошибки с помощью метода `strip()` 449
Окончательное исправление ошибки 450
Остались проблемы посерьезнее 451
Обработка исключений 453
Явная обработка исключений 454
Обработка исключений в игре в слова 456
Последний этап: сохранение результата 457
Обновление остального кода 457

Веб—службы

10 Так хочется большего

Вы научились писать неплохие программы, но всегда хочется большего. В Интернете хранится просто невероятное количество **данных**, которые так и ждут, чтобы с ними поработали. Нужен прогноз погоды? Как насчет базы данных рецептов? Или вас интересуют спортивные результаты? А может, ваш интерес — музыкальная база данных с информацией о музыкантах, альбомах и композициях? Всю эту информацию можно получить с помощью веб-служб. Для работы с ними нужно знать, как организуется передача данных в Интернете, а также познакомиться с парочкой модулей Python: `requests` и `json`. В этой главе вы изучите возможности **веб-служб** и повысите свой уровень знаний Python до заоблачных высот. Так что приготовьтесь к путешествию в космос.

Расширение возможностей благодаря веб-службам 472
Как работает веб-служба 473
Адрес веб-службы 474
Небольшое обновление Python 477
Установка обновления 478
Нам нужна интересная веб-служба 479
Возможности Open Notify 480
Передача данных в формате JSON 481
Вернемся к модулю `requests` 483
Собираем все вместе: создание запроса к службе Open Notify 485
Работа с данными JSON в Python 486
Применение модуля `json` 487
Представим результат в графическом виде 488
Знакомство с объектом `screen` 489
Черепашка-космонавт 491
Черепашка может выглядеть как космическая станция 492
Забудьте об МКС. Где находимся мы? 493
Завершение программы 494



Виджеты, события и непредсказуемое поведение

11 Интерактивные возможности

Вам уже приходилось писать простые графические приложения, но у них не было полноценного графического интерфейса. Другими словами, приложения не позволяли пользователям взаимодействовать с экранными элементами. Чтобы это стало возможным, необходимо применить иную модель выполнения программы, в рамках которой приложение **реагирует** на действия пользователя. Пользователь щелкнул на кнопке? Программа должна знать, как реагировать на такое событие, и быть готовым к нему. Процесс создания графического интерфейса сильно отличается от привычного процедурного стиля программирования, который мы применяли до сих пор. Нам придется совершенно по-иному подойти к решению задач. В этой главе вы создадите свой первый графический интерфейс, причем не какое-то примитивное экранное приложение, а нечто намного более интересное. Мы напишем игровой симулятор искусственной жизни с непредсказуемым поведением.

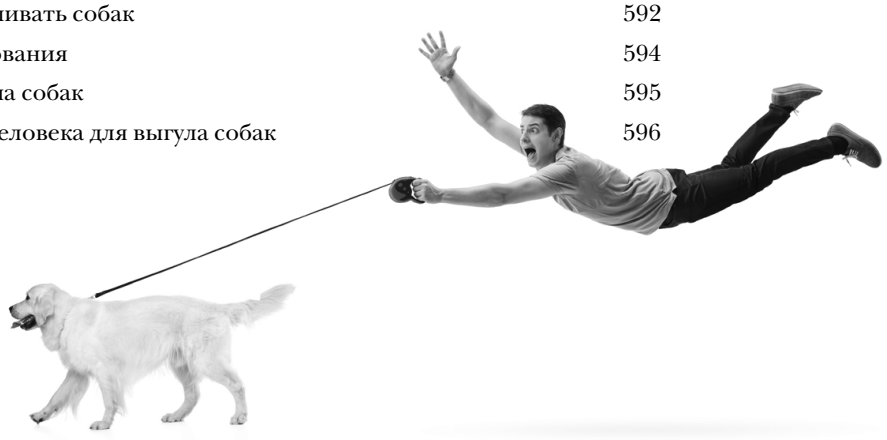
Откройте для себя удивительный мир игры “Искусственная жизнь” 504
Правила игры “Жизнь” 505
Что нам предстоит создать 508
Дизайнерские решения 509
Создание симулятора игры “Жизнь” 512
Построение модели данных 513
Вычисление поколений клеток 514
Завершение кода модели 518
Экранное представление 521
Создание первого виджета 522
Добавление остальных виджетов 523
Исправление макета 524
Размещение виджетов по сетке 525
Переходим к написанию контроллера 527
Новый способ вычислений 530
Обработка щелчков мыши 531
Обработка событий, связанных с кнопкой Start/Pause 533
Реализация кнопки Start/Pause 534
Другой тип событий 535
А как же метод `after()`? 537
Непосредственное редактирование клеток 540
Обработчик `grid_handler()` 541
Добавление шаблонов 542
Обработчик событий для объекта `OptionMenu` 543
Создание собственных фигур 545
Загрузчик шаблонов 546
Улучшаем симулятор игры “Жизнь” 553

объектно-ориентированное программирование

12 Путешествие в страну объектов

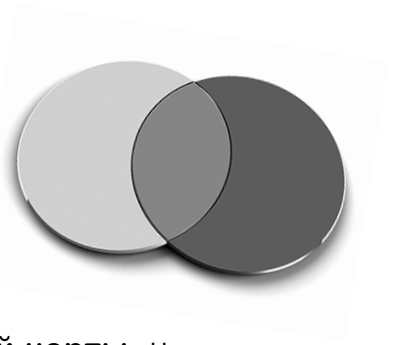
В предыдущих главах для создания абстракций в коде применялись функции. Мы всячески придерживались **процедурного** подхода, используя в функциях простые выражения, условные конструкции и циклы `for`/`while` — ничего такого, что относилось бы к **объектно-ориентированному** программированию. Конечно, вы познакомились с объектами и научились применять их, но вам еще ни разу не приходилось создавать классы и писать полностью объектно-ориентированный код. Пришло время попрощаться со скучным процедурным прошлым. Нас ожидает новый мир, полный объектов и надежд на лучшую жизнь.

Другой подход к структурированию программ 560
Преимущества объектно-ориентированного программирования 561
Разработка своего первого класса 563
Написание кода класса 564
Как работает конструктор 564
Метод `bark()` 567
Как работает метод 568
Немного о наследовании 570
Реализация класса `ServiceDog` 571
Подклассы 572
`ServiceDog` — это потомок класса `Dog` 573
Проверка принадлежности к классу 574
Переопределение и расширение поведения 578
Учим термины 580
Объекты внутри объектов 582
Создание отеля для собак 585
Реализация отеля для собак 586
Усовершенствование отеля для собак 589
Расширение функций отеля 590
Мы с тобой одной крови: полиморфизм 591
Учимся выгуливать собак 592
Сила наследования 594
Служба выгула собак 595
Как нанять человека для выгула собак 596



приложение: не попавшее в избранное

Десять ключевых тем (которые не были рассмотрены)



Вы прошли длинный путь и находитесь у финишной черты. Нам искренне жаль расставаться с вами, и, прежде чем отправить вас в самостоятельное плавание, нам бы хотелось дать последние наставления. Конечно, в одном маленьком приложении невозможно охватить все необходимое, хотя мы попытались это сделать, уменьшив размер шрифта до 0,00004 пункта. К сожалению, никому в издательстве так и не удалось прочитать столь микроскопический текст, поэтому нам пришлось выкинуть все лишнее, оставив только самое важное.

1. Списковые включения 612
2. Работа с датой и временем 613
3. Регулярные выражения 614
4. Другие типы данных: кортежи 615
5. Другие типы данных: множества 616
6. Написание серверного кода 617
7. Отложенные вычисления 618
8. Декораторы 619
9. Объекты первого класса и функции высшего порядка 620
10. Важные библиотеки 621

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: `info@dialektika.com`

WWW: `http://www.dialektika.com`

4 (продолжение) сортировка и вложенные циклы

вернемся к спискам и добавим ряд суперспособностей

Наведение порядка в данных

Иногда стандартный порядок сортировки данных нас не устраивает. Например, у вас есть список лучших результатов в аркадных играх, и вы хотите упорядочить его по названиям игр. Или же это может быть список сотрудников, регулярно опаздывающих на работу, и вы хотите узнать, кто из них проштрафился чаще остальных. Для решения подобных задач нужно знать, как сортировать данные и как самостоятельно настраивать порядок сортировки. В этой главе мы также изучим вложенные циклы и выясним, насколько эффективен написанный нами код. Нам предстоит существенно улучшить навыки вычислительного мышления!

Это снова я! Исследование пузырьковых растворов, проведенное в главе 4, настолько впечатлило меня, что я готов выдать специальную премию. Но мне нужно, чтобы вы написали код, генерирующий еще один отчет. Думаю, для вас это теперь проще простого.

Пузырь-ОК

Мне нужно еще кое-что. Я с удовольствием выдам премии изобретателям лучших растворов. Можете составить для меня список пяти лучших растворов в порядке убывания результата? Пример показан ниже.

— генеральный директор компании "Пузырь-ОК"

```
Лучшие пузырьковые растворы
1. Пузырьковый раствор #10 - результат: 68
2. Пузырьковый раствор #12 - результат: 60
3. Пузырьковый раствор #2 - результат: 57
4. Пузырьковый раствор #31 - результат: 50
5. Пузырьковый раствор #3 - результат: 34
```

Учтите, что это не настоящие результаты. Это лишь иллюстрация того, что мне нужно.

Спасибо!

Офисный диалог

Федор. Мы обязаны справиться, но как все это реализовать?

Юлия. Мы уже столько алгоритмов придумали, должны и сортировку побороть.

Игорь. Некоторые ученые всю жизнь посвятили разработке алгоритмов сортировки, а ты говоришь "побороть". Придется изучить имеющиеся алгоритмы и выбрать самый походящий для нашей задачи.

Федор. Ну, это надолго! Можно я сначала перекушу?

Игорь. Давай не будем терять время. Я уже все разузнал.

Федор. Что же ты сразу не сказал? И какой метод сортировки мы применим?

Игорь. Пузырьковый.

Юлия [гневно]. Очень смешно! Прямо открыл нам Америку. Мы и так знаем, что у нас пузырьковые тесты. А с сортировкой то что?

Игорь. Я, вообще-то, совершенно серьезен. Мы будем использовать пузырьковый метод сортировки. Он не самый эффективный, зато один из самых простых для понимания.

Федор. Что за странное название? Кто его придумал?

Игорь. Его так назвали, потому что в процессе сортировки элементы перемещаются к началу списка, или, как говорят, "всплывают", словно пузырьки. Ты сам все поймешь, когда мы приступим к реализации.

Юлия. Похоже, ты хорошо подготовился. Что ж, бери на себя руководство проектом.

Игорь. Я готов, давайте начинать.

Юлия. Чуть не забыла, я попросила Григория присоединиться к проекту. Я ведь не знала, что ты уже все продумал. Придется сказать ему, что мы все сделали без него. Напомните мне, если забуду, хорошо?

Пузырьковый метод сортировки

К написанию псевдокода пузырькового метода сортировки мы перейдем чуть позже, а пока давайте познакомимся с самим алгоритмом. Для этого попробуем отсортировать следующий список чисел:

```
[6, 2, 5, 3, 9]
```

Не отсортированный список

В общем случае, говоря о сортировке, подразумевают упорядочение чисел по возрастанию. Таким образом, ожидается, что после сортировки список будет приведен к такому виду:

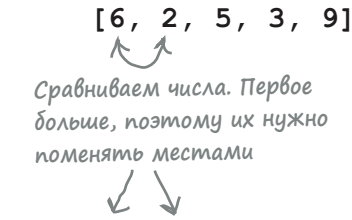
```
[2, 3, 5, 6, 9]
```

Этот же список, отсортированный по возрастанию

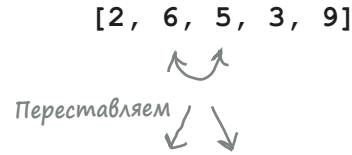
Сразу подчеркнем, что пузырьковый метод сортировки подразумевает упорядочение элементов списка в несколько проходов. Как вы вскоре увидите, если текущий проход завершается перестановкой значений, то нужно совершить еще один проход. Если же на текущем проходе перестановок не было, значит, сортировка завершена.

Начало: первый проход

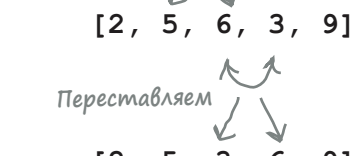
Мы начинаем со сравнения первого и второго элементов (имеющих индексы 0 и 1). Если первый элемент больше второго, меняем их местами.



Далее сравниваем элементы с индексами 1 и 2. Если первый из них (6) больше второго (5), тоже меняем их местами.

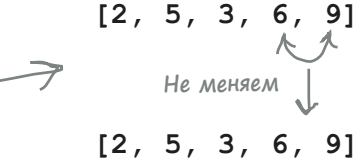


Переходим к сравнению элементов с индексами 2 и 3. И здесь первый из элементов оказывается больше второго. Переставляем их.



Теперь нужно сравнить элемент номер 3 с элементом номер 4. В данном случае большим оказывается второй элемент, поэтому порядок элементов не меняется.

Заметьте, как число 6, которое было первым в списке, постепенно перемещается в конец списка



На этом первый проход завершен, но мы выполнили несколько перестановок, а значит, необходим еще один проход. Переходим ко второму этапу.

Второй проход

На втором проходе список просматривается повторно с самого начала. Вначале сравниваем элементы с индексами 0 и 1. Второй из них больше, поэтому перестановка не происходит.

Далее сравниваем элементы с индексами 1 и 2. В данном случае элемент с индексом 2 больше элемента с индексом 1, поэтому переставляем их местами.

Вам уже должно быть понятно, что будет дальше. Сравниваем элементы с индексами 2 и 3 — поскольку 5 меньше, чем 6, оставляем элементы в исходных позициях.

Продолжаем, сравнивая элементы с индексами 3 и 4. Значение элемента с индексом 4 больше, следовательно, перестановка не происходит.

Второй проход завершен, но поскольку имела место перестановка (элементы с индексами 1 и 2 менялись местами), нужно выполнить еще один проход.

Третий проход

Как и ранее, повторно просматриваем список, начиная с первого элемента. Вначале сравним элементы с индексами 0 и 1. Первое значение меньше второго, поэтому элементы не переставляются.

Далее сравниваем элементы с индексами 1 и 2. В данном случае первый из них меньше второго, поэтому и тут переставлять элементы не нужно.

Переходим к сравнению элементов с индексами 2 и 3. Второе значение больше — оставляем оба элемента в исходных позициях.

Аналогичным образом сравним элементы с индексами 3 и 4. Второе значение больше, поэтому элементы остаются на прежних местах.

На третьем проходе мы не поменяли местами ни одну пару элементов, следовательно, сортировка списка завершена!

Возьмите карандаш

Теперь, когда вы получили представление о пузырьковом методе сортировки, давайте применим его для выполнения практического задания, чтобы закрепить приобретенные навыки. Отсортируйте приведенный ниже список так, как это делалось на двух предыдущих страницах. Если на каком-то этапе запутаетесь, сверьтесь с ответом в конце главы.

Мы не просим вас писать код. Мы лишь просим применить пузырьковый метод для сортировки этого списка

Вот наш список. Не пугайтесь строк, просто сравнивайте их в алфавитном порядке

```
['клубничный', 'банановый', 'ананасовый', 'ягодный']
```

Проход 1

Начните здесь первый проход

Псевдокод алгоритма пузырьковой сортировки

Познакомившись с пузырьковым методом сортировки, давайте опишем алгоритм его работы в виде псевдокода. Вы уже знаете все, что для этого нужно, поскольку мы задействуем лишь простые булевы операции и циклы. Однако циклы применяются здесь ранее не встречавшимся способом — один внутри другого. Такая конструкция называется *вложенным циклом*.

При первом знакомстве вложенные циклы могут несколько обескуражить, но не стоит отчаиваться, ведь вам уже доводилось иметь с ними дело при сортировке списка чисел и выполнении предыдущего упражнения. Внешний цикл представляет отдельные проходы пузырькового метода. А во внутреннем цикле выполняется сравнение (и, если нужно, перестановка) соседних элементов списка на каждом проходе. Принимая это во внимание, алгоритм пузырькового метода можно описать следующим образом.

Вспомним все, что мы узнали из главы 5 о функциях!

Мы хотим написать функцию `bubble_sort()`, которая получает список в качестве параметра

```
DEFINE функция bubble_sort(list):
    DECLARE переменная swapped и присвоить ей True

    WHILE swapped:
        SET swapped равно False
        FOR переменная i in range(0, len(list)-1)
            IF list[i] > list[i+1]:
                DECLARE переменная temp и присвоить ей list[i]
                SET list[i] равно list[i+1]
                SET list[i+1] равно temp
                SET swapped равно True
```

Нам нужна переменная для отслеживания перестановок на текущем проходе. Изначально устанавливаем ее равной True, чтобы сделать первый проход

Мы используем цикл `while`, который выполняется, пока переменная `swapped` равна True

На каждой итерации цикла `while` выполняется новый проход сортировки

Первое, что мы делаем в цикле, — задаем переменную `swapped` равной False

В цикле `for` мы проходим каждый элемент списка (кроме последнего) и выполняем сравнение. Если он больше следующего элемента, то меняем их местами

Псевдокод бывает разным. Здесь мы имитируем код, но это все равно не код, по крайней мере не Python

Если были перестановки, задаем переменную `swapped` равной True. Это означает, что по завершении цикла `for` будет выполнен еще один проход

В приведенном выше псевдокоде список сортируется по возрастанию. Что нужно поменять в случае сортировки по убыванию?

Офисный диалог продолжается...

Юлия. Думаю, да, если я правильно понимаю работу обоих циклов.

Игорь. Тут все просто: имеем цикл `while` и вложенный в него цикл `for`.

Юлия. Внешний цикл `while` выполняется до тех пор, пока переменная `swapped` равна `True`.

Игорь. Да, каждая его итерация соответствует одному проходу по элементам сортируемого списка.

Юлия. По-моему, во всех наших примерах это всегда происходило по три раза.

Игорь. Это чистое совпадение: алгоритм допускает произвольное количество проходов. В худшем случае оно будет равно числу элементов списка.

Юлия. То есть, если список состоит из 100 элементов, то его сортировка может длиться 100 проходов?

Игорь. Да, именно так!

Юлия. Не многовато ли?

Игорь. Ну, это же худший случай, когда исходный список полностью отсортирован по убыванию.

Юлия. Понятно, а что насчет внутреннего цикла `for`? Что происходит в нем?

Игорь. В этом цикле каждый элемент списка сравнивается со следующим элементом. Если текущий элемент больше следующего, то они меняются местами.

Юлия. То есть в этом цикле число итераций равно количеству элементов в списке, причем не в худшем случае, а всегда?

Игорь. Технически число итераций равно количеству элементов списка минус один, а в остальном ты права.

Юлия. Для больших списков получаем что-то совсем уж много итераций.

Игорь. Ну да. Если список содержит 100 элементов, то в худшем случае получаем 100 проходов по 100 сравнений в каждом, итого 100 * 100 = 10 000 сравнений.

Юлия. Ого!

Игорь. Что поделать, пузырьковый метод славится своей простотой, но не эффективностью. Почему, по-твоему, столько людей до сих пор занимаются разработкой быстрых алгоритмов сортировки? К счастью, наши списки очень короткие, поэтому пузырьковый метод вполне для них подходит.

Юлия. В цикле `for` происходит еще кое-что: если имела место перестановка, переменная `swapped` устанавливается равной `True`, что означает необходимость еще одного прохода.

Игорь. Абсолютно верно!

Учимся понимать

Возьмите на себя роль интерпретатора Python и мысленно выполните приведенные ниже фрагменты кода, попытавшись предугадать их результаты. Завершив упражнение, сверьтесь с ответом в конце главы.

```
for i in range(0, 4):
    for j in range(0, 4):
        print(i * j)
```

```
for word in ['як', 'кот', 'пума', 'ягуар', 'собака']:
    for i in range(2, 7):
        letters = len(word)
        if (letters % i) == 0:
            print(i, word)
```

Вспомните, что операция деления по модулю означает нахождение остатка от деления. Например, 4 % 2 = 0, а 4 % 3 = 1

```
full = False

donations = []
full_load = 45

toys = ['робот', 'кукла', 'мяч', 'волчок']

while not full:
    for toy in toys:
        donations.append(toy)
        size = len(donations)
        if (size >= full_load):
            full = True

print('Полон', len(donations), 'игрушек')
print(donations)
```

Реализация пузырьковой сортировки на Python

Наш псевдокод близок к реальному коду, поэтому перевести его на Python не составляет труда. Вот что у нас получилось.

Это наша функция, аргументом которой служит список

```
def bubble_sort(scores):
    swapped = True
    while swapped:
        swapped = False
        for i in range(0, len(scores)-1):
            if scores[i] > scores[i+1]:
                temp = scores[i]
                scores[i] = scores[i+1]
                scores[i+1] = temp
                swapped = True
```

Мы устанавливаем переменную `swapped` *равной* `True`*, чтобы сделать первый проход*

Цикл `while` *выполняется, пока переменная* `swapped` *равна* `True`*. Мы просматриваем весь список, при необходимости делая перестановки*

Вложение циклов: цикл `for` *внутри цикла* `while`

Именно так мы выполняли перестановку переменных в главе 2!

Функция ничего не возвращает, потому что мы выполнили перестановку фактических значений списка. Другими словами, мы отсортировали оригинальный список

Тест-драйв

Создайте новый файл *sort.py*, скопируйте в него приведенный выше код и добавьте в конец следующие инструкции для тестирования.

```
scores = [60, 50, 60, 58, 54, 54,
          58, 50, 52, 54, 48, 69,
          34, 55, 51, 52, 44, 51,
          69, 64, 66, 55, 52, 61,
          46, 31, 57, 52, 44, 18,
          41, 53, 55, 61, 51, 44]

bubble_sort(scores)
print(scores)

smoothies = ['кокосовый', 'клубничный', 'банановый', 'ананасовый']
bubble_sort(smoothies)
print(smoothies)
```

Оператор `>` *в Python работает и со строками, что позволяет отсортировать список строк*

```
Оболочка Python
[18, 31, 34, 41, 44, 44, 44, 46, 48, 50,
50, 51, 51, 51, 52, 52, 52, 52, 53, 54,
54, 54, 55, 55, 55, 57, 58, 58, 60, 60,
61, 61, 64, 66, 69, 69]
['ананасовый', 'банановый', 'клубничный',
'ягодный']
>>>
```

Все отсортировано!

Тест-драйв

Нам необходимо отсортировать список результатов так, чтобы вначале шли растворы с наилучшими результатами. Другими словами, список должен быть отсортирован по убыванию, а не по возрастанию. Для этого достаточно поменять оператор сравнения в функции сортировки с > на <. Внесите изменение в файл и проверьте получаемый результат.

```
def bubble_sort(scores):
    swapped = True

    while swapped:
        swapped = False
        for i in range(0, len(scores)-1):
            if scores[i] < scores[i+1]:
                temp = scores[i]
                scores[i] = scores[i+1]
                scores[i+1] = temp
                swapped = True
```

Это все, что нужно изменить для сортировки списка по убыванию

Вот результаты: списки отсортированы по убыванию

Оболочка Python

```
[69, 69, 66, 64, 61, 61, 60, 60, 58, 58,
57, 55, 55, 55, 54, 54, 54, 53, 52, 52,
52, 52, 51, 51, 51, 50, 50, 48, 46, 44,
44, 44, 41, 34, 31, 18]
['ягодный', 'клубничный', 'банановый',
'ананасовый']
>>>
```

Мы почти справились. Осталось сгенерировать отчет, содержащий 5 лучших растворов с номерами и результатами.

Федор. Игорь молодец, написал весь код сортировки! Но кое-чего не хватает. Мы сортируем список результатов, однако не запоминаем номера исходных растворов. Как же мы узнаем номера растворов-победителей? Их ведь нужно включить в отчет.

Юлия. Согласна. Что же нам делать?

Федор. Игорь, конечно, крутой программист, но я могу улучшить его код. Что, если мы создадим еще один, параллельный, список `solutions_numbers`, элементы которого равны своим индексам, например `[0, 1, 2, 3, ..., 35]`? Тогда при сортировке списка результатов мы аналогичным образом отсортируем и список индексов. В итоге каждый индекс будет находиться в той же позиции, что и соответствующий ему результат.

Юлия. Но как создать список, элементы которого равны своим индексам?

Федор. Вспомни, для этого есть функции `range()` и `list()`.

```
number_of_scores = len(scores)
solution_numbers = list(range(number_of_scores))
```

Определяем длину списка

Создаем диапазон от 0 до длины списка (минус 1) и используем функцию `list()` для преобразования диапазона в список [0, 1, 2, ...]

Юлия. Интересно! Я, кажется, начинаю понимать ход твоих мыслей.

Федор. Некоторые вещи проще показать, чем объяснить. Давай взглянем на готовый код...

Правильное определение номеров растворов

Федор оказался прав. Мы сортируем список результатов, но при этом теряем информацию об их исходных позициях, по которым мы сопоставляем результаты с растворами. Решение заключается в том, чтобы создать второй список, который будет содержать только номера растворов. При сортировке списка результатов мы будем синхронно сортировать список номеров в той же последовательности. Рассмотрим программный код.

По-серьезному

Какой результат возвращает приведенное ниже выражение? Анализируйте его от внутренних скобок к внешним.

```
list(range(number_of_scores))
```

Равно целочисленной длине списка результатов: 36

Возвращает диапазон от 0 до 35

Возвращает список, содержащий числа от 0 до 35

Теперь мы передаем функции `bubble_sort()` два списка: список результатов и список соответствующих номеров растворов

```
def bubble_sort(scores, numbers):
    swapped = True

    while swapped:
        swapped = False
        for i in range(0, len(scores)-1):
            if scores[i] < scores[i+1]:
                temp = scores[i]
                scores[i] = scores[i+1]
                scores[i+1] = temp
                temp = numbers[i]
                numbers[i] = numbers[i+1]
                numbers[i+1] = temp
                swapped = True

scores = [60, 50, 60, 58, 54, 54,
          58, 50, 52, 54, 48, 69,
          34, 55, 51, 52, 44, 51,
          69, 64, 66, 55, 52, 61,
          46, 31, 57, 52, 44, 18,
          41, 53, 55, 61, 51, 44]

number_of_scores = len(scores)
solution_numbers = list(range(number_of_scores))

bubble_sort(scores, solution_numbers)
```

Все остальное работает так же, как и раньше...

...за исключением того, что после перестановки значений в списке результатов мы также переставляем аналогичные значения в списке номеров

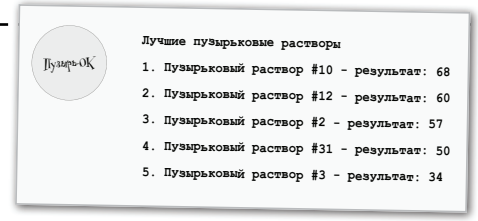
Если вы считаете это избыточным кодом, то вы правы. В следующей главе вы узнаете, как убрать дублирующийся код

Здесь мы создаем список `solution_numbers`, хранящий номера всех растворов (они соответствуют исходным индексам в списке результатов)

Передаем оба списка в функцию сортировки

Возьмите карандаш

Теперь в программе создаются сразу два списка: один хранит результаты исследований, упорядоченные по убыванию, а второй — соответствующие им номера растворов. Напишите код вывода отчета на основе этих двух списков.



Наш отчет должен выглядеть так. И не забывайте о том, что это не фактические данные, выдаваемые отчетом, а всего лишь пример от директора

Тест-драйв

Добавьте код из предыдущего упражнения (приведен ниже) в файл *sort.py*, заменив ненужные инструкции тестирования. Проверьте работу программы.

```
print('Лучшие пузырьковые растворы')
for i in range(0, 5):
    print(str(i+1) + '.',
        'Пузырьковый раствор #' + str(solution_numbers[i]),
         '- результат:', scores[i])
```

Именно так, как и хотел генеральный директор. Он будет счастлив!

Оболочка Python

```
Лучшие пузырьковые растворы
1. Пузырьковый раствор #11 - результат: 69
2. Пузырьковый раствор #18 - результат: 69
3. Пузырьковый раствор #20 - результат: 66
4. Пузырьковый раствор #19 - результат: 64
5. Пузырьковый раствор #23 - результат: 61
>>>
```

Я выдам премию не только изобретателям пяти лучших пузырьковых растворов, но и вам! Компания "Пузырь-ОК" не смогла бы стать настолько успешной без вашего блестящего кода.

Отличная работа!

Это была короткая, но непростая глава. Вам пришлось изучить немало новых концепций, на осмысление которых потребуется время. А пока наслаждайтесь заслуженным призом. Вы хорошо потрудились и вправе немного отдохнуть.

Глава, впрочем, еще не закончена. Нам предстоит подвести итоги и не только...

САМОЕ ГЛАВНОЕ

- Существует много алгоритмов сортировки, обладающих разной сложностью и эффективностью.
- Пузырьковая сортировка — это простейший алгоритм, в котором элементы списка сравниваются и переставляются проход за проходом.
- Пузырьковая сортировка завершается, если на очередном проходе не было сделано ни одной перестановки.
- В большинстве языков программирования есть встроенные функции сортировки.
- Цикл, находящийся внутри другого цикла, называется вложенным.
- Вложенные циклы усложняют структуру программы и могут увеличивать время ее выполнения.
- Полезно изучать алгоритмы сортировки, особенно те, которые включены в стандартную библиотеку языка программирования. Ах да, вы ведь еще не в курсе...

Встроенные средства сортировки есть в большинстве языков программирования.

Да, во многих современных языках программирования имеются функции сортировки. Поэтому чаще всего нет никакой необходимости писать собственный код сортировки. Во-первых, незачем изобретать колесо, а во-вторых, встроенные функции работают более эффективно, чем пузырьковая сортировка, и выполняются значительно быстрее. На самостоятельную реализацию таких функций у вас уйдет слишком много времени. Так почему бы не воспользоваться готовыми решениями?

Но не подумайте, что вы зря читали эту главу. Приемы, которые вы освоили в ходе изучения пузырьковой сортировки, в частности, вложенные списки, пригодятся вам при реализации многих алгоритмов. К тому же пузырьковая сортировка — то, с чего все начинают.

А что касается Python, то для сортировки списка достаточно вызвать одну функцию:

```
scores.sort()
```

Есть много способов настроить порядок сортировки, но это более сложная тема.

Если вас заинтересовала тема сортировки, советуем изучить различные алгоритмы, их преимущества и недостатки. Каждый алгоритм оценивается по скорости выполнения и эффективности использования памяти. Среди наиболее известных алгоритмов — сортировка вставками, сортировка слиянием, быстрая сортировка и многие другие. В Python реализован гибридный алгоритм Timsort, сочетающий сортировку вставками и сортировку слиянием. Обо всем этом можно прочитать в Википедии (`https://ru.wikipedia.org/wiki/Алгоритм_сортировки`).

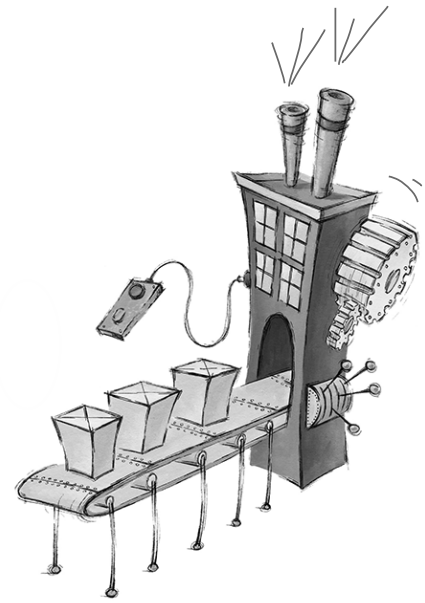
На самый конец у нас припасена небольшая головоломка.

Супержелезяка^2

Супержелезяка — это хитроумная штуковина, которая стучит, гремит и даже гудит. Ей не важно, что перемалывать: хоть списки, хоть строки. Но как она работает? А вы сможете разобраться в ее коде?

```
characters = 'тако'

output = ''
length = len(characters)
i = 0
while (i < length):
    output = output + characters[i]
    i = i + 1

length = length * -1
i = -2

while (i >= length):
    output = output + characters[i]
    i = i - 1

print(output)
```

Все, что мы поменяли,— это список, который теперь стал строкой. Но код работает, как и раньше. Почему?



Попробуйте альтернативные варианты строки символов:

characters = 'amanaplanac'

или

characters = 'wasitar'

Почему код работает и со списками, и со строками? На этот непростой вопрос мы частично дадим ответ в главе 6 (и продолжим в последующих главах)

Возьмите карандаш
Решение

Теперь, когда вы получили представление о пузырьковом методе сортировки, давайте применим его для выполнения практического задания, чтобы закрепить приобретенные навыки. Отсортируйте приведенный ниже список так, как это делалось на двух предыдущих страницах.

```
['клубничный', 'банановый', 'ананасовый', 'ягодный']
```

Вот наш список. Не пугайтесь строк, просто сравнивайте их в алфавитном порядке

Проход 1

```
['клубничный', 'банановый', 'ананасовый', 'ягодный']
```
Переставляем
```
['банановый', 'клубничный', 'ананасовый', 'ягодный']
```
Переставляем
```
['банановый', 'ананасовый', 'клубничный', 'ягодный']
```
Не меняем
```
['банановый', 'ананасовый', 'клубничный', 'ягодный']
```

Сравниваем каждый элемент со следующим, проходя по списку. Делаем перестановку, если первое значение больше второго

На первом проходе были перестановки, поэтому нужен второй проход

Проход 2

```
['банановый', 'ананасовый', 'клубничный', 'ягодный']
```
Переставляем
```
['ананасовый', 'банановый', 'клубничный', 'ягодный']
```
Не меняем
```
['ананасовый', 'банановый', 'клубничный', 'ягодный']
```
Не меняем
```
['ананасовый', 'банановый', 'клубничный', 'ягодный']
```

На втором проходе были перестановки, поэтому нужен третий проход

Проход 3

```
['ананасовый', 'банановый', 'клубничный', 'ягодный']
```
Не меняем
```
['ананасовый', 'банановый', 'клубничный', 'ягодный']
```
Не меняем
```
['ананасовый', 'банановый', 'клубничный', 'ягодный']
```
Не меняем
```
['ананасовый', 'банановый', 'клубничный', 'ягодный']
```

На третьем проходе не было перестановок, так что мы закончили, и список отсортирован

Учимся понимать

Решение

Возьмите на себя роль интерпретатора Python и мысленно выполните приведенные ниже фрагменты кода, попытавшись предугадать их результаты. Завершив упражнение, сверьтесь с ответом в конце главы.

```
for i in range(0, 4):
    for j in range(0, 4):
        print(i * j)
```

Оболочка Python

```
0
0
0
0
0
1
2
3
0
2
4
6
0
3
6
9
>>>
```

```
for word in ['як', 'кот', 'пума', 'ягуар', 'собака']:
    for i in range(2, 7):
        letters = len(word)
        if (letters % i) == 0:
            print(i, word)
```

Оболочка Python

```
2 як
3 кот
2 пума
4 пума
5 ягуар
2 собака
3 собака
6 собака
>>>
```

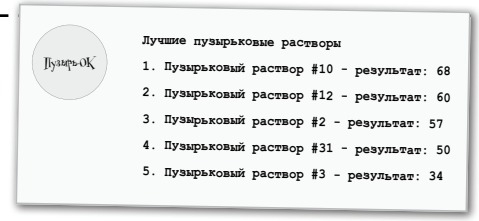
```
full = False

donations = []
full_load = 45

toys = ['робот', 'кукла', 'мяч', 'волчок']

while not full:
    for toy in toys:
        donations.append(toy)
        size = len(donations)
        if (size >= full_load):
            full = True

print('Полон', len(donations), 'игрушек')
print(donations)
```

Оболочка Python

```
Полон 48 игрушек
['робот', 'кукла', 'мяч', 'волчок', 'робот', 'кукла',
'мяч', 'волчок', 'робот', 'кукла', 'мяч', 'волчок',
'робот', 'кукла', 'мяч', 'волчок', 'робот', 'кукла',
'мяч', 'волчок', 'робот', 'кукла', 'мяч', 'волчок',
'робот', 'кукла', 'мяч', 'волчок', 'робот', 'кукла',
'мяч', 'волчок', 'робот', 'кукла', 'мяч', 'волчок',
'робот', 'кукла', 'мяч', 'волчок', 'робот', 'кукла', 'мяч',
'волчок', 'робот', 'кукла', 'мяч', 'волчок']
>>>
```

Возьмите карандаш
Решение

Теперь в программе создаются сразу два списка: один хранит результаты исследований, упорядоченные по убыванию, а второй — соответствующие им номера растворов. Напишите код вывода отчета на основе этих двух списков.



Выводим заголовок

```
print('Лучшие пузырьковые растворы')
for i in range(0, 5):
    print(str(i+1) + '.',
          'Пузырьковый раствор #' + str(solution_numbers[i]),
          '- результат:', scores[i])
```

Выполняем пять итераций для вывода пяти наибольших результатов

В каждой строке вывода мы отображаем итоговую позицию раствора (значение `i+1`), номер раствора из списка `solution_numbers` и результат из списка `scores`