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:

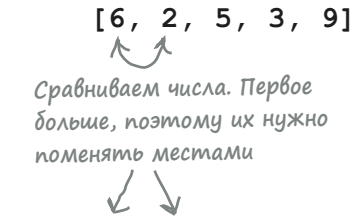
```
[2, 3, 5, 6, 9]
```

Этот же список, отсортированный по возрастанию

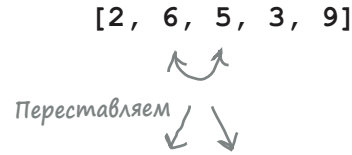
Сразу подчеркнем, что пузырьковый метод сортировки подразумевает упорядочение элементов списка в несколько проходов. Как вы вскоре увидите, если текущий проход завершается перестановкой значений, то нужно совершить еще один проход. Если же на текущем проходе перестановок не было, значит, сортировка завершена.

Начало: первый проход

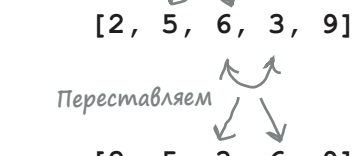
Мы начинаем со сравнения первого и второго элементов (имеющих индексы 0 и 1). Если первый элемент больше второго, меняем их местами.



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

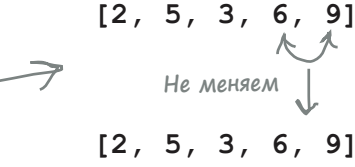


Переходим к сравнению элементов с индексами 2 и 3. И здесь первый из элементов оказывается больше второго. Переставляем их.



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

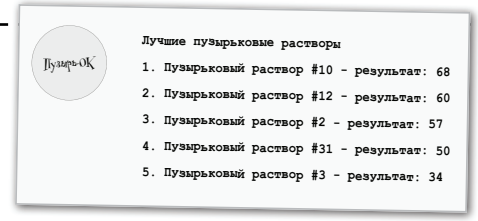
Если вы считаете это избыточным кодом, то вы правы. В следующей главе вы узнаете, как убрать дублирующийся код

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`).

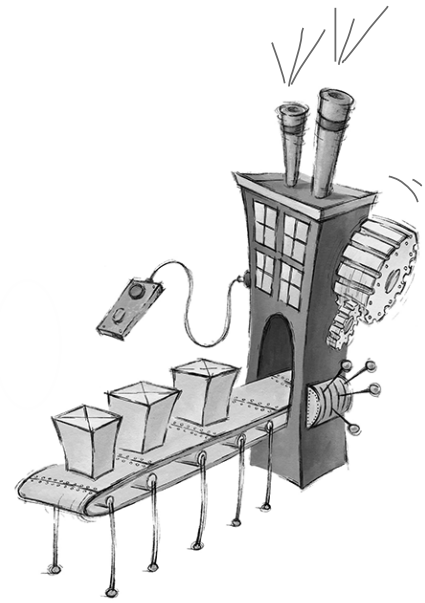
На самый конец у нас припасена небольшая головоломка.

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
>>>
```

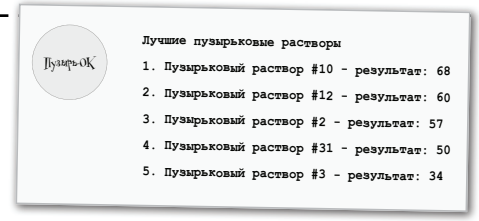
```
full = False

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`