Работа с простыми таблицами (csv). Фильтрация, сортировка, вывод

План урока

Аннотация

Урок посвящён технологии хранения однотипных записей в файлах, форматам с фиксированной и с произвольной длиной записи (DSV, TSV, CSV). Особое внимание уделено форматам с произвольной длиной записи, методах работы с ними при помощи строковых функций, а также специализированной библиотеки csv.

1. Хранение записей в файлах

Любой файл может содержать последовательность байт, которая интерпретируется человеком и прикладными программами различным образом. Например, байт с десятичным значением 65 может означать и число 65, и код латинской буквы A в зависимости от договорённости по формату. Очень быстро стало нужно хранить в файлах массивы однородных объектов (записей), имеющих некоторый набор полей или характеристик. По сути, такой массив представляет из себя таблицу со строками и столбцами. Каждый столбец имеет свой, как правило примитивный, тип: целое или вещественное число, строка и т.д.

Пример таких структур — информация об учениках класса с указанием фамилии, имени, даты рождения, роста, среднего балла и т.д.

Очень важно, что формат при этом может быть совершенно различным. Главное, чтобы все программы, которые с ним работают, были написаны в соответствии со спецификацией этого формата, то есть **понимали** его.

В процессе эволюции из всех форматов для хранения массивов записей **прижились** два: с фиксированной и произвольной длиной записи. А подлинного искусства обработка и хранение таблиц достигли в реляционных базах данных, которые мы не рассматриваем на этой лекции.

2. Форматы с фиксированной длиной записи

Такие форматы практически всегда имеют байтовую, а не текстовую структуру.

Размер каждого поля фиксируется таким образом, чтобы иметь минимальную, но достаточную длину в байтах для представления всего диапазона значений. Как правило, на этом этапе возникают проблемы с выбором длины строковых данных.

Для примера рассмотрим расшифровку записи для хранения информации о школьнике:

— Имя: строка(50)

— Фамилия: строка(50)

— Возраст: однобайтовое число

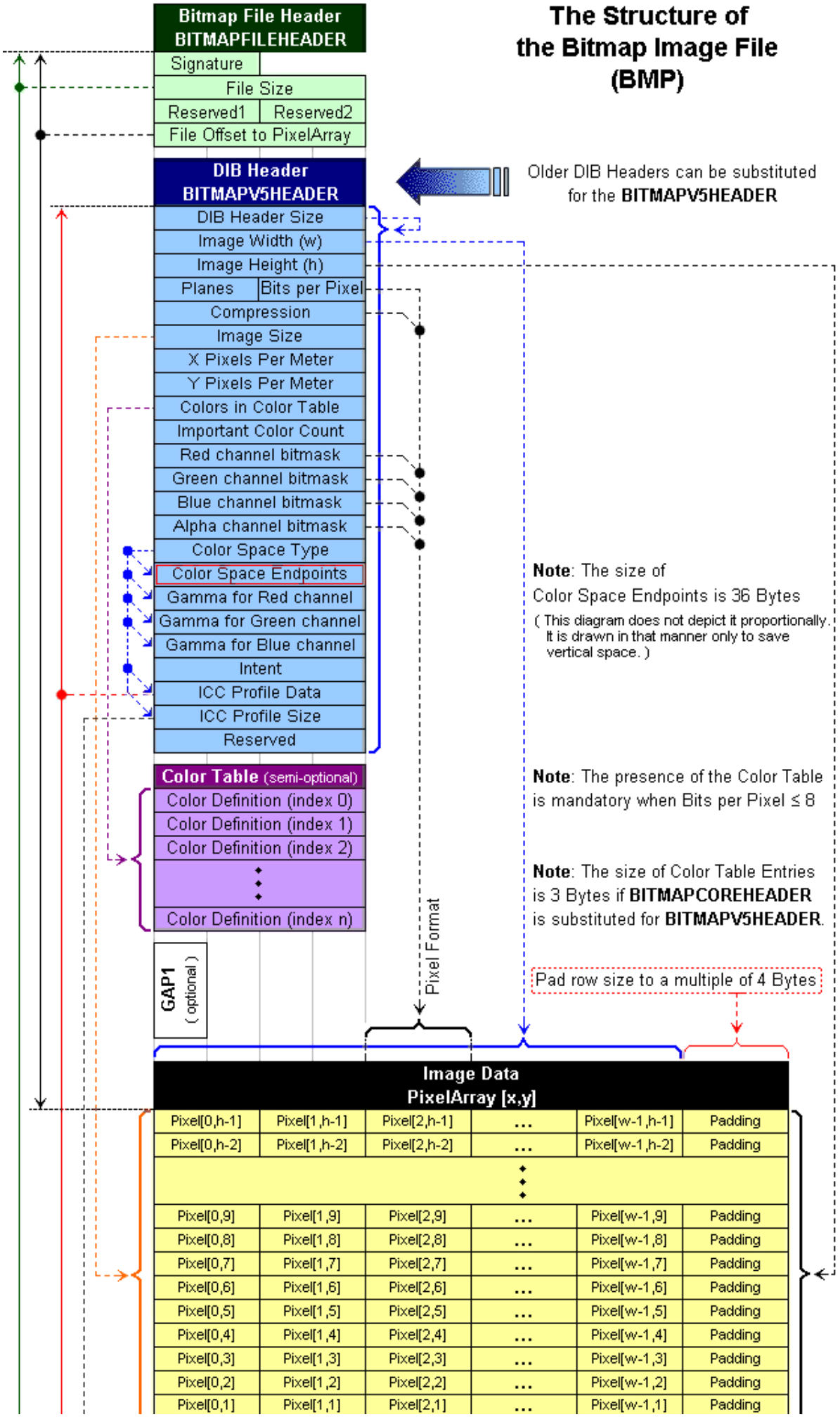
— Пол: строка(1)

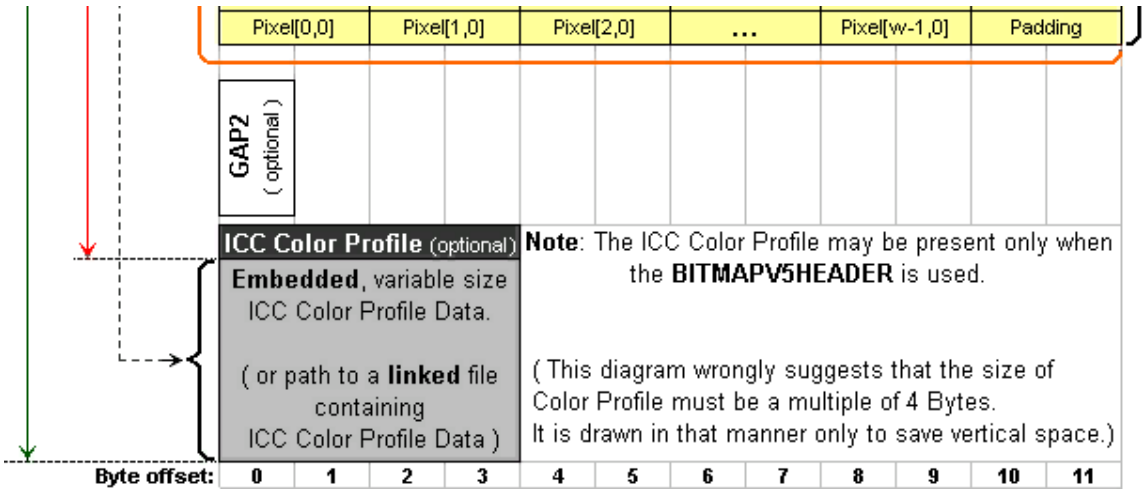
— Адрес: строка(100)

Как видно из примера, в мире нашей программы существует ряд ограничений, например, нет адресов, которые содержат более 100 символов.

Преимущества бинарных форматов с фиксированной длиной записи аналогичны преимуществам по работе с массивами: мы за константное время можем взять любую по счету запись, просто вычислив, по какому смещению в файле она располагается относительно его начала.

Примером файлов такого рода является, например, BMP-файл.





Его структура такова: в первых 54 байтах хранится так называемый **заголовок**. *Иногда его ещё называют мета-данными, то есть данными о данных.* Он информирует о размере изображения, сжатии и других характеристиках всего файла. После заголовка подряд записывается информация о каждом пикселе изображения, на каждый пиксель отводится ровно 3 байта, по одному байту на каналы R, G, B.

Фиксированный формат имеет и недостатки. Один из самых больших в том, что как только появляется запись, одно из полей которой **не влезает** в предопределённую длину, мы не сможем использовать данный формат или вынуждены будем его видоизменить.

Причём, расширяя размер поля для данной записи, мы расширяем её и для всех остальных, расходуя понапрасну ресурсы памяти.

Одна из самых широко известных ситуаций, связанных с фиксированным форматом — проблема 2000 года.

3. Форматы с произвольной длиной записи

Если длина полей в одной записи не фиксирована (например, мы говорим, что адрес клиента может быть любой длины), то нам нужно решить две проблемы:

1. Как мы будем понимать, где **границы полей**.

2. Как мы будем понимать, где **границы записей**.

Решать их можно по-разному, например, перед каждым полем делать служебное поле из 4 байт, которое показывает размер в байтах следующего поля. Таким образом, в нашем случае поле не сможет содержать более 4 Гб данных.

Можно пойти другим путём и ввести специальные разделители. Поговорим про этот случай. Воспользуемся модельным примером и рассмотрим прайс-лист товаров в магазине «Икея».

Вот небольшой фрагмент этого файла:

```
> keywords      price   product_name

> МОРУМ, Ковёр, безворсовый     6999    МОРУМ

> МОРУМ, Ковёр, безворсовый     6999    МОРУМ

> ИДБИ, Придверный коврик       649     ИДБИ

> ХОДДЕ, Ковёр, безворсовый     1399    ХОДДЕ

> ОПЛЕВ, Придверный коврик      599     ОПЛЕВ

> ОПЛЕВ, Придверный коврик      599     ОПЛЕВ
```

Разделителем записей был выбран символ **новой строки**, а разделителем полей — **табуляция** (хотя, на самом деле, увидеть символы табуляции можно с трудом).

В результате мы получили текстовый файл, который легко читается и человеком, и компьютерной программой.

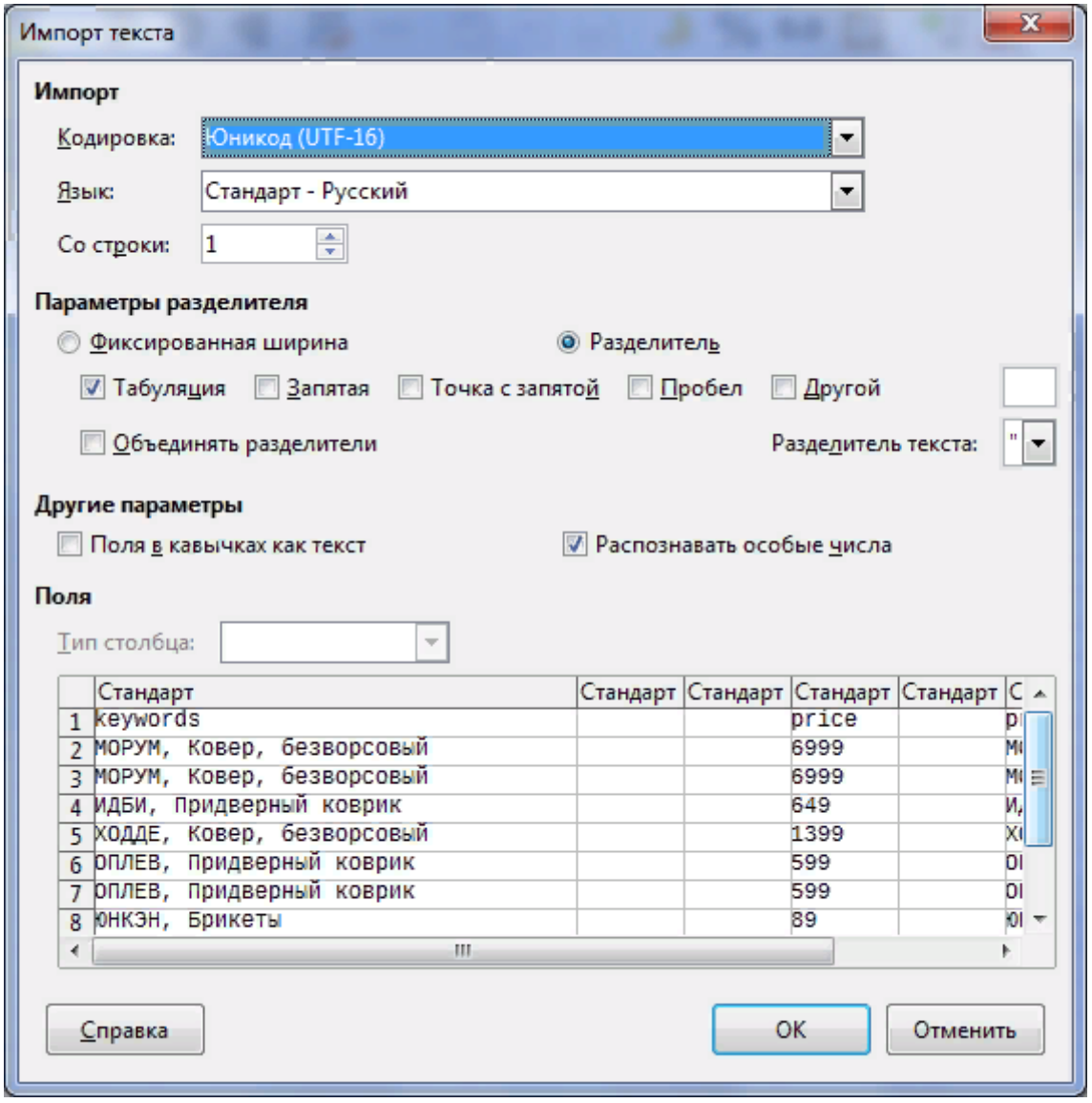
Назовём такой формат **TSV**.

TSV

TSV (англ. tab separated values — значения, разделённые табуляцией) — текстовый формат для представления таблиц баз данных. Каждая запись в таблице — это строка текстового файла. Каждое поле записи отделяется от других символом табуляции, а точнее — горизонтальной табуляции.

TSV — это форма более общего формата **DSV** — значения, разграниченные разделителем (англ. delimiter separated values).

Программы для работы с электронными таблицами (MS Excel, LibreOffice Calc) поддерживают импорт из такого формата.



Посмотрим, как работать с таким форматом в Python. Тут нет ничего сложного, поскольку у нас есть мощный функционал по работе со строками, в частности, метод **split**.

```
data = open('files/ikea.txt', encoding='utf-8').read()
for row in data.split('\n')[:10]:
    print(row.split('\t'))

----------------------------------------------------------------

    ['keywords', 'price', 'product_name']
    ['МОРУМ, Ковёр, безворсовый', '6999', 'МОРУМ']
    ['МОРУМ, Ковёр, безворсовый', '6999', 'МОРУМ']
    ['ИДБИ, Придверный коврик', '649', 'ИДБИ']
    ['ХОДДЕ, Ковёр, безворсовый', '1399', 'ХОДДЕ']
    ['ОПЛЕВ, Придверный коврик', '599', 'ОПЛЕВ']
    ['ОПЛЕВ, Придверный коврик', '599', 'ОПЛЕВ']
    ['ЮНКЭН, Брикеты', '89', 'ЮНКЭН']
    ['БУНСЁ, Детское садовое кресло', '1199', 'БУНСЁ']
    ['ИКЕА ПС ВОГЭ, Садовое лёгкое кресло', '1999', 'ИКЕА ПС ВОГЭ']
```

Вспомним списочные выражения и сразу сделаем «двумерный массив», а потом обратимся к цене пятого по счёту товара:

```
table = [r.split('\t') for r in data.split('\n')]
print(table[5][1])

----------------------------------------------------------

    599
```

Мы можем также отсортировать элементы по цене и напечатать 10 самых дешёвых товаров:

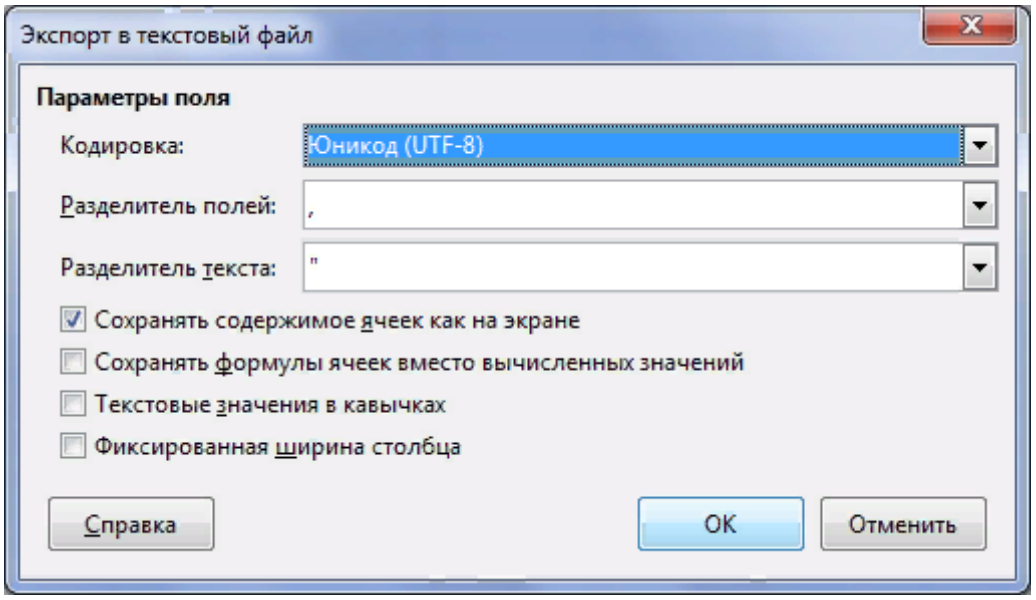
```
table = table[1:]
table.sort(key=lambda x: int(x[1]))
for r in table[:10]:
    print(r)

----------------------------------------------------------

    ['СМОРИСКА, Стопка', '6', 'СМОРИСКА']
    ['СМОРИСКА, Стакан', '12', 'СМОРИСКА']
    ['ДИСТАНС, Контейнер', '12', 'ДИСТАНС']
    ['ДИСТАНС, Контейнер', '12', 'ДИСТАНС']
    ['ОППЕН, Миска', '25', 'ОППЕН']
    ['ДАРРОКА, Стакан', '25', 'ДАРРОКА']
    ['АНТАГЕН, Щетка для мытья посуды', '25', 'АНТАГЕН']
    ['ВАНКИВА, Рама', '25', 'ВАНКИВА']
    ['ХОППЛЁС, Доска разделочная', '27', 'ХОППЛЁС']
    ['ДАРРОКА, Стакан д/виски', '29', 'ДАРРОКА']
```

Одним из самых распространённых форматов DSV стал **CSV** — формат с разделителем полей-запятой (англ. comma separated values). Наш файл будет выглядеть в нём вот так:

```
> keywords,price,product_name

>"МОРУМ, Ковёр, безворсовый",6999,МОРУМ

>"МОРУМ, Ковёр, безворсовый",6999,МОРУМ

>"ИДБИ, Придверный коврик",649,ИДБИ

>"ХОДДЕ, Ковёр, безворсовый",1399,ХОДДЕ
```

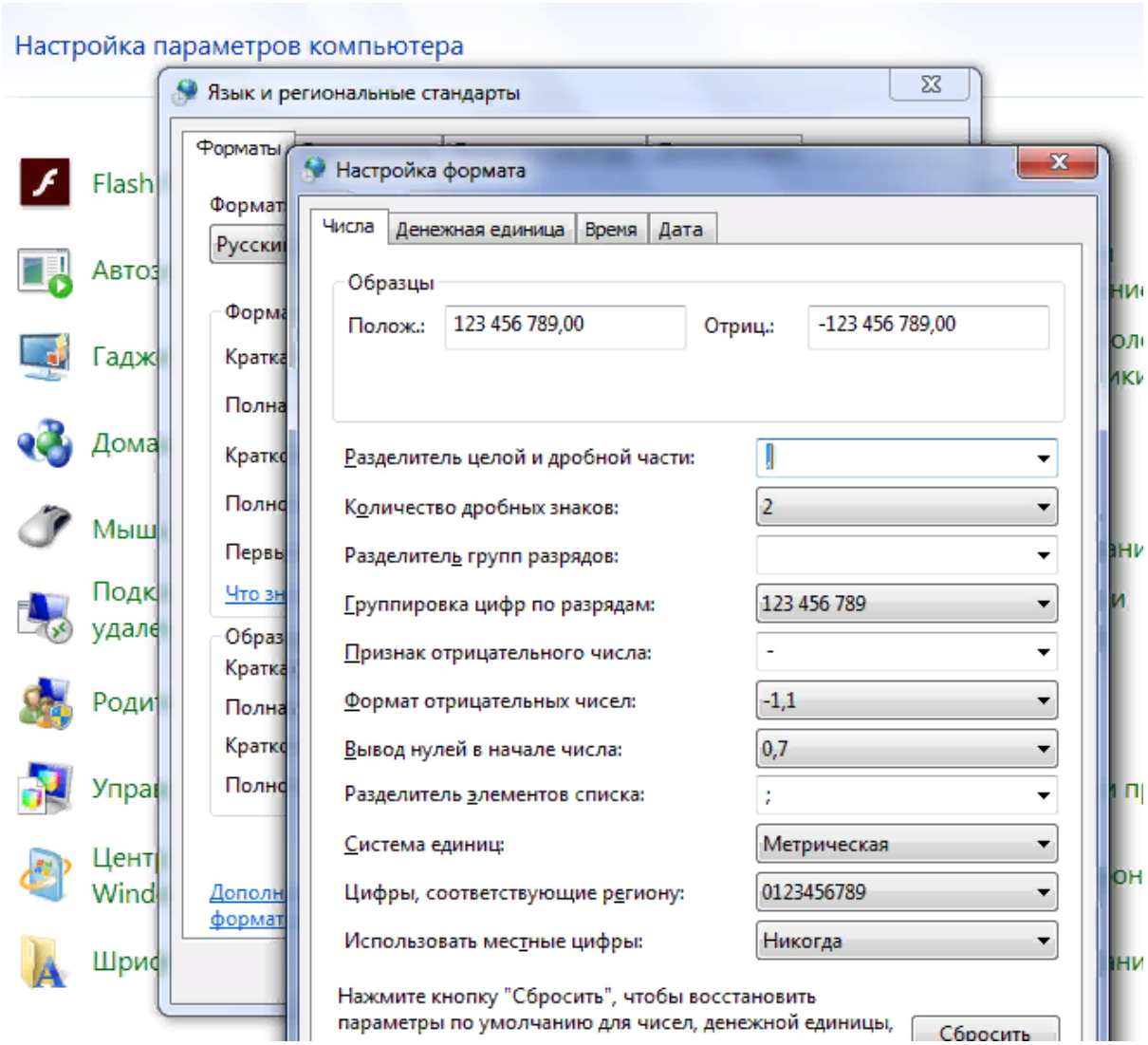
```
>"ОПЛЕВ, Придверный коврик",599,ОПЛЕВ

>"ОПЛЕВ, Придверный коврик",599,ОПЛЕВ

>"ЮНКЭН, Брикеты",89,ЮНКЭН

>"БУНСЁ, Детское садовое кресло",1199,БУНСЁ

>"ИКЕА ПС ВОГЭ, Садовое лёгкое кресло",1999,ИКЕА ПС ВОГЭ
```

Для всех форматов DSV **проблемой** является символ-разделитель полей в данных. В этом случае вводят так называемый **разделитель текста**, в качестве которого выступают двойные кавычки, а если в поле встречается сам разделитель текста, то его удваивают. Например, **ООО "Светлана"** превращается в **"ООО ""Светлана"""** при записи в файл.



Вот почему в приведённом фрагменте CSV-файла первое поле в кавычках — внутри него есть запятые.

Нужно отметить, что в CSV могут быть другие разделители, например, точка с запятой. Очень часто это регулируется настройками операционной системы (параметр «разделитель элементов списка» в ОС Windows):



4. Библиотека CSV

Несмотря на то, что DSV-форматы просты, отсутствие чётких стандартов в выборе разделителей и экранировании символов привели к тому, что с ними лучше работать при помощи специализированных библиотек, а не в стиле «использования функции» split.

Для работы с такими форматами в Питоне есть модуль csv.

В модуле есть два основных объекта: **reader** и **writer**, созданные, чтобы читать и создавать csv-файлы соответственно.

Приведём пример использования **читателя** с почти полным набором значений, указав:

— кодировку файла;

— символ-разделитель;

— разделитель текста.

Объект **reader** даёт доступ к построчному итератору полностью аналогично работе с файлом или списком.

Общности ради в следующем примере мы покажем, что разделителем может быть любой символ.

```
import csv
with open('files/ikea.csv', encoding="utf8") as csvfile:
    reader = csv.reader(csvfile, delimiter=';', quotechar='"')
    for index, row in enumerate(reader):
        if index > 10:
            break
        print(row)

------------------------------------------------------------

    ['keywords', 'price', 'product_name']
    ['МОРУМ, Ковёр, безворсовый', '6999', 'МОРУМ']
    ['МОРУМ, Ковёр, безворсовый', '6999', 'МОРУМ']
    ['ИДБИ, Придверный коврик', '649', 'ИДБИ']
    ['ХОДДЕ, Ковёр, безворсовый', '1399', 'ХОДДЕ']
    ['ОПЛЕВ, Придверный коврик', '599', 'ОПЛЕВ']
    ['ОПЛЕВ, Придверный коврик', '599', 'ОПЛЕВ']
    ['ЮНКЭН, Брикеты', '89', 'ЮНКЭН']
    ['БУНСЁ, Детское садовое кресло', '1199', 'БУНСЁ']
    ['ИКЕА ПС ВОГЭ, Садовое лёгкое кресло', '1999', 'ИКЕА ПС ВОГЭ']
    ['КУНГСХОЛЬМЕН, Садовый табурет', '5500', 'КУНГСХОЛЬМЕН']
```

Давайте разберём построчно, что происходит в этом коде.

Мы пользуемся with, чтобы просто открыть наш файл с кодировкой utf8, а потом создаём объект reader, говоря ему про символы-разделители полей и строк.

Объект reader может служить итератором (и использоваться в цикле for) по строкам, каждая из которых представляет собой список. Кроме того, мы используем enumerate для того, чтобы посчитать строки.

Отметим, что исходный файл содержит подписи полей в первой строке, что будет снова нам мешать (например, при сортировке строк). Для корректной работы мы должны были бы исключить первую строку из обработки.

Но в модуле csv есть специальный объект **DictReader**, который поддерживает создание объекта-словаря на основе подписей к полям.

Теперь мы можем обращаться к полям не по индексу, а по **названию**, что делает программу ещё более понятной.

Найдем топ-10 самых дорогих товаров (как вы думаете, какая запись более понятна: int(x["price"]) или int(x[1])?):

```
with open('files/ikea.csv', encoding="utf8") as csvfile:
    reader = csv.DictReader(csvfile, delimiter=';', quotechar='"')
    expensive = sorted(reader, key=lambda x: int(x['price']), reverse=True)

for record in expensive[:10]:
    print(record)
```

```
------------------------------------------------------------

    OrderedDict([('keywords', 'ГРИЛЬЕРА, Плита'), ('price', '99999'), ('product_r
    OrderedDict([('keywords', 'ГРИЛЬЕРА, Плита'), ('price', '99999'), ('product_r
    OrderedDict([('keywords', 'КИВИК, Диван-кровать 3-местный'), ('price', '79999
    OrderedDict([('keywords', 'КИВИК, Диван-кровать 3-местный'), ('price', '79999
    OrderedDict([('keywords', 'СТОКГОЛЬМ, Диван 3-местный'), ('price', '69999'),
    OrderedDict([('keywords', 'ИСАНДЕ, Встраив холодильник/морозильник A++'), ('p
    OrderedDict([('keywords', 'КУЛИНАРИСК, Комбинир СВЧ с горячим обдувом'), ('pr
    OrderedDict([('keywords', 'ХОГКЛАССИГ, Индукц варочн панель'), ('price', '499
    OrderedDict([('keywords', 'ГРЭНСЛЁС, Комбинир СВЧ с горячим обдувом'), ('pric
    OrderedDict([('keywords', 'КУЛИНАРИСК, Духовка/пиролитическая самоочистка'),
```

Мы привели цены к типу int, потому что строки сравниваются в лексикографическом порядке (по алфавиту). Например:

```
print('11'>'100')
```

```
------------------------------------------------------------

    True
```

```
print(11>100)
```

```
------------------------------------------------------------

    False
```

Использование объекта для записи (**writer**) аналогично «читателю» (**reader**):

```
with open('files/квадраты.csv', 'w', newline='') as csvfile:
    writer = csv.writer(
        csvfile, delimiter=';', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    for i in range(10):
        writer.writerow([i, i**2, \
                 "Квадрат числа %d равен %d" % (i, i**2)])
```

Выполните этот код и посмотрите, что получилось.

Скачать файл ikea.csv можно по ссылке.