



Скачайте Яндекс Браузер для образования

Скачать

&lt; Урок QT CSV

## Работа с простыми таблицами (csv). Работа с табличными данными в PyQT

- 1 Хранение записей в файлах
- 2 Форматы с фиксированной длиной записи
- 3 Форматы с произвольной длиной записи
- 4 Формат TSV
- 5 Формат CSV
- 6 Библиотека CSV
- 7 Использование графического интерфейса. Чтение из .csv файла
- 8 PyQT. Запись из таблицы в .csv файл

### Аннотация

Урок посвящен 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, методам работы с ними при помощи строковых функций, а также специализированной библиотеки csv и PyQT.

### 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 договоренности по формату. При разработке приложений очень быстро возникла необходимость хранения в файлах больших массивов однородных объектов (записей), имеющих некоторый набор полей или характеристик. По сути такой массив представляет собой таблицу со строками и столбцами. Каждый столбец имеет свой, как правило, примитивный тип: целое или вещественное число, строка 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**, о которых мы поговорим уже очень скоро.

## 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 еще 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. То есть информация об изображении представляет собой массив (список) кортежей из трех чисел — значений red, green и blue составляющих цвета пикселя.

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

Причем, расширяя размер поля для данной записи, мы расширяем ее 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), 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 путем 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.

Назовем такой формат **TSV**.

## 4. Формат TSV

### TSV

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

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

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

### Импорт из текстового файла в Excel

Посмотрим, как работать с таким форматом в Python. Тут нет ничего сложного, поскольку у нас есть мощная функциональность 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 счету 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 дешевых 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']
```

## 5. Формат CSV

Одним из самых распространенных форматов **DSV** стал **CSV** — формат с разделителем полей — запятой (англ. comma separated values). Наш файл будет выглядеть в нем **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

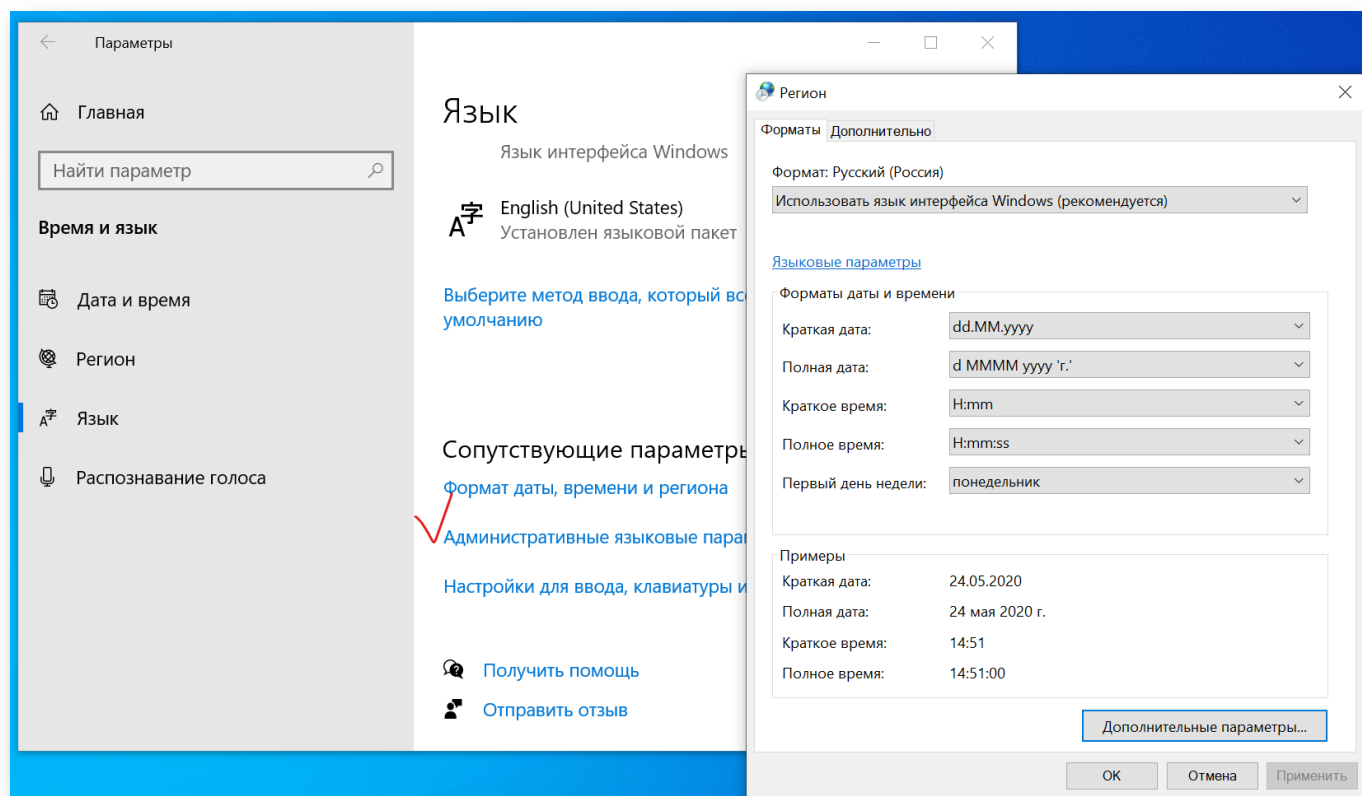
>"ИКЕА ПС ВОГЭ, Садовое лёгкое кресло",1999,ИКЕА ПС ВОГЭ

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"** при записи в файл превращается в **"ООО ""Светлана"""**.

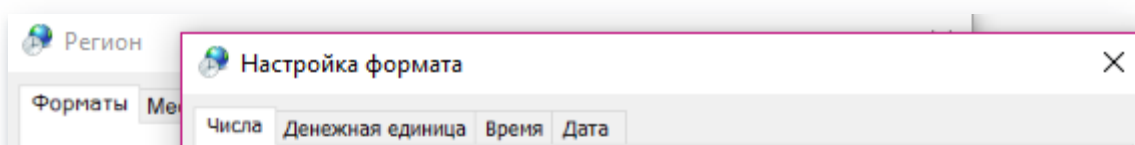
Вот почему в приведенном фрагменте CSV-файла первое поле в кавычках — внутри него есть запятые.

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):

Параметры — Время и язык — Язык:



Дополнительные параметры:



Формат: Рус  
Использовать

Языковые па

Образцы

Положительное: 123 456 789,00    Отрицательное: -123 456 789,00

Разделитель целой и дробной части: ,

Количество дробных знаков: 2

Разделитель групп разрядов:

Группировка цифр по разрядам: 123 456 789

Признак отрицательного числа: -

Формат отрицательных чисел: -1,1

Вывод нулей в начале числа: 0,7

Разделитель элементов списка: ;

Система единиц: Метрическая

Цифры, соответствующие региону: 0123456789

Использовать местные цифры: Никогда

Нажмите кнопку "Сбросить", чтобы восстановить параметры по умолчанию для чисел, денежной единицы, времени и даты.

Сбросить

ОК    Отмена    Применить

## 6. Библиотека CSV

Несмотря на то, что DSV-форматы просты, отсутствие четких 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 Python есть модуль **csv**.

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

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

- Кодировку файла
- Символ-разделитель
- Разделитель текста

Объект **reader** дает 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 разберем построчно, что происходит в этом коде.

Мы пользуемся менеджером контекста `with`, чтобы просто открыть наш файл с кодировкой UTF-8, а потом создаем 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. При создании `reader` мы указываем, что символ-разделитель записей `delimiter` в нашем файле — точка с запятой, а символ кавычек `quotechar` — двойные кавычки. Кроме того, мы используем `enumerate`, 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 еще 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)
```

```
{'keywords': 'ГРИЛЬЕРА, Плита', 'price': '99999', 'product_name': 'ГРИЛЬЕРА'}
{'keywords': 'ГРИЛЬЕРА, Плита', 'price': '99999', 'product_name': 'ГРИЛЬЕРА'}
{'keywords': 'КИВИК, Диван-кровать 3-местный', 'price': '79999', 'product_name': 'КИВИК'}
{'keywords': 'КИВИК, Диван-кровать 3-местный', 'price': '79999', 'product_name': 'КИВИК'}
{'keywords': 'СТОКГОЛЬМ, Диван 3-местный', 'price': '69999', 'product_name': 'СТОКГОЛЬМ'}
```

```
{'keywords': 'ИСАНДЕ, Встраив холодильник/морозильник А++', 'price': '59999', 'product_name': 'ИСАНДЕ, Встраив холодильник/морозильник А++'}
{'keywords': 'КУЛИНАРИСК, Комбинир СВЧ с горячим обдувом', 'price': '54999', 'product_name': 'КУЛИНАРИСК, Комбинир СВЧ с горячим обдувом'}
{'keywords': 'ХОГКЛАССИГ, Индукц варочн панель', 'price': '49999', 'product_name': 'ХОГКЛАССИГ, Индукц варочн панель'}
{'keywords': 'ГРЭНСЛЁС, Комбинир СВЧ с горячим обдувом', 'price': '49999', 'product_name': 'ГРЭНСЛЁС, Комбинир СВЧ с горячим обдувом'}
{'keywords': 'КУЛИНАРИСК, Духовка/пиролитическая самоочистка', 'price': '49999', 'product_name': '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='', encoding="utf8") 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)])
```

В этом случае использовался опциональный параметр функции `open()` `newline`. Он отвечает за переводы строк при чтении или записи в текстовый файл. По умолчанию имеет значение `None`, в этом случае все разделители строк преобразуются в `\n`. Если в файле оказывается лишний перевод строки, то следует использовать этот параметр в режиме `newline=''`, тогда `\n` будет преобразован в пустую строку.

Выполните этот код и посмотрите, что получилось.

Записывать в csv-файл можно и с помощью `DictWriter`, аналогичного `DictReader`. Но нужно ему указать, какие заголовки должны быть в файле и какие значения им соответствуют у каждой записи. Для этого сначала подготовим список словарей:

```
import csv

data = [{
    'lastname': 'Иванов',
    'firstname': 'Пётр',
    'class_number': 9,
    'class_letter': 'A'
}, {
    'lastname': 'Кузнецов',
    'firstname': 'Алексей',
    'class_number': 9,
    'class_letter': 'B'
}, {
    'lastname': 'Меньшова',
    'firstname': 'Алиса',
    'class_number': 10,
    'class_letter': 'A'
}]
```



```

        'class_number': 9,
        'class_letter': 'А'
    }, {
        'lastname': 'Иванова',
        'firstname': 'Татьяна',
        'class_number': 9,
        'class_letter': 'Б'
    }]

with open('dictwriter.csv', 'w', newline='', encoding="utf8") as f:
    writer = csv.DictWriter(
        f, fieldnames=list(data[0].keys()),
        delimiter=';', quoting=csv.QUOTE_NONNUMERIC)
    writer.writeheader()
    for d in data:
        writer.writerow(d)

```

Результат:

```

"lastname";"firstname";"class_number";"class_letter"
"Иванов";"Пётр";9;"А"
"Кузнецов";"Алексей";9;"В"
"Меньшова";"Алиса";9;"А"
"Иванова";"Татьяна";9;"Б"

```

Скачать файл ikea.csv можно по [ссылке](#).

## 7. Использование графического интерфейса. Чтение из .csv файла

Поскольку большинство программ используются людьми без опыта работы в командной строке, в пакетах, отвечающих за создание графического интерфейса, существуют специальные модули для работы с табличными данными. Для работы с таблицами в PyQT существует класс **Table Widget**. Давайте рассмотрим несколько примеров его использования.

Создадим простое приложение, которое будет отображать содержимое таблицы, которая хранится в формате .csv.

Для этого создадим в QtDesigner новую форму, добавим на нее **Table Widget**, а затем подключим интерфейс к нашей программе (в виде ui-файла или конвертируем полученный GUI в файл с расширением .py).

Работа с CSV			
	1	2	3
1	Адамович А., Гранин Д.	Блокадная книга	322
2	Айтманов Ч.	И дольше века длится день	168
3	Аксенов В.	Звездный билет	609
4	Алексин А.	Мой брат играет на кларнете	558
5	Арсеньев В.	ДерсуУзала	153
6	Астафьев В.	Пастух и пастушка	408
7	Бабель И.	Одесские рассказы	230

8	Бажов П.	Уральские рассказы	459
9	Белых Л., Пантелеев Л.	Республика ШКИД	175
10	Богомолов В.	Момент истины	548

```
import csv
import sys

from PyQt5 import uic
from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QMainWindow, QTableWidgetItem

class MyWidget(QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi('UI1.ui', self)
        self.loadTable('Data.csv')

    def loadTable(self, table_name):
        with open(table_name, encoding="utf8") as csvfile:
            reader = csv.reader(csvfile, delimiter=',', quotechar='"')
            title = next(reader)
            self.tableWidget.setColumnCount(len(title))
            self.tableWidget.setHorizontalHeaderLabels(title)
            self.tableWidget.setRowCount(0)
            for i, row in enumerate(reader):
                self.tableWidget.setRowCount(
                    self.tableWidget.rowCount() + 1)
                for j, elem in enumerate(row):
                    self.tableWidget.setItem(
                        i, j, QTableWidgetItem(elem))
            self.tableWidget.resizeColumnsToContents()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = MyWidget()
    ex.show()
    sys.exit(app.exec())
```

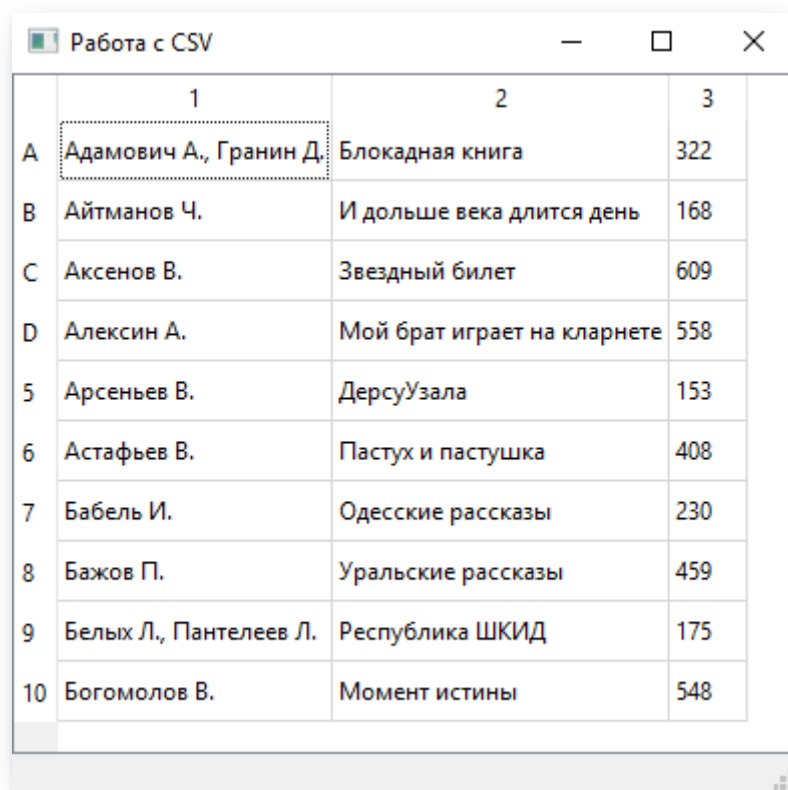
В методе `loadTable()` класса `MyWidget` происходит загрузка содержимого в виджет таблицы. Давайте посмотрим более детально.

Для отображения таблицы необходимо указать ее размеры: количество строк и столбцов. Если бы у нас была фиксированная матрица, мы бы легко узнали ее размеры, но в случае работы с `.csv` мы используем итератор, поэтому необходимо применить некоторые «лайфхаки».

Сначала, так же, как и в прошлых примерах, мы открываем файл для чтения. Поскольку `reader` является итератором, то, для того чтобы получить первую строку, необходимо воспользоваться функцией `next()`.

Чтобы отобразить заголовки в виджете, воспользуемся методом `setHorizontalHeaderLabels()`. Кстати, существует также функция `setVerticalHeaderLabels()` — для задания вертикальных заголовков. По умолчанию, если программист не задает текст заголовков, это просто номера. Если в функцию передано недостаточное количество заголовков, после того как они «закончатся», следующие заголовки будут цифровыми. Например:

```
self.tableWidget.setVerticalHeaderLabels(['A', 'B', 'C', 'D'])
```



	1	2	3
A	Адамович А., Гранин Д.	Блокадная книга	322
B	Айтманов Ч.	И дольше века длится день	168
C	Аксенов В.	Звездный билет	609
D	Алексин А.	Мой брат играет на кларнете	558
5	Арсеньев В.	ДерсуУзала	153
6	Астафьев В.	Пастух и пастушка	408
7	Бабель И.	Одесские рассказы	230
8	Бажов П.	Уральские рассказы	459
9	Белых Л., Пантелеев Л.	Республика ШКИД	175
10	Богомолов В.	Момент истины	548

Зная количество заголовков, можно установить количество столбцов в нашей таблице, используя метод `setColumnCount()`. Но так как мы не знаем (если не преобразовали `reader` в список), сколько элементов в нашем итераторе, установим изначальное количество, равное 0, а затем будем на каждом шаге увеличивать его на единицу.

Основным методом при работе с `QTableWidget` является `tableWidget.setItem(i, j, QTableWidgetItem(elem))`. Он помещает в заданную с помощью координат ячейку соответствующее значение. Важно не забыть, что в таблице хранятся не текстовые, не числовые и не какие-либо другие данные, а данные типа `QTableWidgetItem`, поэтому не забывайте приводить значения к этому типу.

Очень часто непонятно заранее, какого размера будет содержимое той или иной ячейки в таблице. Для того чтобы размеры ячеек соответствовали длине текста, используем метод `resizeColumnsToContents()` после того, как заполним таблицу данными. Он растянет или сузит, при необходимости, размеры ячейки, чтобы пользователю был виден весь текст.

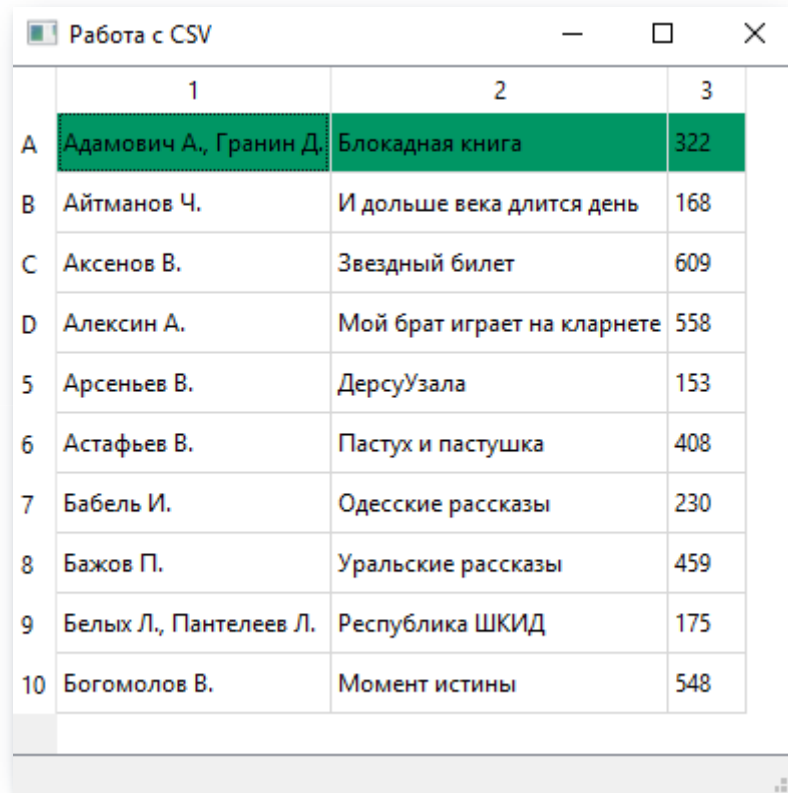
Согласитесь, что сейчас таблица выглядит блекло и неинтересно. Давайте раскрасим ее. Для этого нам понадобится импортировать модуль, отвечающий за работу с цветом: `from PyQt5.QtGui import QColor`. К сожалению, по умолчанию в PyQT нет функции, чтобы изменить цвет целого ряда таблицы, а есть только для одной ячейки. Так что придется написать такой метод самостоятельно.

```
def color_row(self, row, color):
    for i in range(self.tableWidget.columnCount()):
        self.tableWidget.item(row, i).setBackground(color)
```

И теперь, если мы захотим выделить нулевой ряд цветом, нам достаточно лишь вызвать этот метод:

```
self.color_row(0, QColor(0, 150, 100))
```

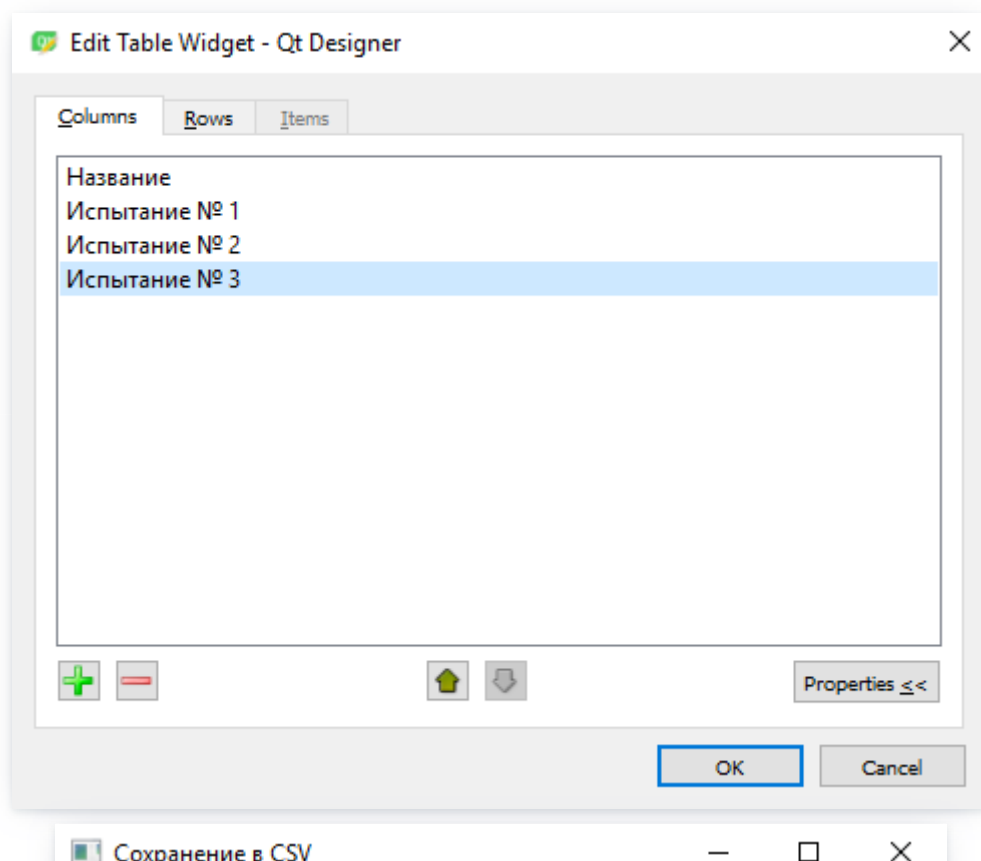
Получим такой результат:



	1	2	3
A	Адамович А., Гранин Д.	Блокадная книга	322
B	Айтманов Ч.	И дольше века длится день	168
C	Аксенов В.	Звездный билет	609
D	Алексин А.	Мой брат играет на кларнете	558
5	Арсеньев В.	ДерсуУзала	153
6	Астафьев В.	Пастух и пастушка	408
7	Бабель И.	Одесские рассказы	230
8	Бажов П.	Уральские рассказы	459
9	Белых Л., Пантелеев Л.	Республика ШКИД	175
10	Богомолов В.	Момент истины	548

## 8. PyQT. Запись из таблицы в .csv файл

Мы уже научились читать данные из файла и отображать их с помощью таблицы. Теперь попробуем обратный процесс. Для этого напишем программу, которая может использоваться при выставлении баллов команде за различные этапы на каком-нибудь соревновании. Известно, что и команд, и испытаний фиксированное число. Так что можно создать сетку для нашей таблицы, используя QtDesigner. Для этого необходимо навести курсор на виджет, нажать правую кнопку мыши, выбрать пункт меню **Edit Items...**, а затем указать данные.



	Название	Испытание №1	Испытание №2	Испытание №3
1				
2				
3				
4				
5				
6				

Сохранить в CSV

Для сохранения данных напомним метод `save_2_csv`:

```
def save_2_csv(self):
    with open('results.csv', 'w', newline='') as csvfile:
        writer = csv.writer(
            csvfile, delimiter=';', quotechar='\"',
            quoting=csv.QUOTE_MINIMAL)
        # Получение списка заголовков
        writer.writerow(
            [self.tableWidget.horizontalHeaderItem(i).text()
             for i in range(self.tableWidget.columnCount())])
        for i in range(self.tableWidget.rowCount()):
            row = []
            for j in range(self.tableWidget.columnCount()):
                item = self.tableWidget.item(i, j)
                if item is not None:
                    row.append(item.text())
            writer.writerow(row)
```

К сожалению, в PyQT нет встроенной функции, которая возвращает список заголовков, так что пришлось использовать генератор и встроенный метод `horizontalHeaderItem(i)`, который возвращает *i*-й заголовок. А затем каждая строка переносится в список, а этот список уже записывается в файл.

Сохранение в CSV

	Название	Испытание №1	Испытание №2	Испытание №3
1	Альтаир	5	8	7
2	Бригантина	5	3	9
3	Будильник	4	6	10
4	220 Вольт	3	4	5
5	Непоседы	5	10	6
6	Трубодуры	1	9	8

Сохранить в CSV

Результатом будет такой .csv-файлик:

Название	Испытание №1	Испытание №2	Испытание №3
Альтаир	5	8	7
Бригантина	5	3	9
Будильник	4	6	10
220 Вольт	3	4	5
Непоседы	5	10	6
Трубодуры	1	9	8

Справка

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках сервиса, принадлежат АНО ДПО «Образовательные технологии Яндекса». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «Образовательные технологии Яндекса».

[Пользовательское соглашение.](#)

© 2018 – 2024 ООО «Яндекс»