* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/165/groups/1257)

[Урок QT csv](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1229)

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

1. [Хранение записей в файлах](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1229/materials/2215#1)
2. [Форматы с фиксированной длиной записи](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1229/materials/2215#2)
3. [Форматы с произвольной длиной записи](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1229/materials/2215#3)
4. [Библиотека CSV](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1229/materials/2215#4)
5. [PyQT. Чтение из .csv файла](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1229/materials/2215#5)
6. [PyQT. Запись из таблицы в .csv файл](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1229/materials/2215#6)

**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 и специализированной библиотеки csv.*

**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. Очень быстро стало нужно хранить в файлах массивы 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.

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](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), которые мы не рассматриваем на этой лекции.

**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.

Фиксированный формат имеет и недостатки. Один из самых больших в том, что, как только появляется запись, одно из полей которой **не влезает** в предопределенную длину, 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](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B0_2000_%D0%B3%D0%BE%D0%B4%D0%B0).

**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](https://yastatic.net/s3/lyceum/content/resources/ikea.txt) 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.

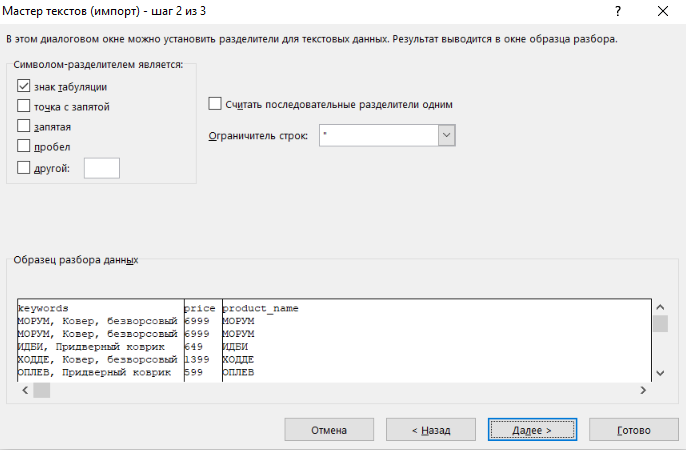
Назовем такой формат **TSV**.

**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.



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']

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](https://yastatic.net/s3/lyceum/content/resources/ikea.csv):

> 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

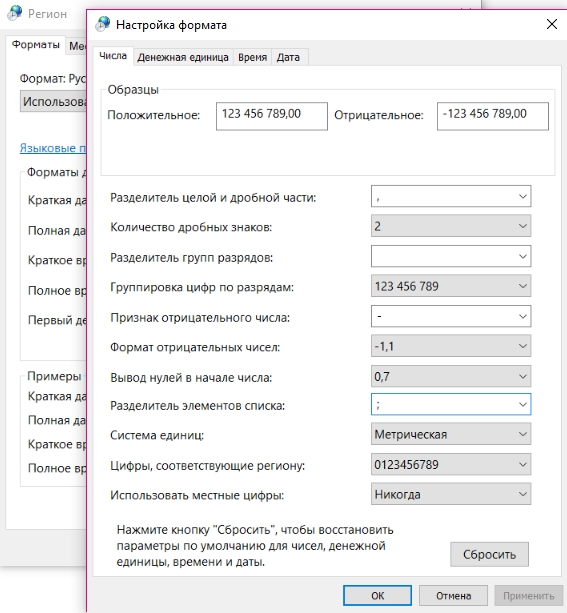
>"БУНСЁ, Детское садовое кресло",1199,БУНСЁ

>"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):



**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](https://docs.python.org/3/library/csv.html).

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 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)

OrderedDict([('keywords', 'ГРИЛЬЕРА, Плита'), ('price', '99999'), ('product\_name', 'ГРИЛЬЕРА')])

OrderedDict([('keywords', 'ГРИЛЬЕРА, Плита'), ('price', '99999'), ('product\_name', 'ГРИЛЬЕРА')])

OrderedDict([('keywords', 'КИВИК, Диван-кровать 3-местный'), ('price', '79999'), ('product\_name', 'КИВИК')])

OrderedDict([('keywords', 'КИВИК, Диван-кровать 3-местный'), ('price', '79999'), ('product\_name', 'КИВИК')])

OrderedDict([('keywords', 'СТОКГОЛЬМ, Диван 3-местный'), ('price', '69999'), ('product\_name', 'СТОКГОЛЬМ')])

OrderedDict([('keywords', 'ИСАНДЕ, Встраив холодильник/морозильник А++'), ('price', '59999'), ('product\_name', 'ИСАНДЕ')])

OrderedDict([('keywords', 'КУЛИНАРИСК, Комбинир СВЧ с горячим обдувом'), ('price', '54999'), ('product\_name', 'КУЛИНАРИСК')])

OrderedDict([('keywords', 'ХОГКЛАССИГ, Индукц варочн панель'), ('price', '49999'), ('product\_name', 'ХОГКЛАССИГ')])

OrderedDict([('keywords', 'ГРЭНСЛЁС, Комбинир СВЧ с горячим обдувом'), ('price', '49999'), ('product\_name', 'ГРЭНСЛЁС')])

OrderedDict([('keywords', 'КУЛИНАРИСК, Духовка/пиролитическая самоочистка'), ('price', '49999'), ('product\_name', 'КУЛИНАРИСК')])

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):

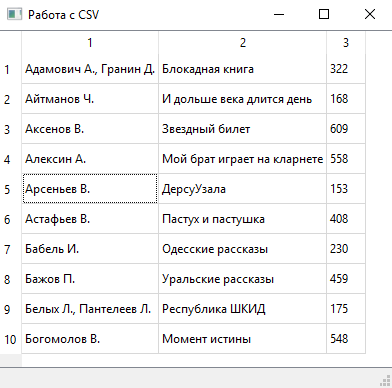
writer.writerow([i, i \*\* 2, f"Квадрат числа {i} равен {i \*\* 2}"])

**PyQT. Чтение из .csv файла**

Поскольку большинство программ используются людьми без опыта работы в командной строке, в пакетах, отвечающих за графический интерфейс, существуют специальные модули для работы с табличными данными. Для работы с таблицами в PyQT существует класс **Table Widget**.

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 конвертируем полученный GUI в файл с расширением .py (ui\_1.py), как мы делали в одном из прошлых уроков.



import sys

from PyQt5.QtWidgets import QApplication, QMainWindow,QTableWidgetItem

from ui\_1 import Ui\_Form

import csv

class MyWidget(QMainWindow, Ui\_Form):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setupUi(self)

self.loadTable('Data.csv')

def loadTable(self, table\_name):

with open('Data.csv', 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()

app = QApplication(sys.argv)

ex = MyWidget()

ex.show()

sys.exit(app.exec\_())

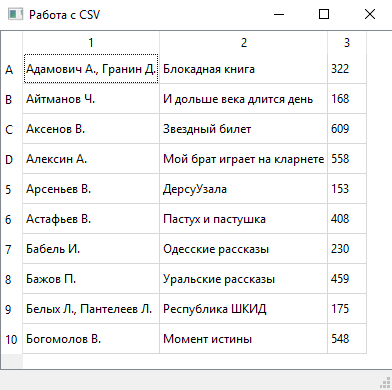
В функции loadTable происходит загрузка содержимого в виджет таблицы. Разберемся, что в ней происходит.

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 функцией 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'])



Зная количество заголовков, можно установить количество столбцов в нашей таблице, используя метод setColumnCount. Но мы ведь не знаем, сколько у нас элементов в нашем итераторе, поэтому установим изначальное количество, равное 0, а затем будем на каждом шаге увеличивать его на единицу.

Основной функцией при работе с TableWidget является tableWidget.setItem(i, j, QTableWidgetItem(elem)). Эта функция 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 функцию resizeColumnsToContents(). Она растянет или сожмет при необходимости размеры ячейки, чтобы пользователю был виден весь текст.

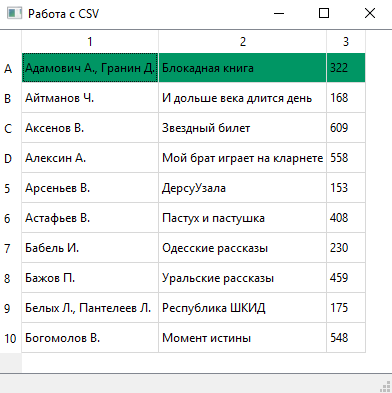
Согласитесь, что сейчас таблица выглядит блекло и неинтересно. Давайте раскрасим ее. Для этого нам понадобится импортировать модуль, отвечающий за работу с цветом: from PyQt5.QtGui import QColor. К сожалению, по умолчанию в PyQT нет функции, чтобы изменить цвет целого ряда, а есть только для одной ячейки.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

def colorRow(self, row, color):

for i in range(self.tableWidget.columnCount()):

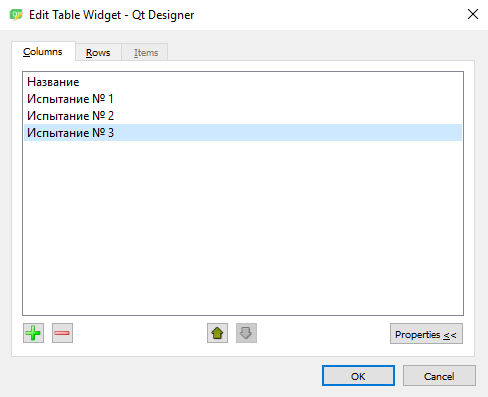
self.tableWidget.item(row, i).setBackground(color)

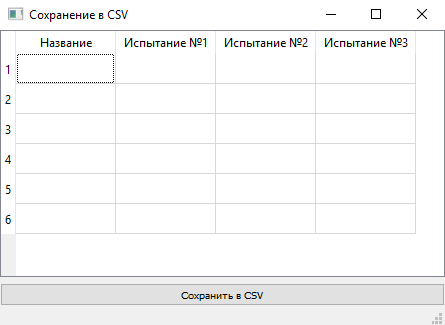
И теперь, если мы захотим выделить нулевой ряд цветом, нам достаточно лишь вызвать эту функцию: self.colorRow(0, QColor(0, 150, 100)). Получим такой результат:



**PyQT. Запись из таблицы в .csv файл**

Мы уже умеем 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. 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.





Для сохранения данных была написана функция saveToCsv:

def saveToCsv(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()):

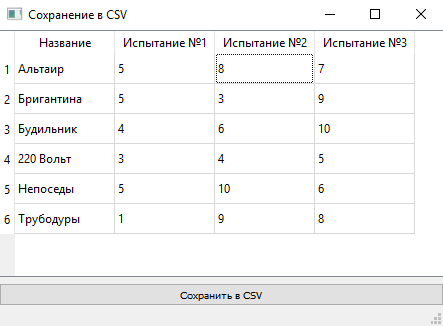
item = self.tableWidget.item(i, j)

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 встроенную функцию horizontalHeaderItem(i), которая возвращает i-й заголовок. А затем каждая строка переносится в список, после чего этот список записывается в файл.



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](https://yandex.ru/support/lyceum-students)

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

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты