

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

**ОТЧЕТ**

по лабораторной работе № 5  
на тему: «Работа с файлами»  
по дисциплине: «Программирование на языке Python»  
Вариант № 18

Выполнил: Шорин В.Д.

Шифр: 171406

Институт приборостроения, автоматизации и информационных технологий

Направление: 09.03.04 «Программная инженерия»

Группа: 71-ПГ

Проверили: Захарова О.В., Раков В.И.

Отметка о зачете:

Дата: «\_\_\_\_» \_\_\_\_\_ 2019 г.

Орел, 2019 г.

### **Задание:**

В программу лабораторной работы № 4 добавить главное меню с пунктами «Файл», «Справка».

Пункт главного меню «Файл» должен включать подпункты «Создать» (очистка формы для ввода новых данных), «Открыть» (считывание из файла информации об объектах и отображение её на форме), «Сохранить» (обновление текущего файла; если файл не создан, то действия пункта «Сохранить как...»), «Сохранить как...» (сохранение информации об объектах в файл по указанному пути), «Выход» (дружественный выход из программы).

Пункт главного меню «Справка» должен показывать информацию о разработчике программы.

### **Код:**

#### **«main.py»**

```
from PyQt5 import QtWidgets
from form import MainWindow
import sys
```

```
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    main_window = MainWindow()
    main_window.show()
    sys.exit(app.exec())
```

#### **«processor.py»**

```
class Processor:
    data = dict

    def __init__(self, **kwargs):
        self.data = kwargs

    def __str__(self):
        return f"\nArticle: {self.data.get('article')}\n" \
               f"Producer: {self.data.get('producer')}\n" \
               f"Name: {self.data.get('name')}\n" \
               f"Cores count: {self.data.get('cores')}\n" \
               f"Frequency: {self.data.get('frequency')}\n" \
               f"Price: {self.data.get('price')}\n"

    @staticmethod
    def attributes():
```

```
        return ['article', 'producer', 'name',  
                'cores', 'frequency', 'price']
```

```
def __dict__(self):  
    return {  
        'article': self.data.get('article'),  
        'producer': self.data.get('producer'),  
        'name': self.data.get('name'),  
        'cores': self.data.get('cores'),  
        'frequency': self.data.get('frequency'),  
        'price': self.data.get('price'),  
    }
```

«form.py»

```
import pickle  
import json  
import os  
from typing import List  
from PyQt5 import QtWidgets  
from PyQt5.QtWidgets import QMessageBox
```

```
import lab5 as view  
import processor as proc
```

```
global processors  
processors = list()
```

```
default_path = "C:/Users/vscho/PycharmProjects/Python_Lab5"  
user_path_to_save = "C:/Users/vscho/PycharmProject/Python_Lab5"
```

```
class MainWindow(QtWidgets.QMainWindow, view.Ui_MainWindow):  
    def __init__(self):  
        super().__init__()  
        self.setupUi(self)  
  
        self.init_actions()  
        self.init_table()  
        self.init_values()  
        self.init_menu()  
  
        self.update_table_content()  
  
    def init_actions(self):  
        # set actions on buttons  
        self.buttonAdd.clicked.connect(self.add_processor)
```

```
self.buttonDelete.clicked.connect(self.delete_processor)
self.buttonSearch.clicked.connect(self.search_processor)
self.buttonPrintAll.clicked.connect(self.print_all)
self.buttonPrintByCheckBox.clicked.connect(self.print_by_combo_box)
```

```
def init_values(self):
    self.comboBoxCores.clear()
    self.comboBoxCores.addItem(['None', '2', '4', '6'])
```

```
processors.append(proc.Processor(
    article=1,
    producer=1,
    name=1,
    cores=1,
    frequency=1,
    price=1
))
```

```
processors.append(proc.Processor(
    article=2,
    producer=2,
    name=2,
    cores=2,
    frequency=2,
    price=1
))
```

```
processors.append(proc.Processor(
    article=3,
    producer=3,
    name=3,
    cores=3,
    frequency=3,
    price=333
))
```

```
processors.append(proc.Processor(
    article=4,
    producer=4,
    name=4,
    cores=4,
    frequency=4,
    price=444
))
```

```
processors.append(proc.Processor(  
    article=5,  
    producer=5,  
    name=5,  
    cores=5,  
    frequency=5,  
    price=555  
))
```

```
processors.append(proc.Processor(  
    article=6,  
    producer=6,  
    name=6,  
    cores=6,  
    frequency=6,  
    price=6  
))
```

```
def add_processor(self):  
    processor = proc.Processor(  
        article=self.editArticle.text(),  
        producer=self.editProducer.text(),  
        name=self.editName.text(),  
        cores=self.editCores.text(),  
        frequency=self.editFrequency.text(),  
        price=self.editPrice.text()  
    )  
    processors.append(processor)  
    self.clear_add_edits()  
  
    self.update_table_content()  
  
def delete_processor(self):  
    article = self.editDeleteArticle.text()  
    for processor in processors:  
        if str(processor.data.get('article')) == str(article):  
            processors.remove(processor)  
    self.editDeleteArticle.clear()  
    self.update_table_content()  
  
def search_processor(self):  
    producer = self.editSearch.text()  
    data = list()  
    for processor in processors:  
        if str(processor.data.get('producer')) == producer:
```

```

        data.append(processor)
self.editSearch.clear()

if len(data) == 0:
    msg_box_no = QtWidgets.QMessageBox.question(self, 'Empty', "There are
no such processors", QMessageBox.Ok)
    if msg_box_no == QMessageBox.Ok:
        pass
    self.update_table_content()
    pass
else:
    self.update_table_content(data)

def print_all(self):
    self.update_table_content()

def init_table(self):
    self.tableResult.setColumnCount(6)
    self.tableResult.setRowCount(1)

    for i, attr in enumerate(proc.Processor.attributes()):
        self.tableResult.setItem(
            0, i, QtWidgets.QTableWidgetItem(attr)
        )

def update_table_content(self, data: List[proc.Processor] = None):
    if data is None:
        data = processors.copy()

    global msg_box
    if len(data) == 0:
        msg_box = QtWidgets.QMessageBox.question(self, 'Empty', "There are no
processors", QMessageBox.Ok)
        if msg_box == QMessageBox.Ok:
            pass

    self.tableResult.clear()

    rows = len(data)
    self.tableResult.setRowCount(rows + 1)

    for i, attr in enumerate(proc.Processor.attributes()):
        self.tableResult.setItem(
            0, i, QtWidgets.QTableWidgetItem(attr)
        )

```

```

if self.checkBoxPrice.isChecked():
    check_list = list()

    for row, processor in enumerate(data):
        if float(data[row].data['price'] > 100):
            check_list.append(processor)

    if len(check_list) == 0:
        msg_box = QtWidgets.QMessageBox.question(self, 'Empty', "There are
no processors", QMessageBox.Ok)
        if msg_box == QMessageBox.Ok:
            pass
        else:
            self.tableResult.clear()
            rows = len(check_list)
            self.tableResult.setRowCount(rows + 1)

            for i, attr in enumerate(proc.Processor.attributes()):
                self.tableResult.setItem(
                    0, i, QtWidgets.QTableWidgetItem(attr)
                )
            for row, processor in enumerate(check_list):
                for column, attribute in enumerate(proc.Processor.attributes()):
                    self.tableResult.setItem(
                        row + 1,
                        column,
                        QtWidgets.QTableWidgetItem(
                            str(check_list[row].data[attribute])
                        )
                    )
            )
        else:
            for row, processor in enumerate(data):
                for column, attribute in enumerate(proc.Processor.attributes()):
                    self.tableResult.setItem(
                        row + 1,
                        column,
                        QtWidgets.QTableWidgetItem(
                            str(data[row].data[attribute])
                        )
                    )
            )

def print_by_combo_box(self):
    index = self.comboBoxCores.currentIndex()
    value = self.comboBoxCores.currentText()

```

```

if index == 0:
    self.update_table_content()
elif index == 1:
    self.print_by_cb_index(value)
elif index == 2:
    self.print_by_cb_index(value)
elif index == 3:
    self.print_by_cb_index(value)

def print_by_cb_index(self, value):
    self.tableResult.clear()

    cores_list = list()

    for row, processor in enumerate(processors):
        if str(processors[row].data['cores']) == value:
            if self.checkBoxPrice.isChecked():
                if float(processors[row].data['price']) > 100:
                    cores_list.append(processor)
            else:
                cores_list.append(processor)

    if len(cores_list) == 0:
        msg_box_cores = QtWidgets.QMessageBox.question(self, 'Empty', "There
are no processors", QMessageBox.Ok)
        if msg_box_cores == QMessageBox.Ok:
            pass

    rows = len(cores_list)
    self.tableResult.setRowCount(rows + 1)

    for i, attr in enumerate(proc.Processor.attributes()):
        self.tableResult.setItem(
            0, i, QtWidgets.QTableWidgetItem(attr)
        )

    for row, processor in enumerate(cores_list):
        for column, attribute in enumerate(proc.Processor.attributes()):
            self.tableResult.setItem(
                row + 1,
                column,
                QtWidgets.QTableWidgetItem(
                    str(cores_list[row].data[attribute])
                )
            )

```



)

```
def clear_add_edits(self):
    self.editArticle.clear()
    self.editProducer.clear()
    self.editName.clear()
    self.editCores.clear()
    self.editFrequency.clear()
    self.editPrice.clear()

def init_menu(self):
    menu = self.menuBar()
    menu.setNativeMenuBar(False)

    self.actionCreate.triggered.connect(self.on_create)
    self.actionOpen.triggered.connect(self.on_open)
    self.actionSave.triggered.connect(self.on_save)
    self.actionSave_as.triggered.connect(self.on_save_as)
    self.actionExit.triggered.connect(self.on_exit)
    self.actionAbout_2.triggered.connect(self.on_about)

def on_create(self):
    self.clear_add_edits()
    self.editDeleteArticle.clear()
    self.editSearch.clear()
    processors.clear()

    self.update_table_content()

def on_open(self):
    path = QtWidgets.QFileDialog.getOpenFileName(
        self,
        'Open file',
        default_path,
        "*.json"
    )[0]

    global user_path_to_save
    user_path_to_save = path

    data = []
    if os.path.isfile(path):
        with open(path, 'r') as f:
            data = json.load(f)
```

```
global processors
processors = list()
```

```
for elem in data:
```

```
    processor = proc.Processor(
        article=elem['article'],
        producer=elem['producer'],
        name=elem['name'],
        cores=elem['cores'],
        frequency=elem['frequency'],
        price=elem['price']
    )
    processors.append(processor)
```

```
self.update_table_content()
```

```
def on_save(self):
```

```
    print("UPTS: " + user_path_to_save)
    if os.path.exists(f'{user_path_to_save}'):
        data = []
```

```
        for p in processors:
            data.append(p.__dict__())
```

```
        with open(f'{user_path_to_save}', "w", encoding='utf-8') as fp:
            json.dump(data, fp)
```

```
    else:
        self.on_save_as()
```

```
def on_save_as(self):
```

```
    path = QtWidgets.QFileDialog.getSaveFileName(
        self,
        'Save file',
        default_path,
        "*.json"
    )[0]
```

```
    if path != "":
        global user_path_to_save
        user_path_to_save = path
```

```
    data = []
```

```
    for p in processors:
        data.append(p.__dict__())
```

```
with open(user_path_to_save, "w", encoding='utf-8') as fp:  
    json.dump(data, fp)
```

```
def on_about(self):  
    about = QtWidgets.QMessageBox.question(self, 'Info', "Vladislav Shorin",  
QMessageBox.Ok)  
    if about == QMessageBox.Ok:  
        pass
```

```
def on_exit(self):  
    exit = QtWidgets.QMessageBox.question(self, 'Exit', "You sure?",  
QMessageBox.Ok, QMessageBox.Abort)  
    if exit == QMessageBox.Ok:  
        QtWidgets.qApp.exit(0)  
    elif exit == QMessageBox.Abort:  
        pass
```