**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П.О.СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

специальности 6-05-0611-01 Информационные системы

и технологии (в игровой индустрии)

ЛАБОРАТОРНАЯ РАБОТА №7

по дисциплине: «Высокоуровневые языки программирования»

на тему: **«Мультимедиа. Печать документов»**

Выполнил: студент гр. ИТИ-21

Важаная Я И

Принял: ассистент

Карась О.В.

Гомель 2024

# **Цель работы:** изучить средства воспроизведения звука и видео с помощью библиотеки PyQt; изучить средства библиотеки PyQt, позволяющих выполнить печать документов, их предварительный просмотр и экспорт в формат PDF.

**Задание:**

Создать приложение, которое при запуске выводит окно с надписью (см. ниже). Поменять значок в заголовке на свой. В заголовке окна указать свои имя, фамилию и группу.

Под надписью должно находится 3 кнопки. При наведении мыши на кнопку должна выводится подсказка.

Кнопки:

1 – «Аудиоплеер». Открывается окно, содержащее кнопки «Открыть файл», «Пауза», «Стоп», «Информация о файле», ползунок громкости.

2 – «Видеоплеер». Открывается окно, содержащее кнопки «Открыть файл», «Пауза», «Стоп», «Информация о файле», ползунок громкости.

3 – «Печать документов». Открывается окно, содержащее следующие кнопки «Открыть файл», «Настройка страницы», «Просмотр», «Печать», «Экспорт в PDF».

**Лабораторная работа №7**

**Мультимедиа. Печать документов.**

Выполнил студент группы ИТ\_-21

Фамилия Имя Отчество

**Ход работы:**

1. Импорт необходимых модулей

Все необходимые модули, нужные для создания окна представлены на рисунке 1.

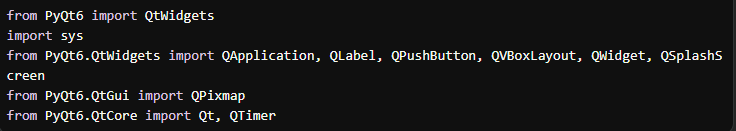


Рисунок 1 – Импорт необходимых модулей

* *PyQt6.QtWidgets*: Содержит все виджеты для создания графического интерфейса.
* *sys*: Используется для работы с системными аргументами и выхода из приложения.
* *QApplication*: Основной класс для управления потоком приложения.
* *QLabel*: Виджет для отображения текстовой информации.
* *QPushButton*: Виджет кнопки.
* *QVBoxLayout*: Макет для вертикального размещения виджетов.
* *QWidget*: Базовый класс для всех виджетов.
* *QPixmap*: Класс для работы с изображениями.

1. Создание главного окна с кнопкой.

На рисунке 2 представлен фрагмент кода для создания макета и добавления виджетов.

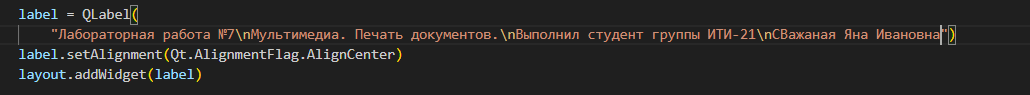


Рисунок 2 – Фрагмент кода для создания макета и добавления виджетов

* Создание макета: layout = *QVBoxLayout*() создает вертикальный макет для размещения виджетов.
* Создание метки: label = *QLabel*(...) создает текстовую метку.
* Добавление метки в макет: *layout.addWidget(label)* добавляет метку в макет.

1. Основной блок запуска приложения

На рисунке 3 представлен фрагмент кода для запуска приложения.

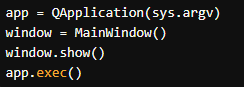


Рисунок 3 – Фрагмент кода для запуска приложения

* Инициализация приложения: *QApplication* запускает приложение.
* Создание главного окна: Создается экземпляр *MainWindow*.
* Запуск основного цикла: *app.exec*() обрабатывает события приложения до его завершения.

1. Результат работы программы

На рисунках 4 представлен результат работы программы (запуск приложения и появление главного окна приложения). В главном меню реализованы 3 функции: «Аудиоплеер» и «Видеоплеер*»* и «Печать документов».

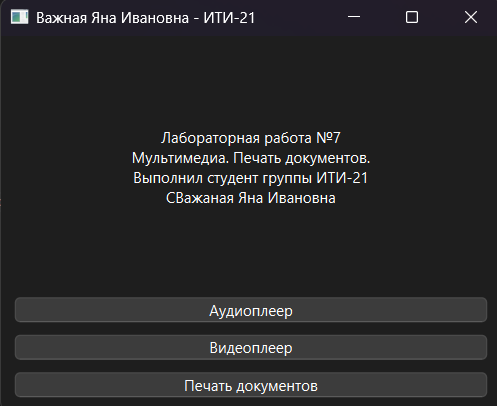


Рисунок 4 – Главное окно приложения

На рисунке 5 показан результат создания аудиоплеера и загрузки в него файла «*audio*.*mp*3». В аудиоплеере можно промотать аудио на 5 секунд вперед или назад, понизить либо же повысить громкость и поставить воспроизведение аудио на паузу.

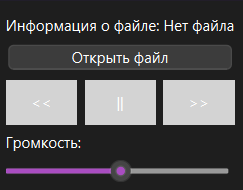


Рисунок 5 – Окно с аудиоплеером

На рисунке 6 показан результат создания видеоплеера и загрузки в него файла «*video*.*mp*3». В видеоплеере можно промотать видео на 5 секунд вперед или назад, понизить либо же повысить громкость и поставить воспроизведение видео на паузу.

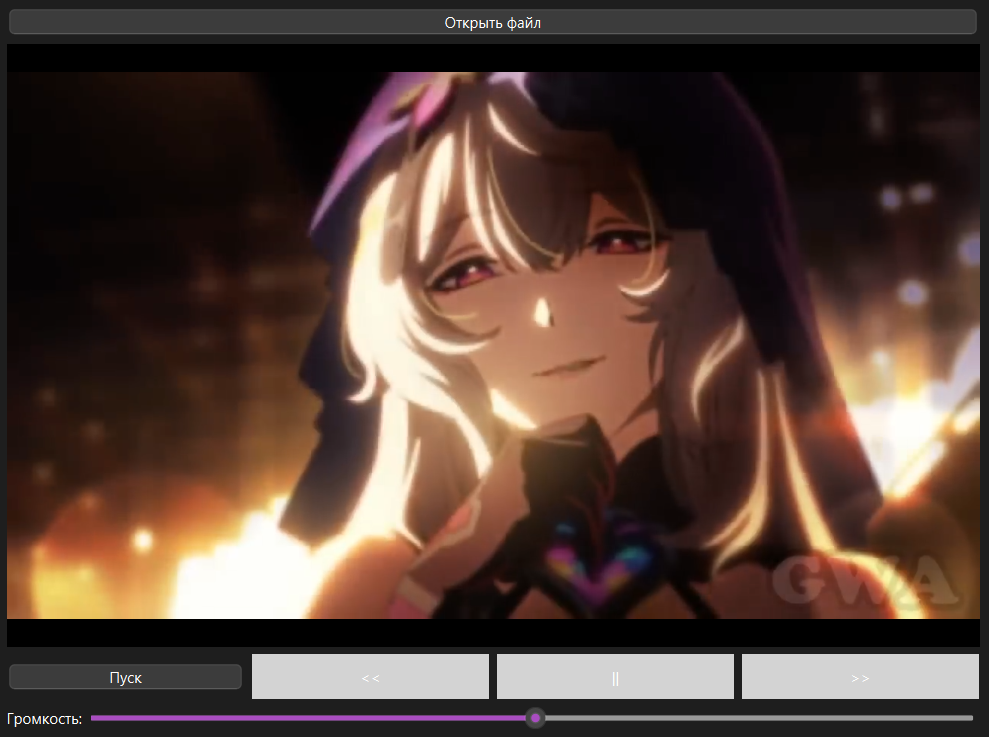


Рисунок 6 – Окно с видеоплеером

На рисунке 7 показано окно для работы с текстовым файлом, предназначенное для печати текстового файла или экспорта файла в *PDF*. Также реализовано окно просмотра содержимого текстового файла. Реализованы функции: «Открыть файл», «Настройка страницы», «Просмотр», «Печать» и «Экспорт в *PDF*».

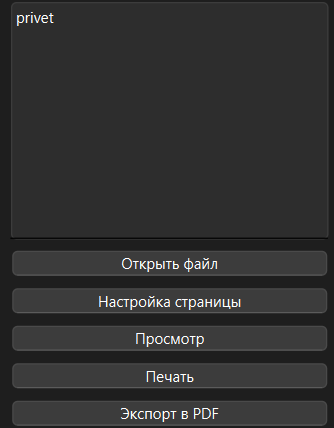


Рисунок 7 – Загруженный текстовый документ и возможные операции над ним

На рисунке 8 показано графическое окно для настройки параметров печати текстового документа.

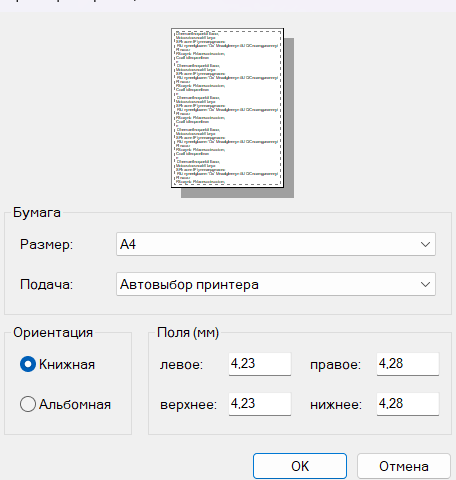


Рисунок 8 – Настройка печати

Листинг кода представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы были изучены средства воспроизведения звука и видео с помощью библиотеки *PyQt;* изучить средства библиотеки *PyQt*, позволяющих выполнить печать документов, их предварительный просмотр и экспорт в формат *PDF*.

**ПРИЛОЖЕНИЕ А  
(обязательное)  
Листинг кода**

*Lab7.py*

import sys

from PyQt5.QtWidgets import (

    QApplication, QMainWindow, QMessageBox, QPushButton, QVBoxLayout,

    QWidget, QColorDialog, QFontDialog, QInputDialog, QDialog, QMenu,

    QMenuBar, QStatusBar, QToolBar, QAction, QSplitter, QLabel, QMdiArea,

    QMdiSubWindow, QPlainTextEdit

)

from PyQt5.QtCore import Qt

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Лабораторная работа 6 - Спиридонов Дмитрий ИТИ-21")

        self.setGeometry(100, 100, 600, 400)

        # Основной виджет

        central\_widget = QWidget()

        self.setCentralWidget(central\_widget)

        # Основной макет

        layout = QVBoxLayout(central\_widget)

        # Кнопка "Диалоговые окна"

        btn\_dialogs = QPushButton("Диалоговые окна")

        btn\_dialogs.setToolTip("Открыть окно с диалогами")

        btn\_dialogs.clicked.connect(self.open\_dialog\_window)

        layout.addWidget(btn\_dialogs)

        # Кнопка "SDI-программа"

        btn\_sdi = QPushButton("SDI-программа")

        btn\_sdi.setToolTip("Открыть SDI программу")

        btn\_sdi.clicked.connect(self.open\_sdi\_program)

        layout.addWidget(btn\_sdi)

        # Кнопка "MDI-программа"

        btn\_mdi = QPushButton("MDI-программа")

        btn\_mdi.setToolTip("Открыть MDI программу")

        btn\_mdi.clicked.connect(self.open\_mdi\_program)

        layout.addWidget(btn\_mdi)

    def open\_dialog\_window(self):

        dialog\_window = DialogWindow(self)

        dialog\_window.exec\_()

    def open\_sdi\_program(self):

        sdi\_window = SDIWindow(self)

        sdi\_window.show()

    def open\_mdi\_program(self):

        mdi\_window = MDIWindow(self)

        mdi\_window.show()

class DialogWindow(QDialog):

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Окно сведений о программе")

        self.setGeometry(100, 100, 400, 200)

        # Основной макет

        layout = QVBoxLayout(self)

        # Текст с информацией

        label = QLabel("ФИО: Спиридонов Дмитрий\nГруппа: ИТИ-21")

        layout.addWidget(label)

        # Кнопка для открытия многостраничного мастера

        btn\_master = QPushButton("Открыть многостраничный мастер")

        btn\_master.clicked.connect(self.open\_master)

        layout.addWidget(btn\_master)

    def open\_master(self):

        master = MasterWindow(self)

        master.exec\_()

class MasterWindow(QDialog):

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Мастер")

        self.setGeometry(100, 100, 400, 300)

        layout = QVBoxLayout(self)

        # Кнопка для открытия окна подтверждения

        btn\_confirm = QPushButton("Открыть окно подтверждения")

        btn\_confirm.clicked.connect(self.open\_confirmation\_dialog)

        layout.addWidget(btn\_confirm)

        # Кнопка для ввода целого числа

        btn\_int\_input = QPushButton("Открыть окно для ввода целого числа")

        btn\_int\_input.clicked.connect(self.open\_int\_input\_dialog)

        layout.addWidget(btn\_int\_input)

        # Кнопка для выбора цвета

        btn\_color = QPushButton("Открыть окно для выбора цвета")

        btn\_color.clicked.connect(self.open\_color\_dialog)

        layout.addWidget(btn\_color)

        # Кнопка для выбора шрифта

        btn\_font = QPushButton("Открыть окно для выбора шрифта")

        btn\_font.clicked.connect(self.open\_font\_dialog)

        layout.addWidget(btn\_font)

    def open\_confirmation\_dialog(self):

        reply = QMessageBox.question(self, "Подтверждение", "Вы уверены?",

                                     QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

        if reply == QMessageBox.Yes:

            print("Подтверждено")

    def open\_int\_input\_dialog(self):

        num, ok = QInputDialog.getInt(self, "Ввод целого числа", "Введите целое число:")

        if ok:

            print(f"Введено число: {num}")

    def open\_color\_dialog(self):

        color = QColorDialog.getColor()

        if color.isValid():

            print(f"Выбран цвет: {color.name()}")

    def open\_font\_dialog(self):

        font, ok = QFontDialog.getFont()

        if ok:

            print(f"Выбран шрифт: {font.family()}")

class SDIWindow(QMainWindow):

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("SDI-программа")

        self.setGeometry(100, 100, 800, 600)

        # Создание меню

        self.menu\_bar = QMenuBar(self)

        self.setMenuBar(self.menu\_bar)

        file\_menu = self.menu\_bar.addMenu("Файл")

        edit\_menu = self.menu\_bar.addMenu("Правка")

        # Добавление действий в меню "Файл"

        new\_action = QAction("Новый", self)

        new\_action.triggered.connect(self.restart\_sdi)

        file\_menu.addAction(new\_action)

        exit\_action = QAction("Выход", self)

        exit\_action.triggered.connect(self.close)

        file\_menu.addAction(exit\_action)

        # Добавление действий в меню "Правка"

        copy\_action = QAction("Копировать", self)

        copy\_action.triggered.connect(self.copy\_text)

        edit\_menu.addAction(copy\_action)

        paste\_action = QAction("Вставить", self)

        paste\_action.triggered.connect(self.paste\_text)

        edit\_menu.addAction(paste\_action)

        change\_orientation\_action = QAction("Изменить ориентацию панелей", self)

        change\_orientation\_action.triggered.connect(self.change\_panel\_orientation)

        edit\_menu.addAction(change\_orientation\_action)

        # Создание панели инструментов

        toolbar = QToolBar("Панель инструментов")

        self.addToolBar(toolbar)

        # Создание панели 1

        self.panel1 = QSplitter(Qt.Horizontal)

        self.panel1.addWidget(QLabel("Панель 1"))

        self.setCentralWidget(self.panel1)

        # Создание кнопок для панели 1

        btn\_console = QPushButton("Вывести сообщение в консоль", self)

        btn\_console.clicked.connect(self.print\_to\_console)

        self.panel1.addWidget(btn\_console)

        # Создание панели 2

        self.panel2 = QSplitter(Qt.Vertical)

        self.panel2.addWidget(QLabel("Панель 2"))

        self.panel1.addWidget(self.panel2)

        # Кнопка для разделения панели

        btn\_split = QPushButton("Разделить панель", self)

        btn\_split.clicked.connect(self.split\_panel)

        self.panel2.addWidget(btn\_split)

        # Панель 3

        self.panel3 = QSplitter(Qt.Horizontal)

        self.panel3.addWidget(QLabel("Панель 3"))

        self.panel2.addWidget(self.panel3)

        # Панель 4

        self.panel4 = QSplitter(Qt.Horizontal)

        self.panel4.addWidget(QLabel("Панель 4"))

        self.panel3.addWidget(self.panel4)

        # Строка состояния

        self.statusBar = QStatusBar(self)

        self.setStatusBar(self.statusBar)

        # Кнопка для изменения состояния меню

        btn\_toggle\_menu = QPushButton("Отключить меню", self)

        btn\_toggle\_menu.clicked.connect(self.toggle\_menu)

        self.panel4.addWidget(btn\_toggle\_menu)

        # Кнопка для вывода/очистки строки состояния

        btn\_toggle\_status = QPushButton("Сообщение в строке состояния", self)

        btn\_toggle\_status.clicked.connect(self.toggle\_status\_message)

        self.panel4.addWidget(btn\_toggle\_status)

    def print\_to\_console(self):

        print("Сообщение выведено в консоль")

    def split\_panel(self):

        # Создаем новый QSplitter и добавляем его в текущую панель

        new\_panel = QSplitter(Qt.Horizontal)

        new\_panel.addWidget(QLabel("Новая панель"))

        self.panel2.addWidget(new\_panel)

    def toggle\_menu(self):

        self.menuBar().setEnabled(not self.menuBar().isEnabled())

    def toggle\_status\_message(self):

        if self.statusBar.currentMessage():

            self.statusBar.clearMessage()

        else:

            self.statusBar.showMessage("Сообщение в строке состояния")

    def restart\_sdi(self):

        # Перезапускаем SDI-программу

        self.close()

        new\_sdi = SDIWindow()

        new\_sdi.show()

    def copy\_text(self):

        print("Текст скопирован")

    def paste\_text(self):

        print("Текст вставлен")

    def change\_panel\_orientation(self):

        # Изменяем ориентацию разделителя между горизонтальной и вертикальной

        if self.panel2.orientation() == Qt.Vertical:

            self.panel2.setOrientation(Qt.Horizontal)

        else:

            self.panel2.setOrientation(Qt.Vertical)

class MDIWindow(QMainWindow):

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("MDI-программа")

        self.setGeometry(100, 100, 800, 600)

        self.mdi\_area = QMdiArea()

        self.setCentralWidget(self.mdi\_area)

        # Меню "Файл"

        menu\_bar = self.menuBar()

        file\_menu = menu\_bar.addMenu("Файл")

        create\_action = QAction("Создать окно", self)

        create\_action.triggered.connect(self.create\_subwindow)

        file\_menu.addAction(create\_action)

        # Меню "Окно"

        window\_menu = menu\_bar.addMenu("Окно")

        cascade\_action = QAction("Расположить каскадом", self)

        cascade\_action.triggered.connect(self.mdi\_area.cascadeSubWindows)

        window\_menu.addAction(cascade\_action)

        tile\_action = QAction("Расположить плиткой", self)

        tile\_action.triggered.connect(self.mdi\_area.tileSubWindows)

        window\_menu.addAction(tile\_action)

        close\_active\_action = QAction("Закрыть активное", self)

        close\_active\_action.triggered.connect(self.mdi\_area.closeActiveSubWindow)

        window\_menu.addAction(close\_active\_action)

        close\_all\_action = QAction("Закрыть все", self)

        close\_all\_action.triggered.connect(self.mdi\_area.closeAllSubWindows)

        window\_menu.addAction(close\_all\_action)

        self.counter = 1

    def create\_subwindow(self):

        sub\_window = QMdiSubWindow()

        text\_edit = QPlainTextEdit()

        text\_edit.setPlainText(f"Окно {self.counter}: Фамилия")

        sub\_window.setWidget(text\_edit)

        sub\_window.setWindowTitle(f"Окно {self.counter}")

        self.mdi\_area.addSubWindow(sub\_window)

        sub\_window.show()

        self.counter += 1

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    main\_window = MainWindow()

    main\_window.show()

    sys.exit(app.exec\_())