

Урок 1: Основы PyQt5. Создание первого окна

Цель урока:

- Установить PyQt5.
 - Понять структуру приложения.
 - Создать простое окно с кнопкой и обработчиком событий.
-

1. Установка PyQt5

Перед началом установим библиотеку:

[bash](#)

[Copy](#)

[Download](#)

[pip install PyQt5 PyQt5-tools](#)

- PyQt5 – основные модули для работы с GUI.
- PyQt5-tools – дополнительные инструменты (включая Qt Designer).

(Если у тебя уже установлено, пропускаем этот шаг.)

2. Базовая структура приложения

Любое PyQt5-приложение состоит из:

1. **QApplication** – главный класс, управляющий циклом событий.
2. **QMainWindow** (или **QWidget**) – основное окно программы.
3. **Виджеты** (кнопки, текстовые поля и т. д.).

Пример 1: Простое окно "Hello, World!"

[python](#)

[Copy](#)

[Download](#)

[from PyQt5.QtWidgets import QApplication, QMainWindow](#)

```
app = QApplication([]) # Создаём приложение
window = QMainWindow() # Главное окно
window.setWindowTitle("Моё первое окно") # Заголовок
window.setGeometry(100, 100, 400, 300) # (x, y, width, height)
window.show()      # Показываем окно
```

```
app.exec_()      # Запускаем цикл событий
```

Что делает код?

- QApplication([]) – создаёт объект приложения (аргумент [] – список параметров командной строки).
- QMainWindow() – создаёт главное окно.
- setWindowTitle() – устанавливает заголовок.
- setGeometry(x, y, w, h) – задаёт положение и размер окна.
- show() – делает окно видимым.
- app.exec_() – запускает бесконечный цикл обработки событий (без него окно закроется сразу).

 **Запусти код и проверь!**

3. Добавляем кнопку и обработчик событий

Теперь добавим кнопку (QPushButton) и свяжем её с функцией через **сигналы и слоты**.

Пример 2: Кнопка с действием

[python](#)

[Copy](#)

[Download](#)

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QLabel
```

```
class MyWindow(QMainWindow):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.setWindowTitle("Урок 1: Кнопка")
```

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

```
        # Создаём кнопку
```

```
        self.button = QPushButton("Нажми меня", self)
```

```
        self.button.setGeometry(150, 100, 100, 40) # (x, y, width, height)
```

```
        self.button.clicked.connect(self.on_button_click) # Привязываем обработчик
```

```
        # Создаём текстовую метку
```

```
        self.label = QLabel("Текст до нажатия", self)
```

```
self.label.setGeometry(150, 150, 200, 30)

def on_button_click(self):
    self.label.setText("Кнопка была нажата!") # Меняем текст метки

app = QApplication([])
window = MyWindow()
window.show()
app.exec_()
```

Разбор кода:

1. class MyWindow(QMainWindow) – создаём своё окно, наследуясь от QMainWindow.
2. super().__init__() – вызываем конструктор родительского класса.
3. self.button = QPushButton(...) – создаём кнопку.
4. button.clicked.connect(...) – привязываем функцию-обработчик к сигналу clicked.
5. self.label = QLabel(...) – создаём текстовую метку.
6. on_button_click(self) – функция, которая выполняется при нажатии кнопки.

◆ Что изменится при запуске?

- Появится окно с кнопкой и текстом.
- При нажатии на кнопку текст изменится.

✍ 4. Размещение виджетов через Layout (QVBoxLayout)

Ручное позиционирование (setGeometry) неудобно при масштабировании. Лучше использовать **Layout'ы** (вертикальный, горизонтальный, сетка).

❖ Пример 3: Вертикальное расположение элементов

python

Copy

Download

```
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QLabel
```

```
class MyWindow(QWidget): # Используем QWidget вместо QMainWindow для простоты
    def __init__(self):
        super().__init__()
```

```
self.setWindowTitle("QVBoxLayout")
self.setGeometry(100, 100, 300, 200)

# Создаём Layout
layout = QVBoxLayout()

# Добавляем виджеты в Layout
self.label = QLabel("Текст сверху")
self.button = QPushButton("Кнопка снизу")
self.button.clicked.connect(lambda: self.label.setText("Кнопка нажата!"))

layout.addWidget(self.label)
layout.addWidget(self.button)

# Устанавливаем Layout для окна
self.setLayout(layout)

app = QApplication([])
window = MyWindow()
window.show()
app.exec_()
```

Что нового?

- QVBoxLayout() – вертикальное расположение виджетов.
- addWidget() – добавляет виджеты в Layout.
- setLayout() – применяет Layout к окну.

◆ Результат:

- Элементы автоматически выстраиваются вертикально.
- При изменении размера окна Layout адаптируется.

⭐ 5. Итоги урока

Сегодня мы изучили:

- Установку PyQt5.
- Создание простого окна (QMainWindow, QWidget).

- Добавление кнопки (QPushButton) и обработчика событий.
 - Использование QVBoxLayout для автоматического размещения элементов.
-

❖ Домашнее задание

1. Создай окно с **двумя** кнопками:
 - Одна меняет цвет фона окна.
 - Вторая выводит сообщение (QMessageBox.information).
2. Добавь **текстовое поле** (QLineEdit) и кнопку, которая выводит введённый текст в консоль.

Пример решения (часть кода):

python

Copy

Download

```
self.line_edit = QLineEdit(self)  
self.button = QPushButton("Вывести текст", self)  
self.button.clicked.connect(lambda: print(self.line_edit.text()))
```

❖ Что дальше?

Следующий урок: «**Работа с диалоговыми окнами (QMessageBox, QFileDialog)**».

Если есть вопросы – задавай! 😊

Проведи следующий урок

Отлично! Переходим ко **второму уроку** по PyQt5, где мы углубимся в **диалоговые окна, меню и работу с файлами**.

⌚ Урок 2: Диалоговые окна, меню и файлы

Цель урока:

1. Научиться использовать стандартные диалоги (QMessageBox, QFileDialog).
 2. Создать меню (QMenuBar) и панель инструментов (QToolBar).
 3. Работать с файлами: открытие, сохранение, чтение.
-

❖ 1. Стандартные диалоговые окна

PyQt5 предоставляет готовые диалоги для удобства.

◆ 1.1 QMessageBox (окно сообщения)

Используется для вывода информации, предупреждений, ошибок и вопросов.

Пример:

python

Copy

Download

```
from PyQt5.QtWidgets import QMessageBox
```

```
# Простое информационное окно
```

```
QMessageBox.information(window, "Успех", "Файл сохранён!")
```

```
# Окно с вопросом (Yes/No)
```

```
answer = QMessageBox.question(window, "Подтверждение", "Вы уверены?",
```

```
    QMessageBox.Yes | QMessageBox.No)
```

```
if answer == QMessageBox.Yes:
```

```
    print("Пользователь согласился")
```

Доступные типы:

- `information()` – информация.
- `warning()` – предупреждение.
- `critical()` – ошибка.
- `question()` – вопрос (да/нет).

◆ **1.2 QFileDialog (работа с файлами)**

Позволяет выбирать файлы для открытия или сохранения.

Пример:

python

Copy

Download

```
from PyQt5.QtWidgets import QFileDialog
```

```
# Открытие файла
```

```
file_path, _ = QFileDialog.getOpenFileName(window, "Выберите файл", "", "Текстовые файлы (*.txt);;Все файлы (*)")
```

```
# Сохранение файла  
save_path, _ = QFileDialog.getSaveFileName(window, "Сохранить файл", "", "Текстовые файлы (*.txt)")  
  
if file_path:  
    with open(file_path, "r", encoding="utf-8") as file:  
        content = file.read()  
        print(content)
```

Методы:

- `getOpenFileName()` – выбор файла для открытия.
- `getSaveFileName()` – выбор пути для сохранения.
- `getExistingDirectory()` – выбор папки.

❖ 2. Создание меню (QMenuBar) и панели инструментов (QToolBar)

◆ 2.1 Меню в QMainWindow

Добавим простое меню с действиями.

Пример:

```
python  
Copy  
Download  
from PyQt5.QtWidgets import QMainWindow, QAction  
  
class MyWindow(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle("Меню и панель инструментов")  
        self.setGeometry(100, 100, 600, 400)  
  
        # Создаём меню  
        menubar = self.menuBar()  
        file_menu = menubar.addMenu("Файл")  
  
        # Добавляем действия (QAction)
```

```
open_action = QAction("Открыть", self)
open_action.triggered.connect(self.open_file)

exit_action = QAction("Выход", self)
exit_action.triggered.connect(self.close)

file_menu.addAction(open_action)
file_menu.addSeparator() # Разделитель
file_menu.addAction(exit_action)

def open_file(self):
    file_path, _ = QFileDialog.getOpenFileName(self, "Открыть файл")
    if file_path:
        print(f"Выбран файл: {file_path}")

app = QApplication([])
window = MyWindow()
window.show()
app.exec_()
```

Что делает код?

- menuBar() – возвращает объект меню.
- addMenu("Файл") – добавляет пункт меню.
- QAction – действие (например, "Открыть файл").
- triggered.connect() – связывает действие с функцией.

◆ 2.2 Панель инструментов (QToolBar)

Добавим кнопки для быстрого доступа.

Пример (дополнение к классу MyWindow):

```
python
Copy
Download
def __init__(self):
```

```

# ... (предыдущий код)

# Создаём панель инструментов
toolbar = self.addToolBar("Инструменты")

# Добавляем кнопки
open_icon = QIcon("folder.png") # Можно использовать свои иконки
open_tool = QAction(open_icon, "Открыть", self)
open_tool.triggered.connect(self.open_file)
toolbar.addAction(open_tool)

# Кнопка выхода
exit_tool = QAction("✖ Выход", self)
exit_tool.triggered.connect(self.close)
toolbar.addAction(exit_tool)

```

Как добавить иконки?

- Используй QIcon("путь_к_изображению.png").
 - Или стандартные иконки Qt (например, QStyle.SP_DialogOpenButton).
-

3. Чтение и запись файлов

Пример: Простой текстовый редактор

python

Copy

Download

```
from PyQt5.QtWidgets import QTextEdit
```

```

class TextEditor(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Текстовый редактор")
        self.setGeometry(100, 100, 800, 600)

```

```
# Текстовое поле
self.text_edit = QTextEdit(self)
self.setCentralWidget(self.text_edit)

# Меню "Файл"
menubar = self.menuBar()
file_menu = menubar.addMenu("Файл")

# Действия
open_action = QAction("Открыть", self)
open_action.triggered.connect(self.open_file)

save_action = QAction("Сохранить", self)
save_action.triggered.connect(self.save_file)

file_menu.addAction(open_action)
file_menu.addAction(save_action)

def open_file(self):
    file_path, _ = QFileDialog.getOpenFileName(self, "Открыть файл")
    if file_path:
        with open(file_path, "r", encoding="utf-8") as file:
            self.text_edit.setText(file.read())

def save_file(self):
    file_path, _ = QFileDialog.getSaveFileName(self, "Сохранить файл")
    if file_path:
        with open(file_path, "w", encoding="utf-8") as file:
            file.write(self.text_edit.toPlainText())

app = QApplication([])
editor = TextEditor()
```

```
editor.show()
```

```
app.exec_()
```

Что получится?

- Программа с текстовым полем.
 - Меню "Файл" → "Открыть" и "Сохранить".
 - Возможность читать и записывать .txt-файлы.
-

📌 4. Итоги урока

✓ Стандартные диалоги:

- QMessageBox – информация, вопросы.
- QFileDialog – работа с файлами.

✓ Меню и панель инструментов:

- QMenuBar – главное меню.
- QToolBar – панель быстрого доступа.

✓ Текстовый редактор:

- Чтение/запись файлов (QTextEdit).
-

📌 Домашнее задание

1. Доработай текстовый редактор:

- Добавь кнопку "Очистить" (self.text_edit.clear()).
- Сделай проверку перед закрытием (если текст изменён, спрашивай "Сохранить?").

2. Создай программу "Заметки":

- Поле для ввода заметки (QTextEdit).
- Кнопка "Добавить" (сохраняет заметку в файл notes.txt).
- Кнопка "Показать все" (читает notes.txt и выводит в QMessageBox).

Пример кода для "Заметок":

```
python
```

```
Copy
```

```
Download
```

```
def save_note(self):
```

```
    note = self.text_edit.toPlainText()
```

```
    if note:
```

```
        with open("notes.txt", "a", encoding="utf-8") as file:
```

```
file.write(note + "\n--\n")
QMessageBox.information(self, "Успех", "Заметка сохранена!")
```

Что дальше?

Третий урок:

- Таблицы (**QTableWidget**) – работа с данными.
- Графика (**QPainter**) – рисование фигур.

Если есть вопросы – спрашивай! 😊

Проведи 3 урок

Отлично! Переходим к **третьему уроку** по PyQt5, где мы научимся работать с таблицами, графикой и создадим мини-проект — «Крестики-нолики» с нуля.

Урок 3: Таблицы, графика и игра «Крестики-нолики»

Цель урока:

1. Работать с табличными данными (**QTableWidget**).
 2. Освоить рисование (**QPainter**).
 3. Создать игру «Крестики-нолики» с графическим интерфейсом.
-

1. Работа с таблицами (**QTableWidget**)

1.1 Создание таблицы

QTableWidget — виджет для отображения и редактирования табличных данных.

Пример: Простая таблица 3x3

python

Copy

Download

```
from PyQt5.QtWidgets import QTableWidget, QTableWidgetItem
```

```
table = QTableWidget(3, 3) # 3 строки, 3 столбца
```

```
table.setHorizontalHeaderLabels(["A", "B", "C"]) # Заголовки столбцов
```

```
# Заполняем ячейки
```

```
table.setItem(0, 0, QTableWidgetItem("Python"))
```

```
table.setItem(1, 1, QTableWidgetItem("C++"))
table.setItem(2, 2, QTableWidgetItem("Java"))
```

```
table.resize(400, 300)
table.show()
```

Методы:

- `setItem(row, col, item)` — вставляет текст в ячейку.
 - `item(row, col)` — получает содержимое ячейки.
 - `setStyleSheet()` — стилизация (как в CSS).
-

◆ 1.2 Обработка кликов в таблице

Пример: Вывод текста ячейки при клике

python

Copy

Download

```
table.cellClicked.connect(self.on_cell_click)
```

```
def on_cell_click(self, row, col):
    item = table.item(row, col)
    if item:
        print(f"Выбрано: {item.text()}")
```

◆ 2. Рисование графики (QPainter)

◆ 2.1 Основы QPainter

Используется для рисования фигур, текста и изображений.

Пример: Рисование прямоугольника и круга

python

Copy

Download

```
from PyQt5.QtWidgets import QWidget
from PyQt5.QtGui import QPainter, QPen, QColor
from PyQt5.QtCore import Qt
```

```
class Canvas(QWidget):  
    def paintEvent(self, event):  
        painter = QPainter(self)  
  
        # Настройки кисти  
        pen = QPen(Qt.blue, 3) # Цвет: синий, толщина: 3px  
        painter.setPen(pen)  
  
        # Рисуем прямоугольник  
        painter.drawRect(50, 50, 200, 100)  
  
        # Рисуем круг  
        painter.setBrush(Qt.red) # Заливка красным  
        painter.drawEllipse(300, 50, 100, 100)  
  
    # Использование:  
    app = QApplication([])  
    canvas = Canvas()  
    canvas.resize(500, 300)  
    canvas.show()  
    app.exec_()
```

Что можно рисовать:

- drawLine() — линия.
- drawRect() — прямоугольник.
- drawEllipse() — круг/эллипс.
- drawText() — текст.

◆ 2.2 Обработка событий мыши

Пример: Рисование мышью

python

Copy

Download

```
class DrawingApp(QWidget):  
  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle("Рисовалка")  
        self.setGeometry(100, 100, 600, 400)  
        self.points = []  
  
    def mousePressEvent(self, event):  
        self.points.append(event.pos())  
        self.update() # Перерисовываем окно  
  
    def paintEvent(self, event):  
        painter = QPainter(self)  
        painter.setPen(QPen(Qt.black, 5))  
  
        # Рисуем линии между точками  
        for i in range(1, len(self.points)):  
            painter.drawLine(self.points[i-1], self.points[i])
```

📌 3. Проект: «Крестики-нолики»

◆ 3.1 Интерфейс игры

Создадим поле 3x3 с кнопками и логикой победы.

Код:

python

Copy

Download

```
from PyQt5.QtWidgets import QMainWindow, QPushButton, QMessageBox, QVBoxLayout, QWidget
```

```
class TicTacToe(QMainWindow):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
self.setWindowTitle("Крестики-нолики")
self.setGeometry(100, 100, 300, 300)

self.current_player = "X"
self.board = [""] * 9 # Поле 3x3

# Создаём кнопки
self.buttons = []
layout = QVBoxLayout()

for i in range(9):
    button = QPushButton("")
    button.setStyleSheet("font-size: 24px;")
    button.clicked.connect(lambda _, idx=i: self.on_click(idx))
    self.buttons.append(button)
    layout.addWidget(button)

container = QWidget()
container.setLayout(layout)
self.setCentralWidget(container)

def on_click(self, idx):
    if not self.board[idx]:
        self.board[idx] = self.current_player
        self.buttons[idx].setText(self.current_player)

    if self.check_winner():
        QMessageBox.information(self, "Победа!", f"Игрок {self.current_player} победил!")
        self.reset_game()

    elif "" not in self.board:
        QMessageBox.information(self, "Ничья!", "Игра окончена!")
        self.reset_game()
```

```

else:
    self.current_player = "O" if self.current_player == "X" else "X"

def check_winner(self):
    # Проверяем строки, столбцы и диагонали
    lines = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8], # Горизонтали
        [0, 3, 6], [1, 4, 7], [2, 5, 8], # Вертикали
        [0, 4, 8], [2, 4, 6]      # Диагонали
    ]
    for line in lines:
        if self.board[line[0]] == self.board[line[1]] == self.board[line[2]] != "":
            return True
    return False

def reset_game(self):
    self.current_player = "X"
    self.board = [""] * 9
    for button in self.buttons:
        button.setText("")

app = QApplication([])
game = TicTacToe()
game.show()
app.exec_()

```

Логика игры:

- Игроки по очереди ставят X и O.
- Проверка победы по линиям.
- Рестарт игры при победе или ничье.



Итоги урока

- Таблицы:** Работа с QTableWidgetItem.
 - Графика:** Рисование через QPainter.
 - Игра:** Полнценные «Крестики-нолики» с GUI.
-

Домашнее задание

1. Улучши игру:
 - Добавь счётчик побед для каждого игрока.
 - Сделай кнопку «Новая игра».
2. Создай «Змейку»:
 - Используй QPainter для отрисовки.
 - Управление стрелками клавиатуры.

Подсказка для «Змейки»:

```
python
Copy
Download
def keyPressEvent(self, event):
    if event.key() == Qt.Key_Left:
        print("Движение влево")
```

Что дальше?

Четвёртый урок:

- **Многопоточность (QThread)** — чтобы интерфейс не зависал.
- **Базы данных (QtSql)** — сохранение результатов игр.

Если есть вопросы — задавай! 😊

Давай 4 урок

Отлично! Переходим к **четвертому уроку** по PyQt5, где мы разберем **многопоточность, базы данных и создадим приложение для учета задач**.

Урок 4: Многопоточность, базы данных и To-Do приложение

Цель урока:

1. Научиться работать с потоками (QThread).
2. Подключать базы данных (QtSql).
3. Создать приложение для учета задач (To-Do List).

◆ 1. Многопоточность (QThread)

◆ 1.1 Зачем нужны потоки?

Главный поток (GUI) не должен блокироваться долгими операциями (например, загрузкой файлов или вычислениями). Для этого используют QThread.

◆ 1.2 Пример: Загрузка файла в отдельном потоке

python

Copy

Download

```
from PyQt5.QtCore import QThread, pyqtSignal  
from PyQt5.QtWidgets import QPushButton, QVBoxLayout, QWidget, QProgressBar
```

```
class Worker(QThread):  
    progress_updated = pyqtSignal(int) # Сигнал для обновления прогресса  
    finished = pyqtSignal()          # Сигнал завершения
```

```
def run(self):  
    for i in range(1, 101):  
        self.msleep(50) # Имитация долгой операции  
        self.progress_updated.emit(i)  
    self.finished.emit()
```

```
class MainWindow(QWidget):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle("Многопоточность")  
        self.setGeometry(100, 100, 300, 150)
```

```
        self.progress = QProgressBar(self)  
        self.button = QPushButton("Начать загрузку", self)  
        self.button.clicked.connect(self.start_loading)
```

```
layout = QVBoxLayout()
layout.addWidget(self.progress)
layout.addWidget(self.button)
self.setLayout(layout)

def start_loading(self):
    self.worker = Worker()
    self.worker.progress_updated.connect(self.progress.setValue)
    self.worker.finished.connect(lambda: self.button.setEnabled(True))
    self.button.setEnabled(False)
    self.worker.start()

app = QApplication([])
window = MainWindow()
window.show()
app.exec_()
```

Разбор кода:

- `Worker(QThread)` — класс потока.
- `progress_updated` и `finished` — сигналы для связи с GUI.
- `run()` — метод, выполняющийся в отдельном потоке.
- `self.worker.start()` — запуск потока.

Важно:

- Не изменяйте виджеты напрямую из потока! Используйте сигналы.
- Управляйте потоком через `start()`, `quit()`, `wait()`.

❖ 2. Базы данных (QtSql)

◆ 2.1 Подключение SQLite

SQLite — легкая встраиваемая БД, не требующая сервера.

Пример: Создание таблицы и запись данных

[python](#)

[Copy](#)

[Download](#)

```
from PyQt5.QtSql import QSqlDatabase, QSqlQuery

# Подключение БД
db = QSqlDatabase.addDatabase("QSQLITE")
db.setDatabaseName("tasks.db") # Файл БД
if not db.open():
    print("Ошибка подключения к БД!")
    exit()

# Создание таблицы
query = QSqlQuery()
query.exec("""
CREATE TABLE IF NOT EXISTS tasks (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    description TEXT NOT NULL,
    status BOOLEAN DEFAULT 0
)
""")

# Добавление задачи
query.prepare("INSERT INTO tasks (description) VALUES (?)")
query.addBindValue("Купить молоко")
query.exec()

# Чтение задач
query.exec("SELECT * FROM tasks")
while query.next():
    print(query.value("description"), query.value("status"))


```

◆ **2.2 Отображение данных в QTableWidget**

python

Copy

Download

```
from PyQt5.QtWidgets import QTableWidget

# ... (код подключения БД)

table = QTableWidget()
table.setColumnCount(2)
table.setHorizontalHeaderLabels(["Описание", "Статус"])

query.exec("SELECT * FROM tasks")
table.setRowCount(0)
while query.next():

    row = table.rowCount()
    table.insertRow(row)
    table.setItem(row, 0, QTableWidgetItem(query.value("description")))
    table.setItem(row, 1, QTableWidgetItem("✓" if query.value("status") else "X"))
```

3. Проект: To-Do List

◆ 3.1 Интерфейс приложения

python

Copy

Download

```
from PyQt5.QtWidgets import (QMainWindow, QLineEdit, QPushButton,
```

```
    QListWidget, QVBoxLayout, QWidget)
```

```
class ToDoApp(QMainWindow):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.setWindowTitle("To-Do List")
```

```
        self.setGeometry(100, 100, 400, 500)
```

```
# Виджеты
```

```
self.task_input = QLineEdit()
self.add_button = QPushButton("Добавить")
self.task_list = QListWidget()

# Разметка
layout = QVBoxLayout()
layout.addWidget(self.task_input)
layout.addWidget(self.add_button)
layout.addWidget(self.task_list)

container = QWidget()
container.setLayout(layout)
self.setCentralWidget(container)

# Подключение БД
self.init_db()
self.load_tasks()

# Сигналы
self.add_button.clicked.connect(self.add_task)
self.task_list.itemDoubleClicked.connect(self.toggle_task)

def init_db(self):
    self.db = QSqlDatabase.addDatabase("QSQLITE")
    self.db.setDatabaseName("tasks.db")
    if not self.db.open():
        QMessageBox.critical(self, "Ошибка", "Не удалось открыть БД!")
        self.close()

query = QSqlQuery()
query.exec("""
CREATE TABLE IF NOT EXISTS tasks (

```

```
        id INTEGER PRIMARY KEY AUTOINCREMENT,  
        description TEXT,  
        status BOOLEAN DEFAULT 0  
    )  
    """)  
  
def load_tasks(self):  
    self.task_list.clear()  
    query = QSqlQuery("SELECT * FROM tasks")  
    while query.next():  
        item = QListWidgetItem(query.value("description"))  
        item.setCheckState(2 if query.value("status") else 0)  
        self.task_list.addItem(item)  
  
def add_task(self):  
    description = self.task_input.text()  
    if description:  
        query = QSqlQuery()  
        query.prepare("INSERT INTO tasks (description) VALUES (?)")  
        query.addBindValue(description)  
        query.exec()  
        self.load_tasks()  
        self.task_input.clear()  
  
def toggle_task(self, item):  
    description = item.text()  
    query = QSqlQuery()  
    query.prepare("UPDATE tasks SET status = NOT status WHERE description = ?")  
    query.addBindValue(description)  
    query.exec()  
    self.load_tasks()
```

```
app = QApplication([])  
todo = ToDoApp()  
todo.show()  
app.exec_()
```

Функционал:

- Добавление задач в БД.
 - Отметка выполненных задач (двойной клик).
 - Автоматическая загрузка при запуске.
-

📌 Итоги урока

- Многопоточность:** Работа с QThread и сигналами.
 - Базы данных:** Подключение SQLite и выполнение запросов.
 - To-Do List:** Полноценное приложение с хранением задач.
-

📌 Домашнее задание

1. **Доработай To-Do List:**
 - Добавь кнопку удаления задач.
 - Реализуй фильтрацию: «Все»/«Выполненные»/«Активные».
2. **Создай «Погодное приложение»:**
 - Используй QThread для запроса к API (например, OpenWeatherMap).
 - Отображай температуру и иконку погоды.

Пример API-запроса:

```
python  
Copy  
Download  
  
import requests  
  
response =  
requests.get("https://api.openweathermap.org/data/2.5/weather?q=Москва&appid=YOUR_API_KEY")  
  
data = response.json()  
  
print(data["main"]["temp"])
```

📌 Что дальше?

Пятый урок:

- Сетевые запросы (**QNetworkAccessManager**).
- Графики (**QChart**) — визуализация данных.

Если есть вопросы — задавай! 😊

Давай еще один урок

🎯 Урок 5: Сетевые запросы, графики и создание погодного приложения

Отлично! В этом уроке мы научимся:

1. Делать HTTP-запросы с помощью **QNetworkAccessManager**
 2. Строить графики данных с использованием **QChart**
 3. Создадим полноценное погодное приложение
-

⭐ 1. Сетевые запросы (**QNetworkAccessManager**)

◆ 1.1 Основы работы с сетью

PyQt5 предоставляет **QNetworkAccessManager** для выполнения сетевых запросов. Это более правильный способ, чем использование `requests`, так как он интегрирован в event loop Qt.

Пример: GET-запрос к API

python

Copy

Download

```
from PyQt5.QtNetwork import QNetworkAccessManager, QNetworkRequest  
from PyQt5.QtCore import QUrl
```

class NetworkExample:

```
def __init__(self):  
    self.manager = QNetworkAccessManager()  
    self.manager.finished.connect(self.handle_response)
```

```
def make_request(self):  
    url = QUrl("https://api.example.com/data")  
    request = QNetworkRequest(url)  
    self.manager.get(request)
```

```
def handle_response(self, reply):
```

```
data = reply.readAll().data().decode()
```

```
print("Ответ сервера:", data)
```

◆ **1.2 Обработка JSON-ответов**

Для работы с JSON добавим стандартный модуль Python:

```
python
```

Copy

Download

```
import json
```

```
def handle_response(self, reply):
```

```
    data = json.loads(reply.readAll().data().decode())
```

```
    print("Температура:", data["main"]["temp"])
```

✍ **2. Работа с графиками (QChart)**

◆ **2.1 Установка необходимых модулей**

Для работы с графиками нужно установить:

```
bash
```

Copy

Download

```
pip install PyQtChart
```

◆ **2.2 Простой линейный график**

```
python
```

Copy

Download

```
from PyQt5.QtChart import QChart, QChartView, QLineSeries
```

```
from PyQt5.QtCore import Qt
```

```
def create_chart():
```

```
    series = QLineSeries()
```

```
    series.append(0, 6)
```

```
    series.append(2, 4)
```

```
    series.append(3, 8)
```

```
series.append(7, 4)
series.append(10, 5)

chart = QChart()
chart.addSeries(series)
chart.createDefaultAxes()

chart_view = QChartView(chart)
chart_view.setRenderHint(QPainter.Antialiasing)
return chart_view

# Добавление в окно:
chart = create_chart()
layout.addWidget(chart)
```

❸ 3. Погодное приложение

◆ 3.1 Полный код приложения

```
python
Copy
Download
import sys
import json
from PyQt5.QtWidgets import (QApplication, QMainWindow, QLabel,
                             QLineEdit, QPushButton, QVBoxLayout, QWidget)
from PyQt5.QtNetwork import QNetworkAccessManager, QNetworkRequest
from PyQt5.QtCore import QUrl
from PyQt5.QtGui import QPixmap
from PyQt5.QtChart import QChart, QChartView, QLineSeries

class WeatherApp(QMainWindow):
    def __init__(self):
        super().__init__()
```

```
self.setWindowTitle("Погодное приложение")
self.setGeometry(100, 100, 400, 500)

# Виджеты
self.city_input = QLineEdit()
self.search_button = QPushButton("Поиск")
self.temp_label = QLabel("Температура: ")
self.weather_icon = QLabel()
self.forecast_chart = QChartView()

# Разметка
layout = QVBoxLayout()
layout.addWidget(self.city_input)
layout.addWidget(self.search_button)
layout.addWidget(self.temp_label)
layout.addWidget(self.weather_icon)
layout.addWidget(self.forecast_chart)

container = QWidget()
container.setLayout(layout)
self.setCentralWidget(container)

# Сетевой менеджер
self.net_manager = QNetworkAccessManager()
self.net_manager.finished.connect(self.handle_weather_response)

# Обработчики
self.search_button.clicked.connect(self.get_weather)

def get_weather(self):
    city = self.city_input.text()
    if city:
```

```
url =
QUrl(f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid=ВАШ_АПИ_КЛЮЧ&units=metric")
request = QNetworkRequest(url)
self.net_manager.get(request)

def handle_weather_response(self, reply):
    data = json.loads(reply.readAll().data().decode())
    temp = data["main"]["temp"]
    icon_code = data["weather"][0]["icon"]

    self.temp_label.setText(f"Температура: {temp}°C")

    # Загрузка иконки погоды
    pixmap = QPixmap()

    pixmap.loadFromData(self.download_image(f"https://openweathermap.org/img/wn/{icon_code}@2x.png"))
    self.weather_icon.setPixmap(pixmap)

    # Получение прогноза для графика
    self.get_forecast()

def get_forecast(self):
    city = self.city_input.text()
    url =
QUrl(f"https://api.openweathermap.org/data/2.5/forecast?q={city}&appid=ВАШ_АПИ_КЛЮЧ&units=metric")
    request = QNetworkRequest(url)
    self.net_manager.finished.connect(self.handle_forecast_response)
    self.net_manager.get(request)

def handle_forecast_response(self, reply):
    data = json.loads(reply.readAll().data().decode())
```

```

series = QLineSeries()

for i, entry in enumerate(data["list"][:8]): # Берем первые 8 периодов
    series.append(i, entry["main"]["temp"])

chart = QChart()
chart.addSeries(series)
chart.createDefaultAxes()
chart.setTitle("Прогноз температуры на 24 часа")

self.forecast_chart.setChart(chart)

def download_image(self, url):
    request = QNetworkRequest(QUrl(url))
    reply = self.net_manager.get(request)

    # Ждем завершения запроса (в реальном приложении нужно использовать сигналы)
    loop = QEventLoop()
    reply.finished.connect(loop.quit)
    loop.exec_()

    return reply.readAll()

```

```

app = QApplication(sys.argv)
weather_app = WeatherApp()
weather_app.show()
sys.exit(app.exec_())

```

◆ 3.2 Ключевые компоненты приложения

1. Ввод города - QLineEdit для поиска
2. Отображение данных:
 - Текущая температура (QLabel)
 - Иконка погоды (QLabel + QPixmap)

- График прогноза (QChartView)

3. Сетевые запросы:

- Текущая погода
 - Прогноз на 24 часа
 - Загрузка иконки
-

Домашнее задание

1. Улучшите погодное приложение:

- Добавьте отображение влажности и скорости ветра
- Реализуйте выбор единиц измерения (Celsius/Fahrenheit)

2. Создайте приложение для отслеживания курсов валют:

- Используйте API Центробанка или другого источника
 - Отобразите данные в виде таблицы и графика
-

Что дальше?

В следующем уроке мы:

1. Разберем **интернационализацию приложений** (перевод на разные языки)
2. Научимся создавать **собственные виджеты**
3. Создадим **кастомный графический редактор**

Если есть вопросы по текущему уроку - задавайте! 😊