Qlabel

QLabel - это класс в PyQt6, который представляет собой виджет отображения текста или изображения. QLabel используется для отображения статического текста или графики на экране.

Основные методы и атрибуты

1. **Создание и конфигурация QLabel**:
   - `QLabel(QWidget *parent = None)`: Создает объект QLabel с указанным родительским виджетом (или без родителя).
   - `QLabel(const QString &text, QWidget *parent = None)`: Создает объект QLabel с указанным текстом.
2. **Управление текстом**:
   - `setText(const QString &text)`: Устанавливает текст, отображаемый на метке.
   - `text() const`: Возвращает текущий текст.
   - `clear()`: Очищает текст на метке.
3. **Управление выравниванием**:
   - `setAlignment(Qt.Alignment)`: Устанавливает выравнивание текста или изображения.
   - `alignment() const`: Возвращает текущее выравнивание.
4. **Настройка форматирования текста**:
   - `setWordWrap(bool on)`: Включает или выключает перенос слов.
   - `wordWrap() const`: Возвращает `true`, если перенос слов включен.
   - `setIndent(int)`: Устанавливает отступ для текста.
   - `indent() const`: Возвращает текущий отступ.
5. **Управление изображением**:
   - `setPixmap(const QPixmap &pixmap)`: Устанавливает изображение (QPixmap) на метке.
   - `pixmap() const`: Возвращает текущее изображение.
   - `setScaledContents(bool)`: Указывает, должен ли QLabel масштабировать содержимое, чтобы соответствовать его размеру.
   - `hasScaledContents() const`: Возвращает `true`, если содержимое масштабируется.
6. **Управление стилем**:
   - `setStyleSheet(const QString &styleSheet)`: Устанавливает стиль для метки с использованием стилей CSS.
7. **Сигналы**:
   - QLabel сам по себе не имеет сигналов, но вы можете обрабатывать события, наследуя QLabel и переопределяя методы событий.

QpushButton

QPushButton - это класс в PyQt6, который представляет кнопку, которую можно нажимать для выполнения определенного действия. QPushButton является основным элементом управления, который используется для запуска команд или взаимодействия с пользователем.

Основные методы и атрибуты

1. **Создание и конфигурация QPushButton**:
   - `QPushButton(QWidget *parent = None)`: Создает объект QPushButton с указанным родительским виджетом (или без родителя).
   - `QPushButton(const QString &text, QWidget *parent = None)`: Создает объект QPushButton с указанным текстом.
   - `QPushButton(const QIcon &icon, const QString &text, QWidget *parent = None)`: Создает объект QPushButton с указанными значком и текстом.
2. **Управление текстом**:
   - `setText(const QString &text)`: Устанавливает текст, отображаемый на кнопке.
   - `text() const`: Возвращает текущий текст.
3. **Управление значком**:
   - `setIcon(const QIcon &icon)`: Устанавливает значок для кнопки.
   - `icon() const`: Возвращает текущий значок.
4. **Управление состоянием**:
   - `setCheckable(bool)`: Делает кнопку переключаемой (включение/выключение).
   - `isCheckable() const`: Возвращает `true`, если кнопка переключаемая.
   - `setChecked(bool)`: Устанавливает состояние переключаемой кнопки.
   - `isChecked() const`: Возвращает `true`, если кнопка находится в состоянии "включено".
   - `toggle()`: Переключает состояние переключаемой кнопки.
5. **Сигналы**:
   - `clicked(bool checked = False)`: Сигнал, который срабатывает при нажатии на кнопку.
   - `pressed()`: Сигнал, который срабатывает при нажатии на кнопку (без отпускания).
   - `released()`: Сигнал, который срабатывает при отпускании кнопки.
   - `toggled(bool checked)`: Сигнал, который срабатывает при изменении состояния переключаемой кнопки.
6. **Настройка внешнего вида**:
   - `setFlat(bool)`: Делает кнопку плоской (без рельефа).
   - `isFlat() const`: Возвращает `true`, если кнопка плоская.

QCheckBox

QCheckBox - это класс в PyQt6, который предоставляет флажок, позволяющий пользователю выбирать или снимать выбор опции. Флажок может находиться в одном из трех состояний: неотмеченный, отмеченный и неопрятный (трехсостояние).

Основные методы и атрибуты

1. **Создание и конфигурация QCheckBox**:
   - `QCheckBox(QWidget *parent = None)`: Создает объект QCheckBox с указанным родительским виджетом (или без родителя).

    - `QCheckBox(const QString &text, QWidget *parent = None)`: Создает объект QCheckBox с указанным текстом.
2. **Управление состоянием**:
    - `setChecked(bool)`: Устанавливает состояние флажка (отмечен/не отмечен).
    - `isChecked() const`: Возвращает `true`, если флажок отмечен.
    - `toggle()`: Переключает состояние флажка.
3. **Настройка текста**:
    - `setText(const QString &text)`: Устанавливает текст, отображаемый рядом с флажком.
    - `text() const`: Возвращает текущий текст.
4. **Трехсостояние**:
    - `setTristate(bool = true)`: Включает или отключает трехсостояние для флажка.
    - `isTristate() const`: Возвращает `true`, если трехсостояние включено.
    - `setCheckState(Qt.CheckState state)`: Устанавливает состояние флажка (Unchecked, PartiallyChecked, Checked).
    - `checkState() const`: Возвращает текущее состояние флажка.
5. **Сигналы**:
    - `stateChanged(int)`: Сигнал, который срабатывает при изменении состояния флажка.
    - `toggled(bool)`: Сигнал, который срабатывает при переключении флажка.
6. **Настройка внешнего вида**:
    - `setIcon(const QIcon &icon)`: Устанавливает значок рядом с текстом флажка.
    - `setIconSize(const QSize &size)`: Устанавливает размер значка.
    - `icon() const`: Возвращает текущий значок.

QRadioButton

QRadioButton - это класс в PyQt6, который предоставляет переключатель, который позволяет пользователю выбрать одну из нескольких опций. Переключатели обычно используются в группах, где только один переключатель может быть выбран в одно и то же время.

Основные методы и атрибуты

1. **Создание и конфигурация QRadioButton**:
    - `QRadioButton(QWidget *parent = None)`: Создает объект QRadioButton с указанным родительским виджетом (или без родителя).
    - `QRadioButton(const QString &text, QWidget *parent = None)`: Создает объект QRadioButton с указанным текстом.
2. **Управление состоянием**:
    - `setChecked(bool)`: Устанавливает состояние переключателя (выбран/не выбран).
    - `isChecked() const`: Возвращает `true`, если переключатель выбран.
    - `toggle()`: Переключает состояние переключателя.
3. **Настройка текста**:

- `setText(const QString &text)`: Устанавливает текст, отображаемый рядом с переключателем.
- `text() const`: Возвращает текущий текст.

4. **Сигналы**:
   - `toggled(bool)`: Сигнал, который срабатывает при изменении состояния переключателя.
   - `clicked(bool)`: Сигнал, который срабатывает при клике на переключатель.
5. **Настройка внешнего вида**:
   - `setIcon(const QIcon &icon)`: Устанавливает значок рядом с текстом переключателя.
   - `setIconSize(const QSize &size)`: Устанавливает размер значка.
   - `icon() const`: Возвращает текущий значок.
6. **Группировка переключателей**:
   - Переключатели обычно группируются с помощью `QButtonGroup`, чтобы обеспечить возможность выбора только одного из них в группе.

```
self.radio_button1.toggled.connect(self.on_radio_button_toggled)
button.text()
```

QGroupBox

QGroupBox - это виджет контейнера в PyQt6, который используется для группировки других виджетов внутри рамки с заголовком. Он помогает организовать элементы интерфейса логически и визуально.

Основные методы и атрибуты

1. **Создание QGroupBox**:
   - `QGroupBox(title, parent=None)`: Создаёт группу с заголовком `title`. Параметр `parent` указывает родительский виджет.
2. **Управление заголовком**:
   - `QGroupBox.setTitle(title)`: Устанавливает заголовок группы.
   - `QGroupBox.title()`: Возвращает текущий заголовок группы.
3. **Управление флажком**:
   - `QGroupBox.setCheckable(state)`: Делает группу отмечаемой (имеющей флажок).
   - `QGroupBox.isCheckable()`: Возвращает `True`, если группа отмечаемая.
   - `QGroupBox.setChecked(state)`: Устанавливает состояние флажка.
   - `QGroupBox.isChecked()`: Возвращает `True`, если флажок установлен.
4. **Сигналы**:
   - `QGroupBox.toggled(state)`: Этот сигнал срабатывает, когда состояние флажка изменяется. Он передаёт текущее состояние (`True` или `False`) как параметр.

QSlider

QSlider - это класс в PyQt6, который предоставляет ползунок (слайдер) для выбора значения из диапазона. Ползунок может быть горизонтальным или вертикальным, в зависимости от ориентации.

Основные методы и атрибуты

1. **Создание QSlider**:
   - `QSlider(Qt.Orientation orientation, QWidget parent=None)`: Создает ползунок с заданной ориентацией. `orientation` может быть `Qt.Orientation.Horizontal` или `Qt.Orientation.Vertical`. Параметр `parent` указывает родительский виджет.
2. **Управление значениями**:
   - `QSlider.setValue(int value)`: Устанавливает текущее значение.
   - `QSlider.value()`: Возвращает текущее значение.
   - `QSlider.setMinimum(int min)`: Устанавливает минимальное значение.
   - `QSlider.setMaximum(int max)`: Устанавливает максимальное значение.
   - `QSlider.setRange(int min, int max)`: Устанавливает минимальное и максимальное значения.
   - `QSlider.setSingleStep(int step)`: Устанавливает шаг изменения значения.
   - `QSlider.setPageStep(int step)`: Устанавливает шаг изменения значения при прокрутке страницы.
3. **Управление видом**:
   - `QSlider.setTickPosition(QSlider.TickPosition position)`: Устанавливает позицию меток (делений) на ползунке. Позиции могут быть `QSlider.NoTicks`, `QSlider.TicksAbove`, `QSlider.TicksBelow`, `QSlider.TicksBothSides`, `QSlider.TicksLeft`, `QSlider.TicksRight`.
   - `QSlider.setTickInterval(int interval)`: Устанавливает интервал между метками.
4. **Сигналы**:
   - `QSlider.valueChanged(int value)`: Сигнал, который срабатывает при изменении значения ползунка.
   - `QSlider.sliderPressed()`: Сигнал, который срабатывает при нажатии на ползунок.
   - `QSlider.sliderReleased()`: Сигнал, который срабатывает при отпускании ползунка.

```
def on_value_changed(self):
    sender = self.sender()
    value = sender.value()
    self.label.setText(f"Value: {value}")
```

QScrollBar

QScrollBar - это класс в PyQt6, который предоставляет функциональность прокрутки содержимого. Прокрутка может быть вертикальной или горизонтальной, в зависимости от ориентации.

Основные методы и атрибуты

1. **Создание QScrollBar**:
    - `QScrollBar(Qt.Orientation orientation, QWidget parent=None)`: Создает полосу прокрутки с заданной ориентацией. `orientation` может быть `Qt.Orientation.Horizontal` или `Qt.Orientation.Vertical`. Параметр `parent` указывает родительский виджет.
2. **Управление значениями**:
    - `QScrollBar.setValue(int value)`: Устанавливает текущее значение.
    - `QScrollBar.value()`: Возвращает текущее значение.
    - `QScrollBar.setMinimum(int min)`: Устанавливает минимальное значение.
    - `QScrollBar.setMaximum(int max)`: Устанавливает максимальное значение.
    - `QScrollBar.setRange(int min, int max)`: Устанавливает минимальное и максимальное значения.
    - `QScrollBar.setSingleStep(int step)`: Устанавливает шаг прокрутки.
    - `QScrollBar.setPageStep(int step)`: Устанавливает шаг прокрутки страницы.
3. **Сигналы**:
    - `QScrollBar.valueChanged(int value)`: Сигнал, который срабатывает при изменении значения полосы прокрутки.

```
QScrollBar(Qt.Orientation.Vertical)
self.v_scrollbar.setRange(0, 100)
self.v_scrollbar.setSingleStep(1)
self.v_scrollbar.setPageStep(10)
self.v_scrollbar.valueChanged.connect(self.on_value_changed)
def on_value_changed(self):
    value = self.sender().value()
    self.label.setText(f"Value: {value}")
```

QLineEdit

QLineEdit - это класс в PyQt6, который предоставляет однострочное текстовое поле для ввода и редактирования текста. Он широко используется для получения текстового ввода от пользователя.

Основные методы и атрибуты

1. **Создание и конфигурация QLineEdit**:
    - `QLineEdit(QWidget *parent = nullptr)`: Создает объект QLineEdit с указанным родительским виджетом (или без родителя).
    - `QLineEdit(const QString &contents, QWidget *parent = nullptr)`: Создает объект QLineEdit с указанным текстом.
2. **Управление текстом**:
    - `setText(const QString &text)`: Устанавливает текст в поле.
    - `text() const`: Возвращает текущий текст.
    - `clear()`: Очищает текстовое поле.
    - `setPlaceholderText(const QString &text)`: Устанавливает текст-заполнитель, который отображается, когда поле пусто.
    - `placeholderText() const`: Возвращает текст-заполнитель.
3. **Настройка и валидация ввода**:

- `setMaxLength(int)`: Устанавливает максимальное количество символов, которые можно ввести.
- `maxLength() const`: Возвращает максимальное количество символов.
- `setValidator(const QValidator *validator)`: Устанавливает валидатор для ввода.
- `validator() const`: Возвращает текущий валидатор.
- `setEchoMode(QLineEdit.EchoMode mode)`: Устанавливает режим эха (например, скрытый ввод для паролей).
- `echoMode() const`: Возвращает текущий режим эха.

4. **Управление выделением и курсором**:
   - `setSelection(int start, int length)`: Выделяет текст в указанном диапазоне.
   - `selectAll()`: Выделяет весь текст.
   - `cursorPosition() const`: Возвращает текущую позицию курсора.
   - `setCursorPosition(int)`: Устанавливает позицию курсора.
5. **События и сигналы**:
   - `textChanged(const QString &text)`: Сигнал, который срабатывает при изменении текста.
   - `textEdited(const QString &text)`: Сигнал, который срабатывает при редактировании текста пользователем.
   - `editingFinished()`: Сигнал, который срабатывает при завершении редактирования (например, при потере фокуса или нажатии Enter).
   - `returnPressed()`: Сигнал, который срабатывает при нажатии клавиши Enter.
6. **Настройка внешнего вида**:
   - `setAlignment(Qt.Alignment alignment)`: Устанавливает выравнивание текста.
   - `alignment() const`: Возвращает текущее выравнивание.
   - `setReadOnly(bool)`: Устанавливает режим только для чтения.
   - `isReadOnly() const`: Возвращает `true`, если поле только для чтения.

```
displayText
editingFinished
```

textChanged

QtextEdit

QTextEdit - это класс в PyQt6, который предоставляет многострочный текстовый редактор, поддерживающий форматированный текст (HTML, Rich Text) и простой текст. `QTextEdit` используется для ввода и отображения текста с возможностью редактирования.

Основные методы и атрибуты

1. **Создание QTextEdit**:
   - `QTextEdit(QWidget parent=None)`: Создает текстовый редактор. Параметр `parent` указывает родительский виджет.
2. **Управление текстом**:

- `QTextEdit.setPlainText(str text)`: Устанавливает простой текст (без форматирования).
- `QTextEdit.plainText()`: Возвращает простой текст.
- `QTextEdit.setHtml(str text)`: Устанавливает текст в формате HTML.
- `QTextEdit.toHtml()`: Возвращает текст в формате HTML.
- `QTextEdit.append(str text)`: Добавляет текст в конец текущего содержимого.

3. **Форматирование текста**:
   - `QTextEdit.setFont(QFont font)`: Устанавливает шрифт для текста.
   - `QTextEdit.setFontFamily(str family)`: Устанавливает семейство шрифтов для текущего текста.
   - `QTextEdit.setFontPointSize(float size)`: Устанавливает размер шрифта для текущего текста.
   - `QTextEdit.setFontWeight(int weight)`: Устанавливает жирность шрифта для текущего текста.
   - `QTextEdit.setFontItalic(bool italic)`: Устанавливает курсивное начертание для текущего текста.
   - `QTextEdit.setFontUnderline(bool underline)`: Устанавливает подчеркивание для текущего текста.
   - `QTextEdit.setTextColor(QColor color)`: Устанавливает цвет текста.
   - `QTextEdit.setTextBackgroundColor(QColor color)`: Устанавливает цвет фона текста.
4. **Редактирование и курсор**:
   - `QTextEdit.undo()`: Отменяет последнее действие.
   - `QTextEdit.redo()`: Повторяет последнее отмененное действие.
   - `QTextEdit.cut()`: Вырезает выделенный текст.
   - `QTextEdit.copy()`: Копирует выделенный текст.
   - `QTextEdit.paste()`: Вставляет текст из буфера обмена.
   - `QTextEdit.selectAll()`: Выделяет весь текст.
   - `QTextEdit.clear()`: Очищает текстовый редактор.
   - `QTextEdit.textCursor()`: Возвращает объект `QTextCursor`, который используется для управления положением курсора и выделением текста.
5. **Сигналы**:
   - `QTextEdit.textChanged()`: Сигнал, который срабатывает при изменении текста.
   - `QTextEdit.cursorPositionChanged()`: Сигнал, который срабатывает при изменении положения курсора.

```
setPlainText
setPlainText("Hello, QTextEdit!")
append("Appended text!")
clear()
```

QSpinBox

```
spin_box.valueChanged.connect(self.on_value_changed)
def on_value_changed(self, value):
    self.label.setText(f"Value: {value}")
```

QSpinBox - это класс в PyQt6, который предоставляет виджет для ввода числовых значений с использованием кнопок увеличения и уменьшения. Этот виджет позволяет пользователю вводить числа в заданном диапазоне с шагом, определяемым разработчиком.

Основные методы и атрибуты

1. **Создание QSpinBox**:
   - `QSpinBox(QWidget parent=None)`: Создает спин-бокс. Параметр `parent` указывает родительский виджет.
2. **Управление значениями**:
   - `QSpinBox.setValue(int value)`: Устанавливает текущее значение.
   - `QSpinBox.value()`: Возвращает текущее значение.
   - `QSpinBox.setMinimum(int min)`: Устанавливает минимальное значение.
   - `QSpinBox.setMaximum(int max)`: Устанавливает максимальное значение.
   - `QSpinBox.setRange(int min, int max)`: Устанавливает минимальное и максимальное значения.
   - `QSpinBox.setSingleStep(int step)`: Устанавливает шаг изменения значения.
3. **Форматирование**:
   - `QSpinBox.setPrefix(str prefix)`: Устанавливает префикс, который будет отображаться перед значением.
   - `QSpinBox.setSuffix(str suffix)`: Устанавливает суффикс, который будет отображаться после значения.
   - `QSpinBox.setDisplayIntegerBase(int base)`: Устанавливает основание системы счисления (например, 10 для десятичной системы).
4. **Сигналы**:
   - `QSpinBox.valueChanged(int value)`: Сигнал, который срабатывает при изменении значения спин-бокса.

QListWidget

```
self.list_widget = QListWidget()
self.list_widget.addItems(["Item 1", "Item 2", "Item 3"])
self.list_widget.itemClicked.connect(self.on_item_clicked)
def on_item_clicked(self, item):
    QMessageBox.information(self, "Item Clicked", f"You clicked:
{item.text()}")
```

QListWidget - это класс в PyQt6, который предоставляет виджет для отображения и управления списком элементов. Он упрощает работу с элементами списка, позволяя добавлять, удалять и изменять элементы, а также обрабатывать различные события, связанные с ними.

Основные методы и атрибуты

1. **Создание и конфигурация QListWidget**:
   - `QListWidget(QWidget *parent = nullptr)`: Создает объект QListWidget с указанным родительским виджетом (или без родителя).

- `addItem(const QString &label)`: Добавляет элемент списка с указанным текстом.
- `addItems(const QStringList &labels)`: Добавляет несколько элементов списка.
- `insertItem(int row, const QString &label)`: Вставляет элемент в указанную строку.
- `clear()`: Удаляет все элементы из списка.

2. **Доступ к элементам списка**:
   - `item(int row) const`: Возвращает элемент на указанной строке.
   - `row(const QListWidgetItem *item) const`: Возвращает индекс указанного элемента.
   - `takeItem(int row)`: Удаляет и возвращает элемент из указанной строки.
3. **Управление элементами списка**:
   - `setItemSelected(QListWidgetItem *item, bool)`: Устанавливает или снимает выделение указанного элемента.
   - `setSelectionMode(QAbstractItemView::SelectionMode mode)`: Устанавливает режим выбора (например, один элемент, множественный выбор).
   - `selectedItems() const`: Возвращает список выделенных элементов.
4. **События и сигналы**:
   - `itemClicked(QListWidgetItem *item)`: Сигнал, который срабатывает при щелчке на элементе.
   - `itemDoubleClicked(QListWidgetItem *item)`: Сигнал, который срабатывает при двойном щелчке на элементе.
   - `currentItemChanged(QListWidgetItem *current, QListWidgetItem *previous)`: Сигнал, который срабатывает при изменении текущего элемента.
5. **Настройка внешнего вида**:
   - `setIconSize(const QSize &size)`: Устанавливает размер значков элементов.
   - `setItemWidget(QListWidgetItem *item, QWidget *widget)`: Устанавливает виджет для элемента списка.

QTableWidget

QTableWidget - это класс в PyQt6, который предоставляет виджет для отображения и редактирования таблиц. Таблицы в QTableWidget могут содержать любое количество строк и столбцов, и каждая ячейка таблицы может содержать текст, изображения, виджеты и другие типы данных.

Основные методы и атрибуты

1. **Создание QTableWidget**:
   - `QTableWidget(int rows, int columns, QWidget parent=None)`: Создает таблицу с заданным количеством строк и столбцов. Параметр `parent` указывает родительский виджет.
2. **Управление строками и столбцами**:
   - `QTableWidget.setRowCount(int rows)`: Устанавливает количество строк.
   - `QTableWidget.setColumnCount(int columns)`: Устанавливает количество столбцов.

- `QTableWidget.rowCount()`: Возвращает количество строк.
- `QTableWidget.columnCount()`: Возвращает количество столбцов.
- `QTableWidget.insertRow(int row)`: Вставляет новую строку на указанную позицию.
- `QTableWidget.insertColumn(int column)`: Вставляет новый столбец на указанную позицию.
- `QTableWidget.removeRow(int row)`: Удаляет строку на указанной позиции.
- `QTableWidget.removeColumn(int column)`: Удаляет столбец на указанной позиции.

3. **Управление ячейками**:
   - `QTableWidget.setItem(int row, int column, QTableWidgetItem item)`: Устанавливает элемент в указанную ячейку.
   - `QTableWidget.item(int row, int column)`: Возвращает элемент из указанной ячейки.
   - `QTableWidget.takeItem(int row, int column)`: Удаляет элемент из указанной ячейки и возвращает его.
   - `QTableWidget.setCellWidget(int row, int column, QWidget widget)`: Устанавливает виджет в указанную ячейку.
4. **Работа с заголовками**:
   - `QTableWidget.setHorizontalHeaderLabels([str labels])`: Устанавливает метки заголовков столбцов.
   - `QTableWidget.setVerticalHeaderLabels([str labels])`: Устанавливает метки заголовков строк.
   - `QTableWidget.horizontalHeaderItem(int column)`: Возвращает элемент заголовка для указанного столбца.
   - `QTableWidget.verticalHeaderItem(int row)`: Возвращает элемент заголовка для указанной строки.
5. **Выбор и редактирование**:
   - `QTableWidget.setSelectionMode(QAbstractItemView.SelectionMode mode)`: Устанавливает режим выбора элементов.
   - `QTableWidget.setSelectionBehavior(QAbstractItemView.SelectionBehavior behavior)`: Устанавливает поведение при выборе элементов.
   - `QTableWidget.editItem(QTableWidgetItem item)`: Позволяет редактировать указанный элемент.
6. **Сигналы**:
   - `QTableWidget.itemChanged(QTableWidgetItem item)`: Сигнал, который срабатывает при изменении элемента.
   - `QTableWidget.cellClicked(int row, int column)`: Сигнал, который срабатывает при нажатии на ячейку.

```
7. self.table = QTableWidget(3, 3)  # 3 строки и 3 столбца
   self.table.setHorizontalHeaderLabels(["Column 1", "Column 2", "Column
   3"])
   self.table.setVerticalHeaderLabels(["Row 1", "Row 2", "Row 3"])
   for row in range(3):
       for column in range(3):
           item = QTableWidgetItem(f"Item {row+1}, {column+1}")
           self.table.setItem(row, column, item)
   self.table.itemChanged.connect(self.on_item_changed)
8. def on_item_changed(self, item):
       print(f"Item changed: {item.text()} at row {item.row()}, column
   {item.column()}")
```

QComboBox

QComboBox - это класс в PyQt6, который предоставляет выпадающий список, из которого пользователь может выбрать один элемент. Также QComboBox может предоставлять возможность пользователю вводить текст.

Основные методы и атрибуты

1. **Создание и конфигурация QComboBox**:
   - `QComboBox(QWidget *parent = nullptr)`: Создает объект QComboBox с указанным родительским виджетом (или без родителя).
   - `addItem(const QString &text, const QVariant &userData = QVariant())`: Добавляет элемент в выпадающий список с опциональными пользовательскими данными.
   - `addItems(const QStringList &texts)`: Добавляет несколько элементов в выпадающий список.
   - `insertItem(int index, const QString &text, const QVariant &userData = QVariant())`: Вставляет элемент в выпадающий список по указанному индексу.
2. **Доступ к элементам списка**:
   - `itemText(int index) const`: Возвращает текст элемента по индексу.
   - `itemData(int index, int role = Qt.ItemDataRole.UserRole) const`: Возвращает пользовательские данные элемента по индексу.
   - `setItemText(int index, const QString &text)`: Устанавливает текст элемента по индексу.
   - `setItemData(int index, const QVariant &value, int role = Qt.ItemDataRole.UserRole)`: Устанавливает пользовательские данные элемента по индексу.
   - `removeItem(int index)`: Удаляет элемент по индексу.
   - `clear()`: Удаляет все элементы из выпадающего списка.
3. **Настройка и управление выбором**:
   - `currentIndex() const`: Возвращает текущий индекс выбранного элемента.
   - `setCurrentIndex(int index)`: Устанавливает текущий индекс выбранного элемента.
   - `currentText() const`: Возвращает текст текущего выбранного элемента.
   - `setEditable(bool editable)`: Устанавливает возможность редактирования текста в QComboBox.
   - `isEditable() const`: Возвращает `true`, если QComboBox редактируемый.
4. **События и сигналы**:
   - `activated(int index)`: Сигнал, который срабатывает при выборе элемента (или при вводе и подтверждении текста в редактируемом QComboBox).
   - `currentIndexChanged(int index)`: Сигнал, который срабатывает при изменении текущего индекса.
   - `currentTextChanged(const QString &text)`: Сигнал, который срабатывает при изменении текста текущего выбранного элемента.
5. **Настройка внешнего вида**:
   - `setIconSize(const QSize &size)`: Устанавливает размер значков элементов.
   - `setMaxVisibleItems(int maxItems)`: Устанавливает максимальное количество видимых элементов в выпадающем списке.

```
widget.addItems(["One", "Two", "Three", 'Four'])

# Sends the current index (position) of the selected item.
widget.currentIndexChanged.connect( self.index_changed )

# There is an alternate signal to send the text.
widget.currentTextChanged.connect( self.text_changed )
6. def index_changed(self, i): # i is an int
       print(i)
```

QTabWidget - это класс в PyQt6, который предоставляет контейнер для организации нескольких страниц (или вкладок) в одном виджете. Вкладки позволяют пользователю переключаться между различными группами виджетов, сохраняя при этом общий интерфейс.

Основные методы и атрибуты

1. **Создание QTabWidget**:
   - `QTabWidget(QWidget parent=None)`: Создает контейнер вкладок. Параметр `parent` указывает родительский виджет.
2. **Управление вкладками**:
   - `QTabWidget.addTab(QWidget page, str label)`: Добавляет новую вкладку с заданной страницей и меткой.
   - `QTabWidget.insertTab(int index, QWidget page, str label)`: Вставляет вкладку на указанную позицию.
   - `QTabWidget.removeTab(int index)`: Удаляет вкладку на указанной позиции.
   - `QTabWidget.setTabText(int index, str text)`: Устанавливает текст метки для указанной вкладки.
   - `QTabWidget.tabText(int index)`: Возвращает текст метки для указанной вкладки.
   - `QTabWidget.setCurrentIndex(int index)`: Устанавливает текущую активную вкладку по индексу.
   - `QTabWidget.currentIndex()`: Возвращает индекс текущей активной вкладки.
   - `QTabWidget.setTabsClosable(bool closable)`: Устанавливает, могут ли вкладки быть закрыты (появляется кнопка закрытия на каждой вкладке).
   - `QTabWidget.setMovable(bool movable)`: Устанавливает, могут ли вкладки быть перемещены пользователем.
3. **Работа с иконками**:
   - `QTabWidget.setTabIcon(int index, QIcon icon)`: Устанавливает иконку для указанной вкладки.
   - `QTabWidget.tabIcon(int index)`: Возвращает иконку указанной вкладки.
4. **Сигналы**:
   - `QTabWidget.currentChanged(int index)`: Сигнал, который срабатывает при изменении текущей активной вкладки.
   - `QTabWidget.tabCloseRequested(int index)`: Сигнал, который срабатывает при запросе закрытия вкладки (если вкладки могут быть закрыты).

```
self.tab1 = QWidget()
self.tab1_layout = QVBoxLayout()
```

```
self.tab1_layout.addWidget(QLabel("Content of Tab 1"))
self.tab1.setLayout(self.tab1_layout)
self.tabs.addTab(self.tab1, "Tab 1")
```

QFile - это класс, предоставляющий интерфейс для работы с файлами в PyQt6. Он поддерживает операции по открытию, чтению, записи и закрытию файлов.

Основные методы и атрибуты

1. **Открытие файла**:
   - `QFile.open(mode)`: Открывает файл в указанном режиме. `mode` задается с использованием флагов `QFile.OpenModeFlag` (например, `ReadOnly`, `WriteOnly`, `ReadWrite`).
2. **Чтение файла**:
   - `QFile.read(size)`: Читает `size` байт из файла. Если `size` не указан, читается весь файл.
   - `QFile.readAll()`: Читает весь файл и возвращает его содержимое.
3. **Запись в файл**:
   - `QFile.write(data)`: Записывает `data` в файл. `data` должно быть типа `bytes` или `bytearray`.
4. **Закрытие файла**:
   - `QFile.close()`: Закрывает файл.
5. **Проверка состояния файла**:
   - `QFile.exists()`: Проверяет, существует ли файл.
   - `QFile.remove()`: Удаляет файл.