

Изучение PyQT (Python GUI) /Создание графического интерфейса на PYTHON

Что такое PyQt?

PyQt — это библиотека, которая позволяет использовать фреймворк **Qt GUI** (GUI — это графический интерфейс пользователя) в Python. Qt написан на C++. Используя его в Python, можно создавать приложения намного быстрее, не жертвуя при этом значительной частью производительности C++.

Есть два способа создания GUI приложений при помощи PyQt:

1. Дизайн виджетов при помощи кода;
2. Использование Qt Designer.

Установка PyQt5

Библиотека PyQt5 не входит в комплект поставки Python, и прежде чем начать ее использовать, необходимо установить эту библиотеку на компьютер. В настоящее время установка библиотеки PyQt 5 выполняется исключительно просто. Для этого достаточно запустить командную строку и отдать в ней команду:

```
pip3 install PyQt5
```

Дизайн виджетов при помощи кода

В первую очередь необходимо импортировать некоторые классы и модули:

Модуль , который содержит классы, реализующие компоненты графического интерфейса: окна, надписи, кнопки, текстовые поля...

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow
import sys
```

Класс, позволяющий создать приложение

Класс, позволяющий создавать окна

Модуль из которого используется список параметров, переданных в командной строке (argv), а также функция exit(), позволяющая завершить выполнение программы.

Дизайн виджетов при помощи кода

Напишем основную функцию:

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow
import sys

def application():
    app = QApplication(sys.argv)
    window = QMainWindow()
    window.setWindowTitle("Простая программа")
    window.setGeometry(300, 250, 350, 200)
    window.show()
    sys.exit(app.exec_())

application()
```

Метод, задающий текст, который будет выводиться в заголовке окна

Метод, задающий смещение окна относительно верхнего левого угла экрана и размеры (ширину и высоту) окна в пикселях

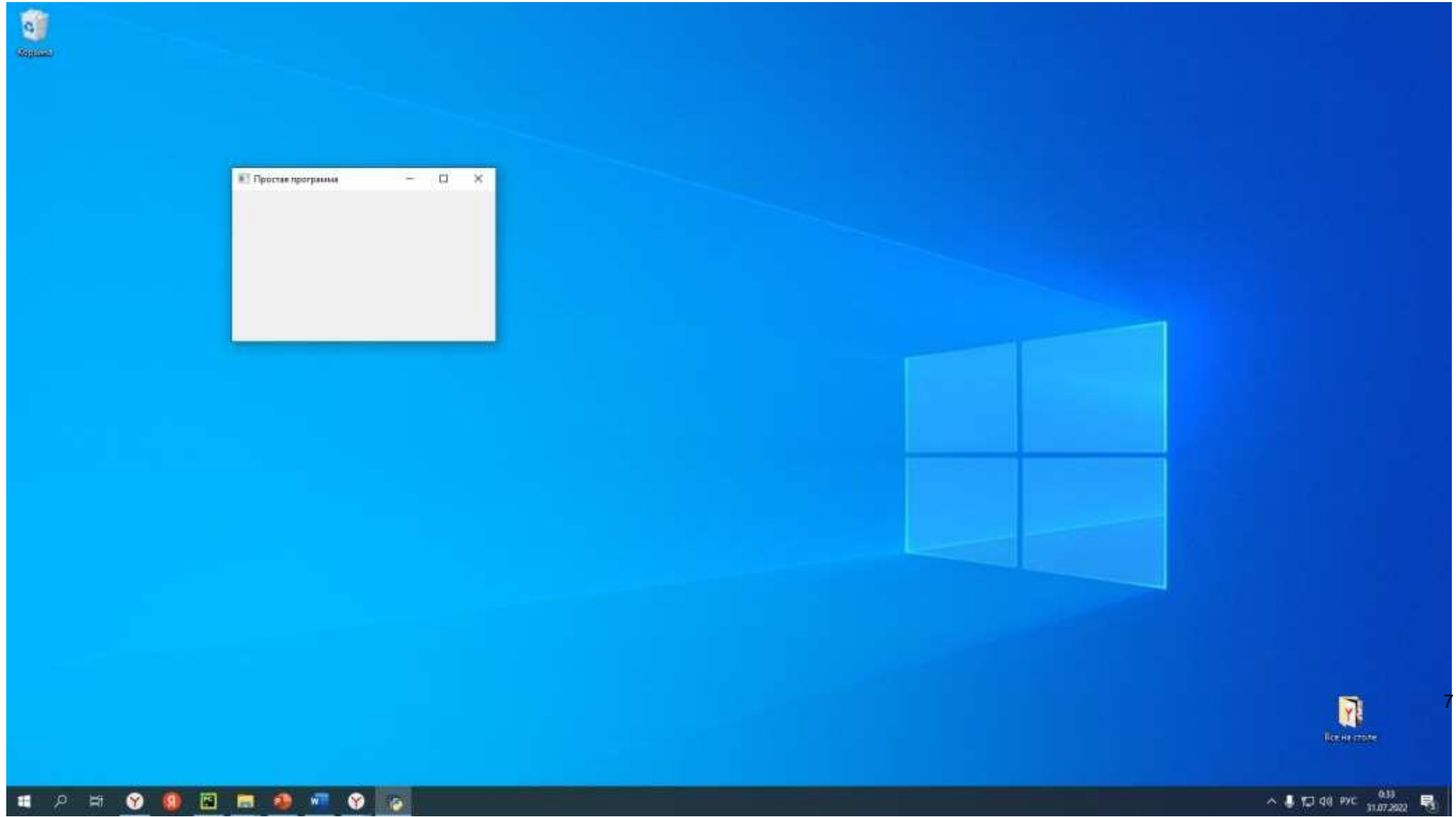
Выводит на экран окно и все его компоненты

Обеспечивает корректное завершение работы приложения

Вызов функции

Дизайн виджетов при помощи кода

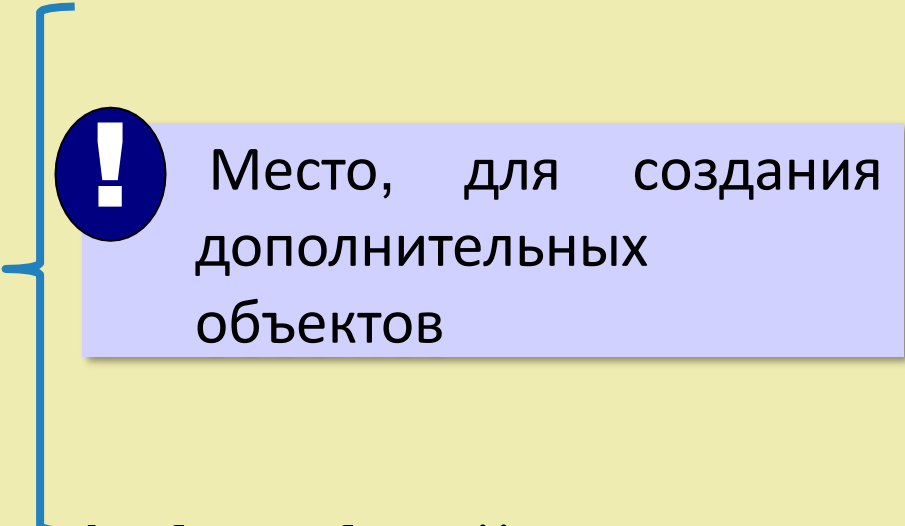
Результат:



Дизайн виджетов при помощи кода

Добавление дополнительных объектов:

```
def application():  
    app = QApplication(sys.argv)  
    window = QMainWindow()  
    window.setWindowTitle("Простая программа")  
    window.setGeometry(300, 250, 350, 200)  
  
    window.show()  
    sys.exit(app.exec_())
```



Место, для создания дополнительных объектов

Дизайн виджетов при помощи кода

Добавление дополнительных объектов:

```
main_text = QtWidgets.QLabel(window)
```

Создание объекта
надпись

```
main_text.setText("Это базовая надпись")
```

Добавление текста в
надпись

```
main_text.move(100, 100)
```

Смещение надписи относительно верхнего
левого угла окна

```
main_text.adjustSize()
```

Подбор ширины объекта под его содержимое

```
btn = QtWidgets.QPushButton(window)
```

Создание объекта
кнопка

```
btn.move(70, 150)
```

```
btn.setText("Нажми на меня")
```

Смещение кнопки относительно
верхнего левого угла окна

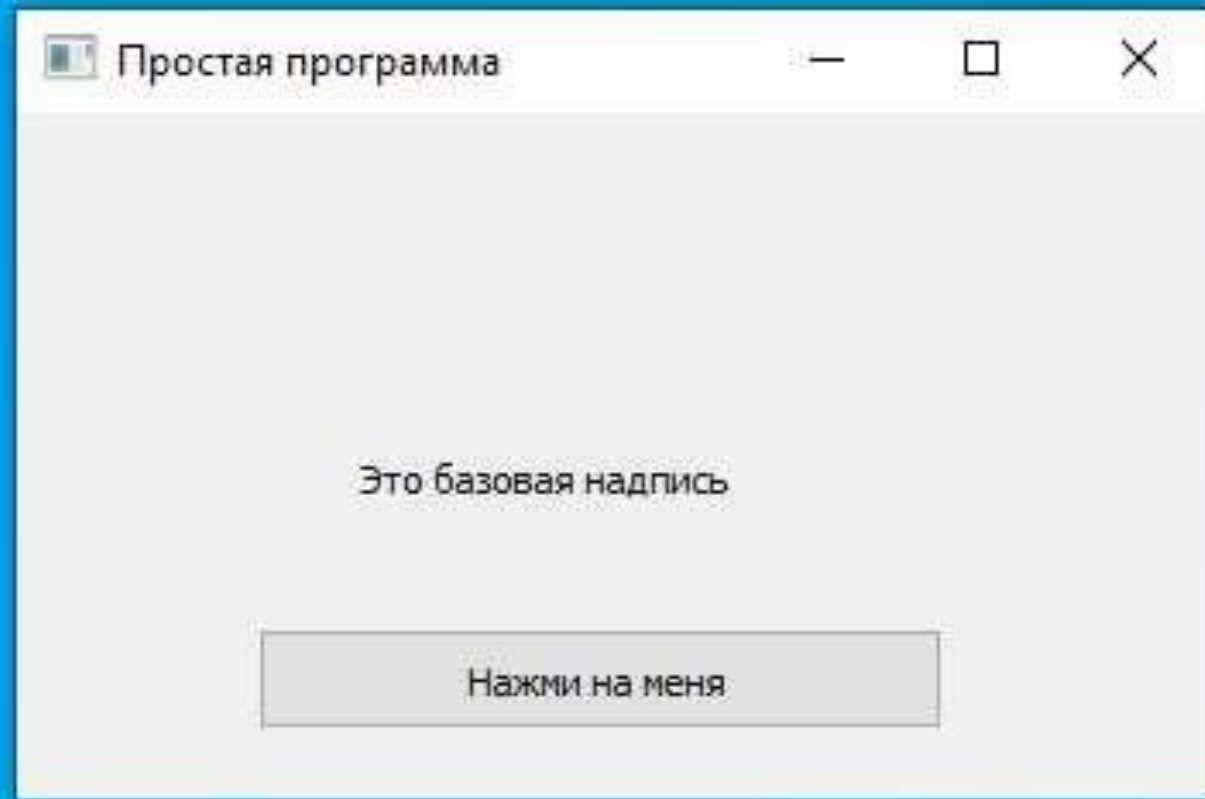
```
btn.setFixedWidth(200)
```

Указание ширины
кнопки

Добавление текста
на кнопку

Дизайн виджетов при помощи кода

Результат:



Дизайн виджетов при помощи кода

Добавление действия для кнопки:

```
def add_label():  
    print("add")
```

Добавление метода, выполняемого при нажатии на кнопку

```
def application():
```

```
    ...
```

```
    btn = QtWidgets.QPushButton(window)
```

```
    btn.move(70, 150)
```

```
    btn.setText("Нажми на меня")
```

```
    btn.setFixedWidth(200)
```

```
    btn.clicked.connect(add_label)
```

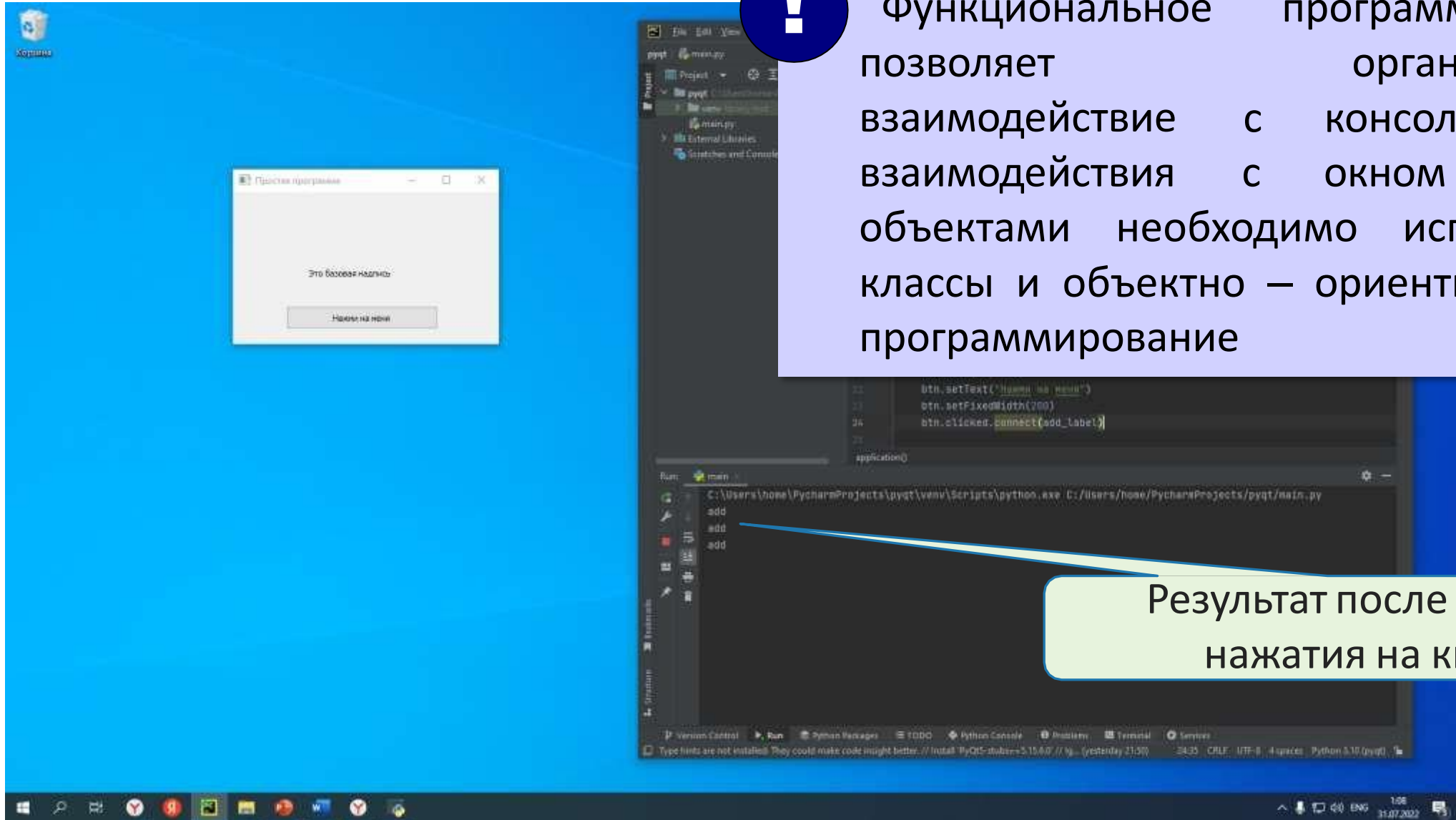
```
    window.show()
```

```
    sys.exit(app.exec_())
```

Обработка нажатия кнопки. После нажатия₁₁ на кнопку вызывается функция **add_label()**

Дизайн виджетов при помощи кода

Результат:



Функциональное программирование позволяет организовывать взаимодействие с консолью. Для взаимодействия с окном и его объектами необходимо использовать классы и объектно — ориентированное программирование

Результат после третьего нажатия на кнопку

Дизайн виджетов при помощи кода (ООП)

Создание класса:

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow
import sys
class Window (QMainWindow):
    def __init__(self):
        super(Window, self). __ini
```

Создаем класс который наследуется от QMainWindow, данный класс будет нашим основным

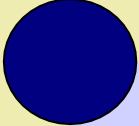
Дизайн виджетов при помощи кода (ООП)

Дизайн виджетов при помощи кода (ООП)

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow
import sys

class Window (QMainWindow):

    def __init__(self):
        super(Window, self). __init__()
        self w.setWindowTitle("Простая программа")
        self w.setGeometry(300, 250, 350, 200)
        main_text = QtWidgets.QLabel(self)
        main_text.setText("Это базовая надпись")
        main_text.move(100, 100)
        main_text.adjustSize()
```



Теперь окно это объект класса **Window**, а свойства окна — свойства объекта класса **Window**

Дизайн виджетов при помощи кода (ООП)

Дизайн виджетов при помощи кода (ООП)

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow
import sys
class Window (QMainWindow):
    def __init__(self):
        super(Window, self). __init__()
        self.setWindowTitle("Простая программа")
        self.setGeometry(300, 250, 350, 200)
        self.main_text = QtWidgets.QLabel(self)
        self.main_text.setText("Это базовая надпись")
        self.main_text.move(100, 100)
        self.main_text.adjustSize()
```



Теперь окно это объект класса **Window**, а свойства окна — свойства объекта класса **Window**

Дизайн виджетов при помощи кода (ООП)

Добавление метода `add_label()` к классу **Window**:

```
class Window (QMainWindow) :  
    def __init__(self):  
        super(Window, self).__init__()  
        self.setWindowTitle("Простая программа")  
        self.setGeometry(300, 250, 350, 200)  
        self.main_text = QtWidgets.QLabel(self)  
        self.main_text.setText("Это базовая надпись")  
        self.main_text.move(100, 100)  
        self.main_text.adjustSize()  
  
def add_label():  
    print("add")
```



Добавим отступы,
чтобы показать
принадлежность классу

Дизайн виджетов при помощи кода (ООП)

Добавление метода `add_label()` к классу `Window`:

```
class Window (QMainWindow) :  
    def __init__(self):  
        super(Window, self).__init__()  
        self.setWindowTitle("Простая программа")  
        self.setGeometry(300, 250, 350, 200)  
        self.main_text = QtWidgets.QLabel(self)  
        self.main_text.setText("Это базовая надпись")  
        self.main_text.move(100, 100)  
        self.main_text.adjustSize()  
  
    def add_label():  
        print("add")
```



Добавим отступы,
чтобы показать
принадлежность классу

Дизайн виджетов при помощи кода (ООП)

Добавление метода `add_label()` и классу `Window`:
Дизайн виджетов при помощи кода (ООП)

```
class Window (QMainWindow):  
    def __init__(self):  
        super(Window, self).__init__()  
        self.setWindowTitle("Простая программа")  
        self.setGeometry(300, 250, 350, 200)  
        self.main_text = QtWidgets.QLabel(self)  
        self.main_text.setText("Это базовая надпись")  
        self.main_text.move(100, 100)  
        self.main_text.adjustSize()  
  
    def add_label():  
        print("add")
```



Добавим параметр **self**,
он должен быть у любого
метода принадлежащего
классу

Дизайн виджетов при помощи кода (ООП)

Добавление метода `add_label()` к классу `Window`:

```
class Window (QMainWindow) :  
    def __init__(self) :  
        super(Window, self).__init__()  
        self.setWindowTitle("Простая программа")  
        self.setGeometry(300, 250, 350, 200)  
        self.main_text = QtWidgets.QLabel(self)  
        self.main_text.setText("Это базовая надпись")  
        self.main_text.move(100, 100)  
        self.main_text.adjustSize()  
    def add_label(self) :  
        print("add")
```

Дизайн виджетов при помощи кода (ООП)

Изменение метода `application()` при помощи кода (ООП)

```
def application():  
    app = QApplication(sys.argv)  
    window = Window()  
    window.show()  
    sys.exit(app.exec_())  
application()
```



Теперь окно это объект класса **Window**

Дизайн виджетов при помощи кода (ООП)

Изменение метода `application()` при помощи кода (ООП)

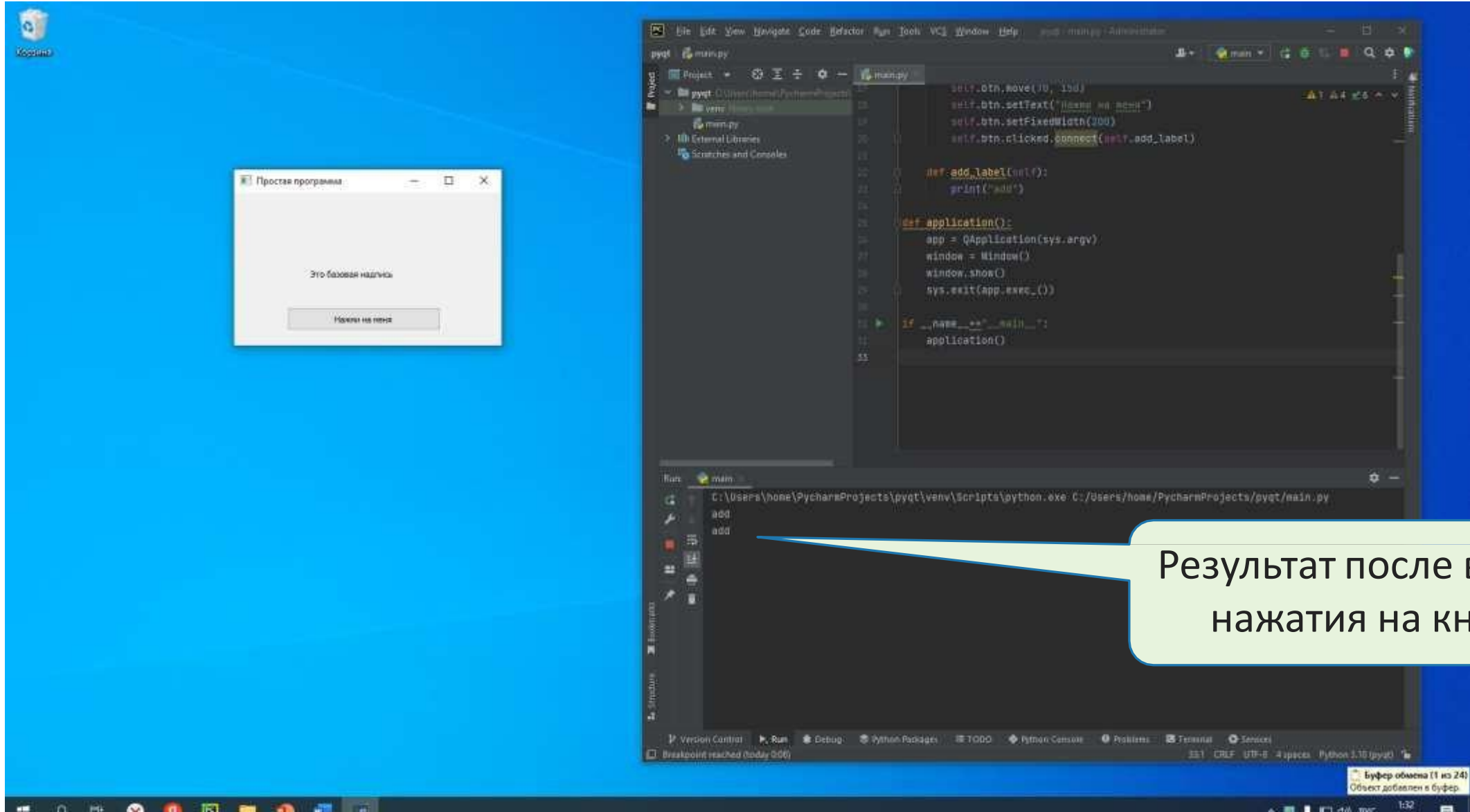
```
def application():  
    app = QApplication(sys.argv)  
    window = Window()  
    window.show()  
    sys.exit(app.exec_())  
  
if __name__ == "__main__":  
    application()
```



Вызов функции `application()` пропишем для вызова основного файла проекта

Дизайн виджетов при помощи кода (ООП)

Результат аналогичный:



Дизайн виджетов при помощи кода (ООП)

Организация взаимодействия с объектом окна - при нажатии на кнопку, в окне должна появляться новая надпись.

Шаг 1. Добавление пустого объекта надпись в конструктор класса **Window**:

```
class Window (QMainWindow):  
    def __init__(self):  
        super(Window, self).__init__()  
        self.setWindowTitle("Простая программа")  
        self.setGeometry(300, 250, 350, 200)  
        self.new_text = QtWidgets.QLabel(self)  
        self.main_text = QtWidgets.QLabel(self)  
        self.main_text.setText("Это базовая надпись")  
        self.main_text.move(100, 100)  
        self.main_text.adjustSize()
```

Дизайн виджетов при помощи кода (ООП)

Организация взаимодействия с объектом окна - при нажатии на кнопку, в окне должна появляться новая надпись.

Шаг 2. Изменение метода **add_label()** :

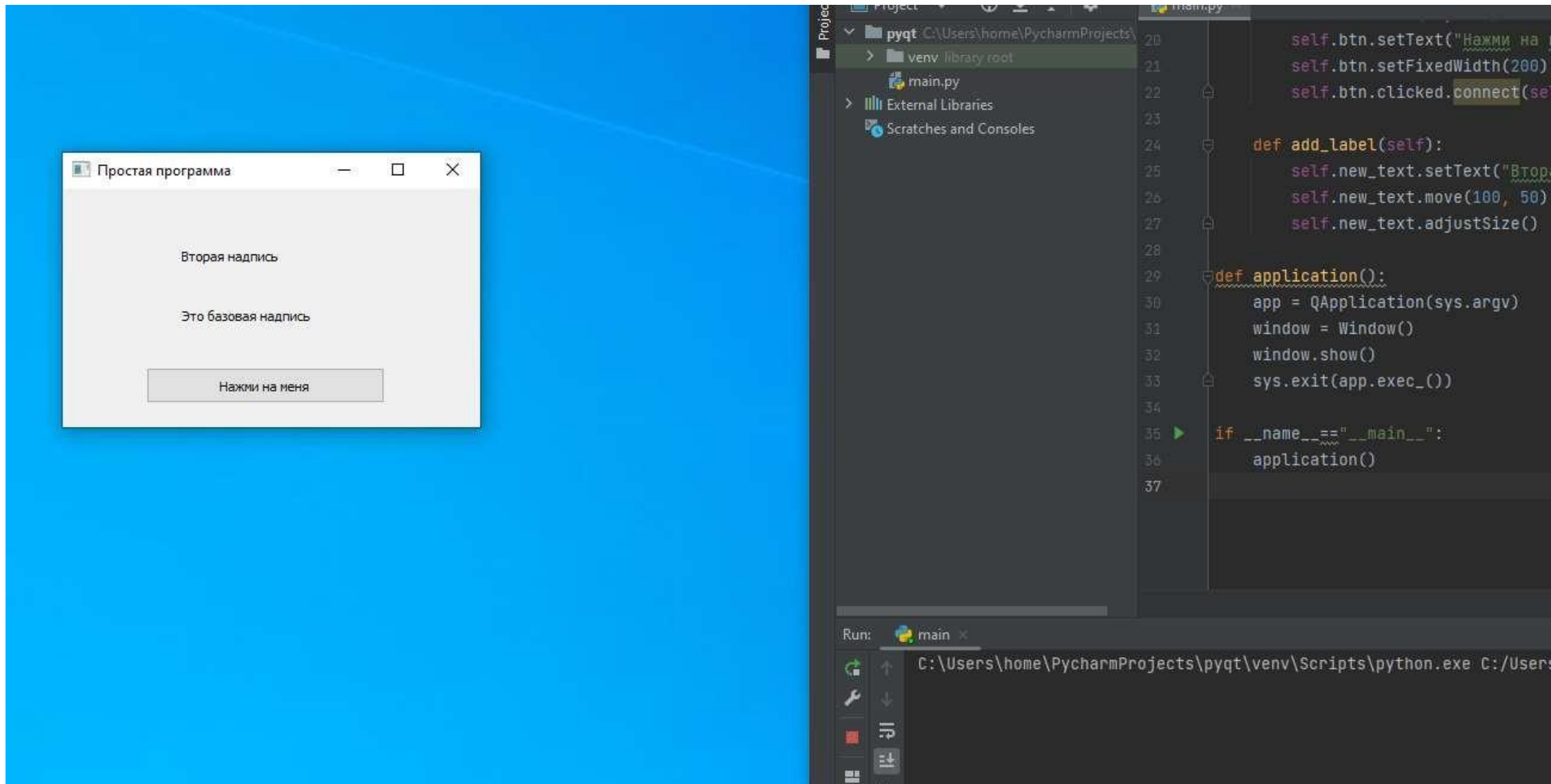
```
def add_label(self):  
    self.new_text.setText("Вторая надпись")  
    self.new_text.move(100, 50)  
    self.new_text.adjustSize()
```



Какие свойства надписи **new_text** задает метод **add_label()**

Дизайн виджетов при помощи кода (ООП)

Результат, после нажатия на кнопку:



Обработка ошибок в PyQt5

При использовании PyQt5, если в коде, который должен выполняться по какому либо действию, происходит ошибка, то нет в консоли привычного описания этой самой ошибки. Для того, что бы вывести стек ошибок нужно прописать дополнительный код, а именно: создать собственный `excepthook`.

Для этого нужно:

1. Импортировать модуль `traceback`:

```
import traceback
```



Всякий раз, когда возникает исключение, вызывается функция, назначенная `sys.excepthook`. Эта функция, называемая перехватчиком исключения, применяется для вывода любых полезных сведений о происшествии в стандартный поток вывода.

Обработка ошибок в PyQt5

2. Создать функцию `excepthook`, в которой разместить такой код:

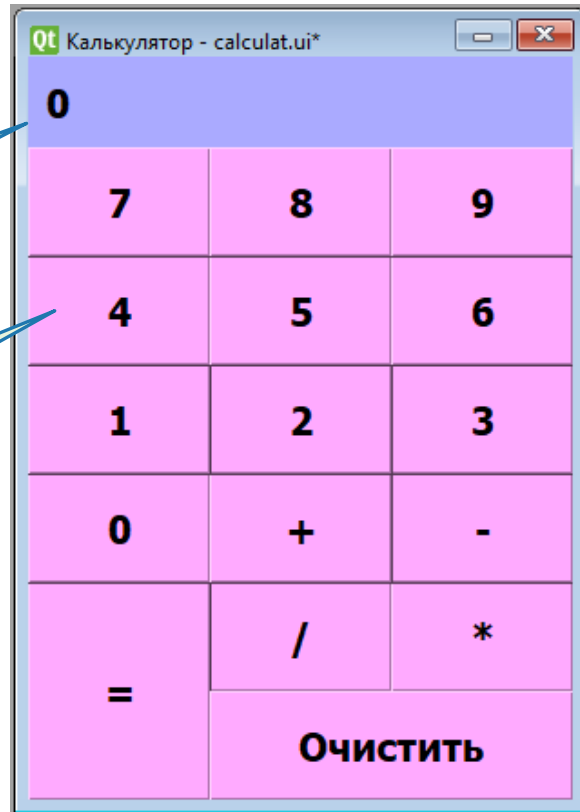
```
def excepthook(exc_type, exc_value, exc_tb):  
    tb = "".join(traceback.format_exception(exc_type,  
                                             exc_value, exc_tb))  
    print("Обнаружена ошибка !:", tb)
```

3. В вызове основного файла проекта подключить `excepthook`:

```
sys.excepthook = excepthook
```

Пример разработки приложения «Калькулятор»

- 1) Выбираем шаблон Main Window
- 2) Дизайн. Размещаем объекты и задаем им нужные свойства



Label

PushButton



В чем будет заключаться сложность при реализации данного проекта?



Много кнопок - много функций!



Решение — использовать lambda функции!

- 3) Сохраняем результат в файл формата .ui
- 4) В режиме командной строки выполняем преобразование файла в формат .py

Разработка приложения «Калькулятор»

5) Импортируем классы и модули, пишем вызов основного файла проекта

6) Добавляем функционал кнопкам 0,1,...,9,+,-,*,/ :

```
self.Button_0.clicked.connect(lambda: self.write_number(self.Button_0.text()))
```

Имя кнопки

Функция, вызываемая
при нажатии кнопки

Аргумент
функции



Текст кнопок 0,1,...,9,+,-,*,/ будет передаваться в качестве параметра в функцию `write_number`



Аналогичный код пропишем для кнопок 0,1,...,9,+,-,*,/

Разработка приложения «Калькулятор»

7) Пишем функцию `write_number` :

```
def write_number (self, number):  
    if self.label.text()=="0":  
        self.label.setText(number)  
    else:  
        self.label.setText(self.label.text()+ number)
```



Какие еще действия необходимо выполнить?

Разработка приложения «Калькулятор»

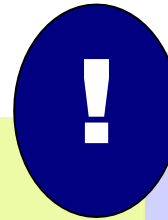
8) Добавление функционала кнопкам «=» и Очистить:

```
self.Button_rav.clicked.connect(self.result)
self.Button_C.clicked.connect(self.clear)
```

9) Пишем функции `result()` и `clear()`:

```
def result(self):
    res = eval(self.label.text())
    self.label.setText(str(res))

def clear(self):
    self.label.setText('0')
```



Функция `eval()` выполняет строку-выражение, переданную ей в качестве обязательного аргумента и возвращает результат выполнения этой строки.

Виджет QLineEdit

QLineEdit используется как один из виджетов ввода данных пользователем приложения. Он поддерживает такие функции редактирования, как выравнивание текста, вырезание, копирование, вставка, повтор или отмена.

Виджет QLineEdit имеет ряд методов для добавления дополнительных функциональных возможностей графическому интерфейсу. Это может быть скрывание текста при его вводе, использование текста - заполнителя или даже установка ограничения на длину вводимого текста.

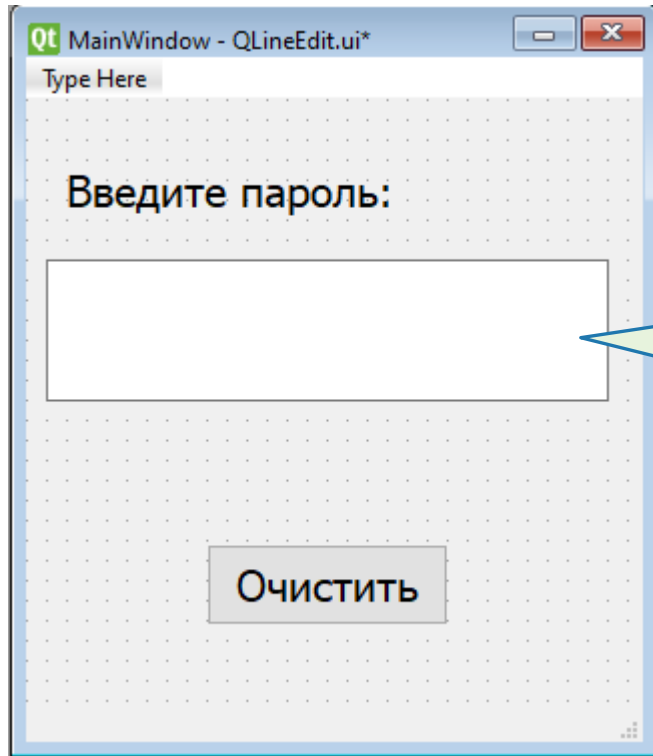
Методы виджета QLineEdit

Название	Описание	
setAlignment()	Выравнивает текст	
clear()	Удаляет содержимое	
EchoMode()	Управляет отображением текста внутри поля.	
	Normal	Отображать символы по мере их ввода. Это значение по умолчанию.
	NoEcho	Ничего не отображать. Это может быть уместно для паролей, где даже длина пароля должна храниться в секрете.
	Password	Отображение символов маски пароля, зависящих от платформы, вместо фактически введенных символов.
	PasswordEchoOnEdit	Редактирование пароля

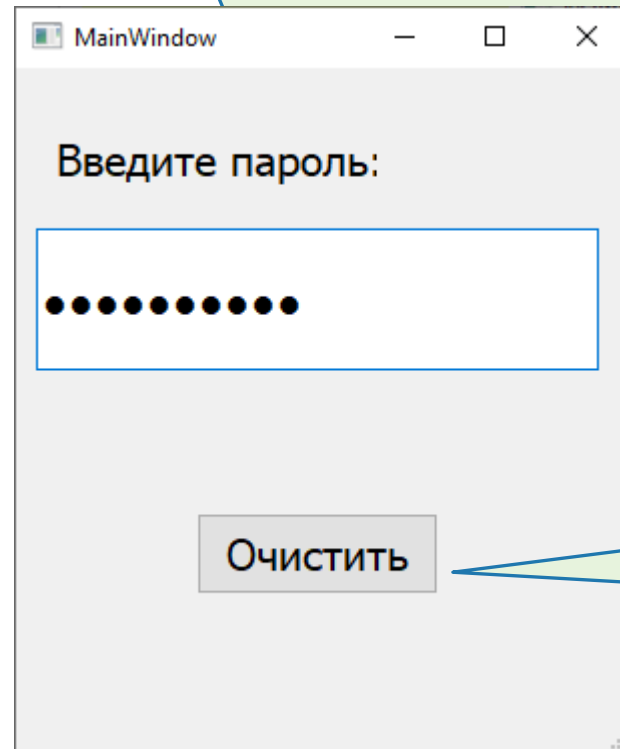
Методы виджета QLineEdit

Название	Описание
setMaxLength()	Задаёт максимальное количество символов для ввода
setReadOnly()	Делает текстовое поле недоступным для редактирования
setText()	Задаёт текст
text()	Извлекает текст
setValidator()	Задаёт правила проверки. Доступные средства проверки QIntValidator – Ограничивает ввод целым числом QDoubleValidator – Дробная часть числа, ограниченная указанными десятичными знаками QRegExpValidator – Проверяет входные данные на соответствие выражению
setInputMask()	Применяет маску комбинации символов для ввода

Пример использования QLineEdit



QLineEdit
Свойства:
EchoMode() –
Password
setMaxLength() - 10



При нажатии на кнопку
текстовое поле
очищается

Виджет QCheckBox

Виджет QCheckBox позволяет переключаться между двумя состояниями: активен или неактивен. Он идеально подходит для представления функций в графическом интерфейсе, которые можно включить или отключить, или для выбора из списка параметров.

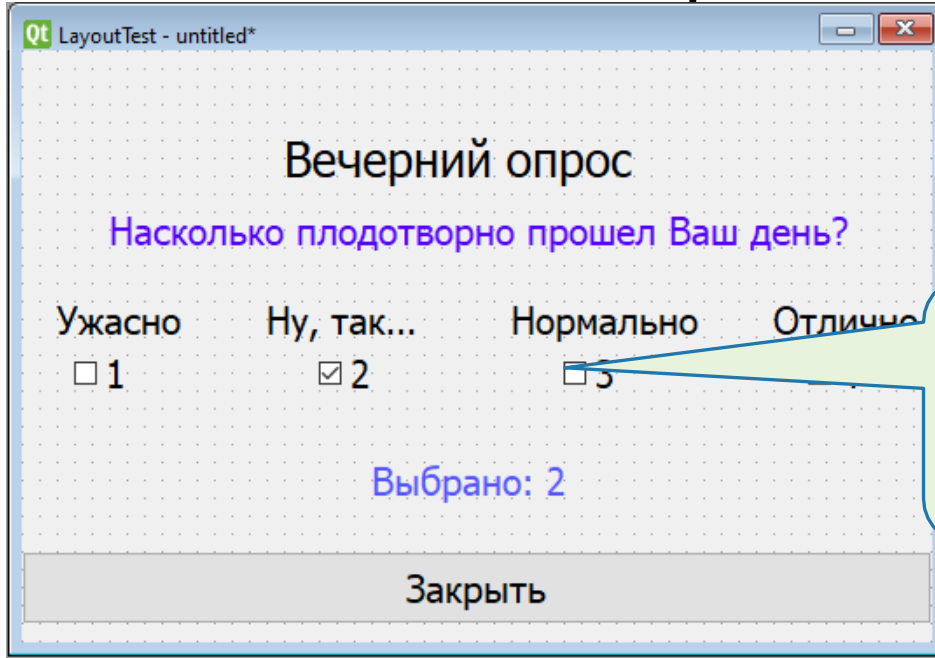
QCheckBox может использоваться для работы с динамическим GUI приложением, например, чтобы изменить заголовок окна и текст меток, когда он включен.

Каждый раз, когда флажок установлен или снят, объект выдает сигнал **StateChanged()** .

Методы виджета QCheckBox

Название	Описание
setChecked()	Изменяет состояние флажка
setText()	Задаёт текст
text()	Извлекает текст
isChecked()	Проверяет, выбран ли флажок

Пример применения QCheckBox



QCheckBox
При нажатии
отмечается

```
self.checkBox.stateChanged.connect(lambda: self.state(self))
```

Имя флажка

объект, если флажок
установлен или снят

Вызываемая функция

Пример применения QCheckBox

Пишем функцию `state` для флажка1 и аналогичные для других флажков:

```
def state(self, b):  
    if self.checkBox.isChecked() == True:  
        self.label_7.setText ("Выбрано:1")  
    else:  
        self.label_7.setText ("Отменено")
```

Добавляем функционал кнопке Заккрыть:

```
self.pushButton.clicked.connect (QtWidgets.qApp.quit)
```

Виджет QSpinBox

Объект QSpinBox предоставляет пользователю текстовое поле, в котором отображается целое число с кнопкой вверх/вниз справа. Значение в текстовом поле увеличивается/уменьшается при нажатии кнопки вверх/вниз.

По умолчанию целое число в поле начинается с 0, доходит до 99 и изменяется на шаг равный 1.

Объект QSpinBox выдает сигнал `valueChanged()` каждый раз, когда нажимается кнопка вверх/вниз. Связанная функция слота может получить текущее значение виджета с помощью метода `value()`.

Для значений с плавающей запятой, используется `QDoubleSpinBox`.

Методы класса QSpinBox

Название	Описание
setMinimum()	Устанавливает нижнюю границу счетчика
setMaximum()	Устанавливает верхнюю границу счетчика
singleStep()	Устанавливает значение шага счетчика
value()	Возвращает текущее значение

Виджет **QRadioButton**

Объект класса **QRadioButton** представляет выбираемую кнопку с текстовой меткой.

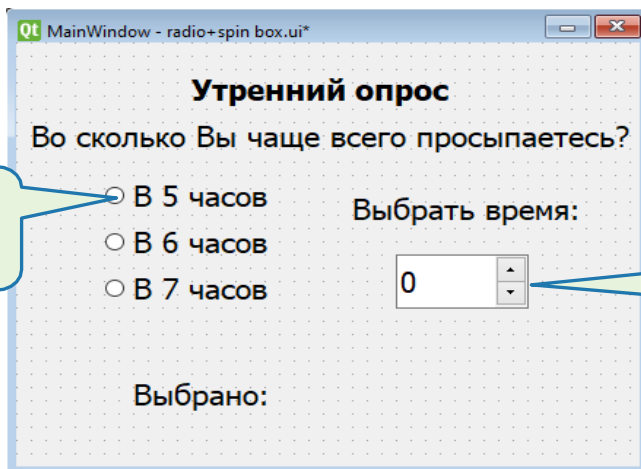
Радиокнопки по умолчанию являются автоэксклюзивными. Следовательно, только один из переключателей в родительском окне может быть выбран одновременно. Если один выбран, ранее выбранная кнопка автоматически отменяется.

Сигналом по умолчанию, связанным с объектом **QRadioButton**, является `toggled()`

Методы класса QPushButton

Название	Описание
setChecked()	Изменяет состояние переключателя
setText()	Задаёт метку, связанную с кнопкой
text()	Извлекает заголовок кнопки
isChecked()	Проверяет, выбрана ли кнопка

Пример использования QSpinBox и QRadioButton



QRadioButton

QSpinBox

Программируем функционал RadioButton и SpinBox:

Имя
переключателя

Сигнал, который выдает объект, если
переключатель установлен или снят

```
self.rB1.toggled.connect(self.btnstate)
```

```
self.spinBox.valueChanged.connect(self.btnstate)
```

Сигнал, который выдает QSpinBox
каждый раз при нажатии кнопки

Вызываемая функция

Пример использования QSpinBox и QRadioButton

6) Пишем функцию btnstate:

```
def btnstate(self):  
    if self.rB1.isChecked() == True:  
        self.label_3.setText("Выбрано: "+ self.rB1.text())  
    elif self.rB2.isChecked() == True:  
        self.label_3.setText("Выбрано: "+ self.rB2.text())  
    elif self.rB3.isChecked() == True:  
        self.label_3.setText("Выбрано: "+ self.rB3.text())  
    elif self.rB4.isChecked() == True:  
        self.label_3.setText("Выбрано: в " +  
                               str(self.spinBox.value()) +" часа(ов)")
```

Виджет QComboBox

Объект QComboBox представляет выпадающий список элементов для выбора.

Сигналы QComboBox

Название	Описание
activated()	Когда пользователь выбирает элемент
currentIndexChanged()	Всякий раз, когда текущий индекс изменяется либо пользователем, либо программно
highlighted()	Когда элемент в списке выделяется

Методы класса QComboBox

Название	Описание
addItem()	Добавляет строку в коллекцию
addItems()	Добавляет элементы в объект списка
Clear()	Удаляет все элементы в коллекции
count()	Извлекает количество элементов в коллекции
currentText()	Извлекает текст выбранного в данный момент элемента
itemText()	Отображает текст, принадлежащий определенному индексу
currentIndex()	Возвращает индекс выбранного элемента
setItemText()	Изменяет текст указанного индекса

Пример использования QCheckBox, QComboBox и QLineEdit

The screenshot shows a Qt application window titled "Favorite color". The window contains the following elements:

- A label "Ваш любимый цвет?" (Your favorite color?) in purple text.
- A **QComboBox** widget displaying "Красный" (Red) with a dropdown arrow.
- A **QCheckBox** widget labeled "Свой вариант" (Your own option) which is checked.
- A **QLineEdit** widget containing the text "Фиолетовый" (Purple).
- A button labeled "Ответить" (Answer).
- A label "Ответ: Фиолетовый" (Answer: Purple) at the bottom.

Callouts identify the widgets:

- QCheckBox**: Points to the "Свой вариант" checkbox.
- QComboBox**: Points to the "Красный" dropdown menu.
- QLineEdit**: Points to the text input field containing "Фиолетовый".

Пример использования QCheckBox, QComboBox и QLineEdit

Добавим элементы в список:

```
self.comboBox.addItem(["Красный", "Желтый", "Зеленый"])
```

Программируем функционал флажка и кнопки:

Имя объекта
QComboBox

Сигнал, который выдает объект

```
self.comboBox.currentIndexChanged.connect(self.selectionchange)  
self.pushButton.clicked.connect(self.selectionchange)
```

Вызываемая функция

Пример использования QCheckBox, QComboBox и QLineEdit

5) Пишем функцию `selectionchange` :

```
def selectionchange(self):  
    if self.checkBox.isChecked() == True:  
        self.label_2.setText("Ответ: " + self.lineEdit.text())  
    else:  
        self.label_2.setText("Ответ: " + self.comboBox.currentText())
```





PyQt QMessageBox

PyQt `QMessageBox` - это окно для отображения некоторого информационного сообщения.

При необходимости, пользователь может ответить, нажав на любую из стандартных кнопок на нем.

Каждая стандартная кнопка имеет predetermined подписью, роль.

Методы класса QMessageBox

Название	Описание
setIcon()	Отображает предварительно определенный значок, соответствующий серьезности сообщения  Question  Information  Warning  Critical
setText()	Задаёт текст основного сообщения, который будет отображаться в диалоговом окне
setInformativeText()	Отображает дополнительную информацию
setDetailedText()	В диалоговом окне отображается кнопка Сведения. Этот текст появляется при нажатии на нее

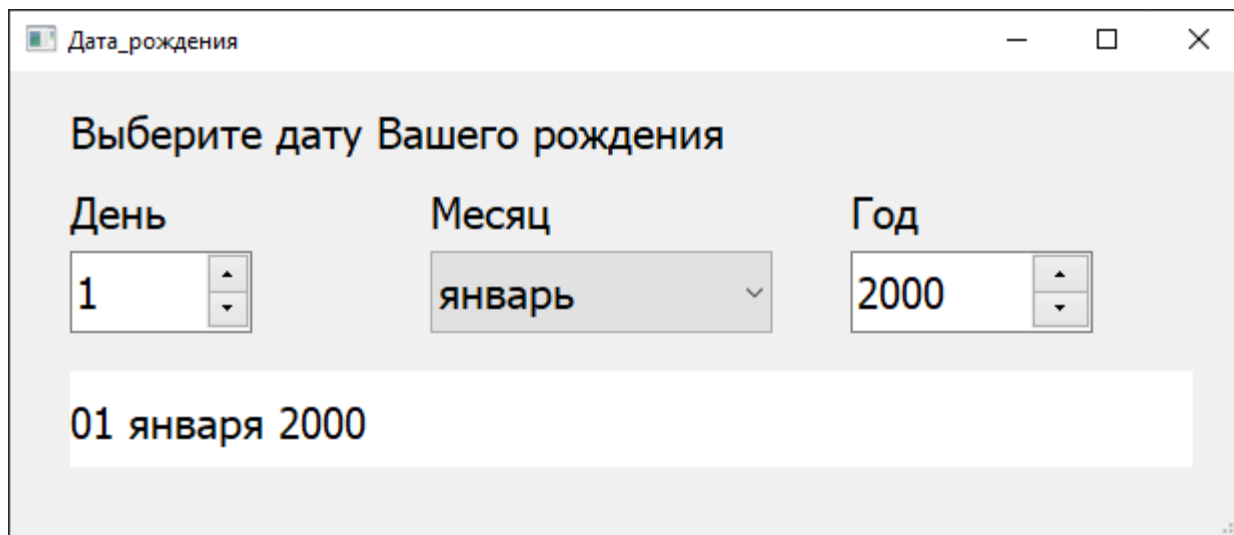
Методы класса QMessageBox

Название	Описание
setStandardButtons()	Установить стандартные кнопки() Список стандартных кнопок, которые будут отображаться. QMessageBox.Ok QMessageBox.Open QMessageBox.Save QMessageBox.Cancel QMessageBox.Close QMessageBox.Yes QMessageBox.No QMessageBox.Abort QMessageBox.Retry QMessageBox.Ignore
setTitle()	Отображает пользовательский заголовок диалогового окна

Методы класса QMessageBox

Название	Описание
setStandardButtons()	Установить стандартные кнопки() Список стандартных кнопок, которые будут отображаться. QMessageBox.Ok QMessageBox.Open QMessageBox.Save QMessageBox.Cancel QMessageBox.Close QMessageBox.Yes QMessageBox.No QMessageBox.Abort QMessageBox.Retry QMessageBox.Ignore
setTitle()	Отображает пользовательский заголовок диалогового окна

Добавление диалогового окна



Задача: При выборе пользователем значения виджета SpinBox - 28 и более, выводить окно с вопросом: «Надеюсь Вы родились не в феврале?»

Импорт класса:

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

Добавление диалогового окна

2) Изменим функцию date_day:

```
def data_day (self):  
    data = self.label_5.text().split()  
    if self.spinBox.value() <= 9:  
        stroka = '0'+str(self.spinBox.value())+' '+data[1]+' '+data[2]  
    else:  
        stroka = str(self.spinBox.value())+' '+data[1]+' '+data[2]  
    self.label_5.setText(stroka)  
    if self.spinBox.value() > 28:  
        msg = QMessageBox()  
        msg.setWindowTitle("Message Box")  
        msg.setText("Надеюсь, Вы родились не в феврале?")  
        msg.setIcon(QMessageBox.Question)  
        msg.setStandardButtons(QMessageBox.Cancel | QMessageBox.Ok)  
        msg.buttonClicked.connect(self.popup)  
        msg.exec_()
```

Создаем объект класса
QMessageBox

Показать окно

Связываем нажатие на кнопки
с функцией popup()

Добавление диалогового окна

3) Пишем функцию popup():

Функция принимает кроме обязательного параметра - второй параметр, кнопку на которую нажали

```
def popup(self, btn):  
    if btn.text() == 'OK':  
        print('Проверка прошла успешно')  
    if btn.text() == 'Cancel':  
        self.spinBox.setProperty("value", 1)
```

Если нажата кнопка с надписью 'OK'

Если нажата кнопка с надписью 'Cancel'

Устанавливаем для SpinBox активное значение - 1