Практическое задание: Построение графического интерфейса в Python

Для построения графического интерфейса приложения на базе языка программирования Python 3 существуют различные библиотеки (PyQt5, Trinket, PySide, Kyvi и другие). В текущем задании мы будем использовать библиотеку PyQt5. В PyQt5 дизайн визуальных элементов интерфейса может производиться с помощью ввода команд в коде, либо с помощью специальной программы с графическим интерфейсом PyQt5 Designer.

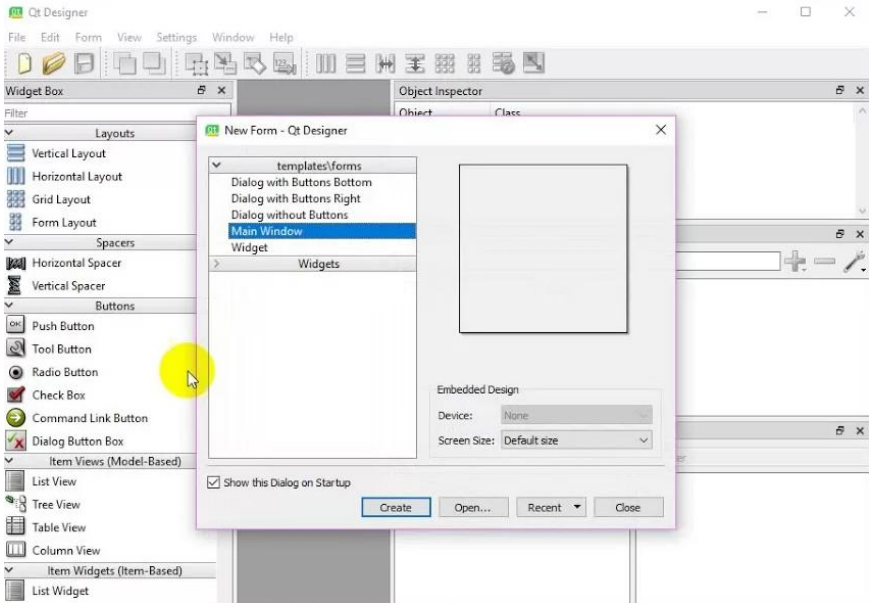

Рис. 1. PyQt5 Designer

Результатом работы PyQt5 Designer также является программный код. В текущем задании PyQt5 Designer не используется.

Цель текущего задания – построить интерфейс программы, строящей график заданной пользователем функции. Для этого необходимо в ваш репозиторий добавить файл практического задания. В виртуальном окружении нужно установить пакеты: PyQt5, Matplotlib, NumPy.

Шаг №1

Далее необходимо импортировать в файл задания библиотеку PyQt5. В данном случае интерфейс будет собираться из так называемых виджетов – элементарных единиц интерфейса (надписи, кнопки, текстовые поля и т.п.). Поэтому импорт будет выглядеть следующим образом:

```
from PyQt5.QtWidgets import (
    QApplication,
    QLabel,
    QLineEdit,
    QMainWindow,
    QPushButton,
    QVBoxLayout,
    QWidget,
)
```

Дополнительно нужно импортировать Numpy для создания векторов:

```
import numpy as np
```

Для отрисовки графика необходимо импортировать библиотеку Matplotlib:

```
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
```

Вторая строчка импорта необходима для встраивания элементов Matplotlib в интерфейс PyQt5. Далее создадим пустое окно с заголовком.

```
class MainWindow(QMainWindow):
    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)

        # Назначим заголовок окна
        self.setWindowTitle("График")

# Создать приложение QApplication
app = QApplication([])

# Создать окно приложения
main_window = MainWindow()
main_window.show()

# Запустить приложение
app.exec_()
```

Данный код является основой для дальнейшего добавления функций приложения. Сейчас приложение выглядит следующим образом:

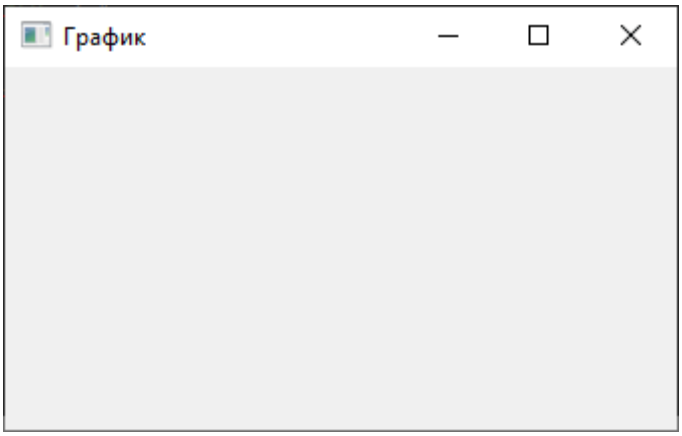

Рис. 2. Пустое окно

Добавление функций с помощью виджетов включает два шага: создание виджета и добавление его на макет с помощью метода addWidget(). Преобразуем код к виду:

```
class MainWindow(QMainWindow):
    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)

        # Назначим заголовок окна
        self.setWindowTitle("График")

        # Создание виджетов
        self.canvas = FigureCanvas(plt.figure())  # Создание полотна
```

```
Matplotlib

        # Создание центрального виджета
        central_widget = QWidget()
        layout = QVBoxLayout()  # макет, на который будут добавляться виджеты
        central_widget.setLayout(layout)  # добавление макета на центральный
виджет

        # Добавление виджетов на макет
        layout.addWidget(self.canvas)

        # Установка центрального виджета
        self.setCentralWidget(central_widget)

        # Вызов функции рисования графика
        self.plot_data()
#
```

График рисуется в функции self.plot_data():

```
def plot_data(self):
    x = np.linspace(0, 1, 50)

    function = lambda x: x**3

    y = [function(value) for value in x]

    plt.plot(x, y)
    plt.grid(True)

    plt.xlabel('x')
    plt.ylabel('y')

    plt.title('Заголовок график y=x**3')

    # Обновление виджета №0 (виджет с полотном Matplotlib)
    self.centralWidget().layout().itemAt(0).widget().draw()
```

Результат выполнения программы представлен на рис. 3.

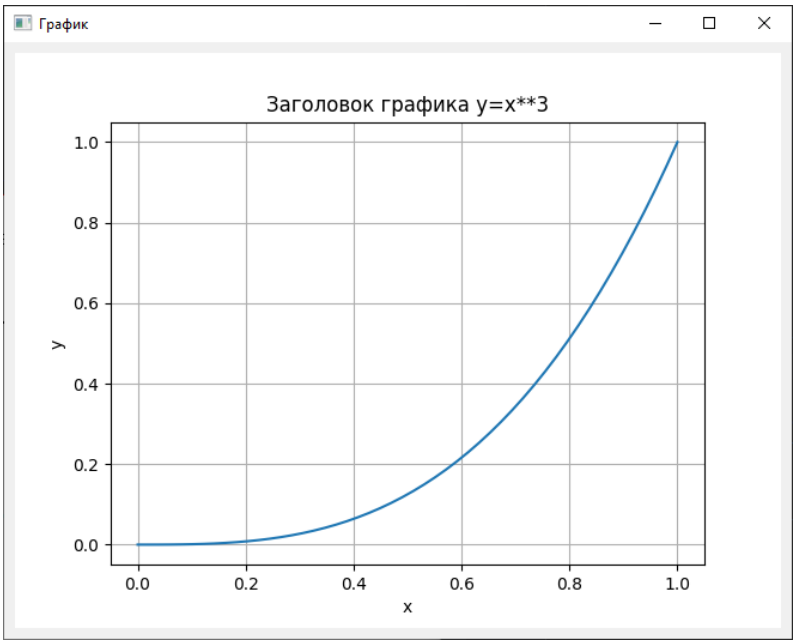

Рис. 3. Окно с графиком

Шаг №2

Добавим в раздел создания виджетов кнопку, при нажатии на которую будет рисоваться график:

```
    self.plot_button = QPushButton("Нарисовать график")
    self.plot_button.clicked.connect(self.plot_data)
```

во второй строчке описывается событие, которое произойдёт при нажатии кнопки. В данном случае будет вызвана функция plot_data. Чтобы функция вызывалась только по нажатию кнопки, необходимо удалить (или закомментировать) предыдущий код вызова функции:

```
    # Вызов функции рисования графика
    # self.plot_data()
```

Кнопка создана, но не добавлена на макет. Поэтому в раздел добавления виджетов на макет нужно включить следующий код:

```
    layout.addWidget(self.plot_button)
```

Результат выполнения этого шага представлен на рис. 4.

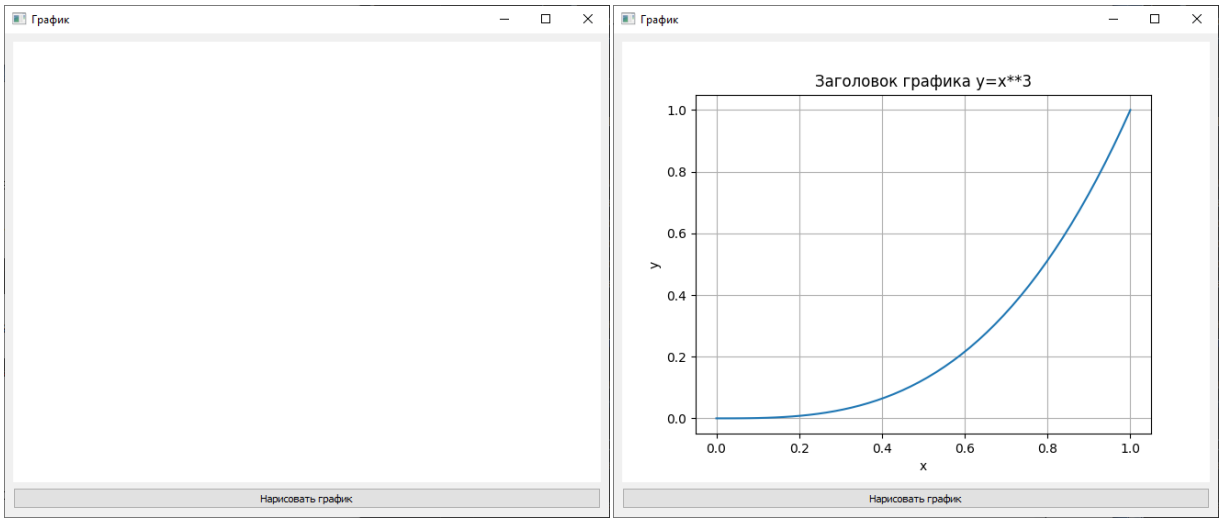

Рис. 4. Окно с графиком и кнопкой

Шаг №3

На данный момент график рисуется в заданном диапазоне от 0 до 1. Для того, чтобы дать пользователю возможность изменять этот диапазон создадим виджеты: надпись, текстовые поля:

```
    self.range_label = QLabel("Диапазон:")
    self.range_start_input = QLineEdit('0')
    self.range_end_input = QLineEdit('1')
```

Здесь ‘0’ и ‘1’ – заданные по-умолчанию значения текстовых полей. Добавим эти виджеты на макет:

```
    layout.addWidget(self.range_label)
    layout.addWidget(self.range_start_input)
    layout.addWidget(self.range_end_input)
```

Приложение можно запустить, но функционально оно не отличается от шага №2. Сейчас данные из текстовых полей не используется. Для получения этих данных добавим в начало функции plot_data следующий код:

```
try:
    range_start = float(self.range_start_input.text())
    range_end = float(self.range_end_input.text())
except ValueError:
    range_start = 0
    range_end = 1
```

Также необходимо изменить код создания вектора x:

```
x = np.linspace(range_start, range_end, 50)
```

Результат выполнения этого шага представлен на рис. 5.

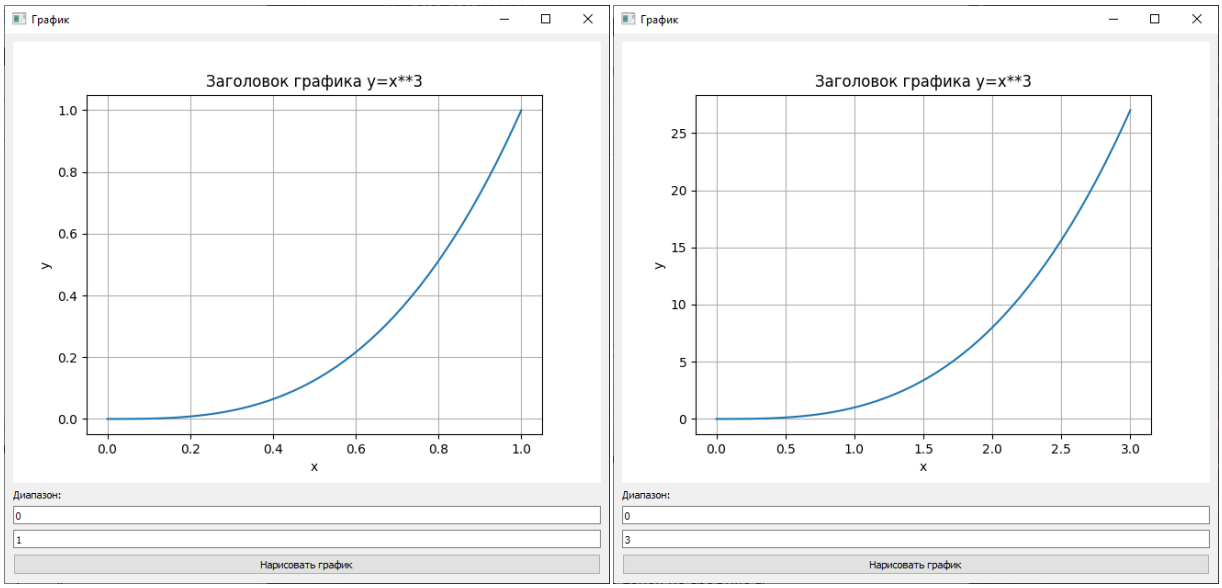

Рис. 5. Окно с графиком, кнопкой и полями ввода диапазона

Шаг №4

Добавим возможность рисования произвольной функции, заданной пользователем. Создадим виджеты надписи и текстового поля:

```
self. function_label = QLabel("Функция:")
self.function_input = QLineEdit('x**3')
```

Добавим виджеты на макет:

```
layout.addWidget(self.function_label)
layout.addWidget(self.function_input)
```

Для получения данных (выражения функции) из этого текстового поля добавим код в plot_data:

```
try:
    expression = self.function_input.text()
except ValueError:
    expression = "x**3"
```

Преобразуем строку в функцию f(x) с помощью кода:

```
functions = {}  # определим словарь функций
exec(f"def f(x): return {expression}", functions)
function = functions ["f"]
```

Здесь функция exec() определяет функцию f(x) в виде выражения возвращения кода expression (заданного пользователем) и записывает эту функцию в словарь functions. Откуда мы получаем ссылку на f(x) и записываем её в function.

Изменим заголовок графика:

```
plt.title('Заголовок графика ' + expression)
```

Результат выполнения этого шага представлен на рис. 6.

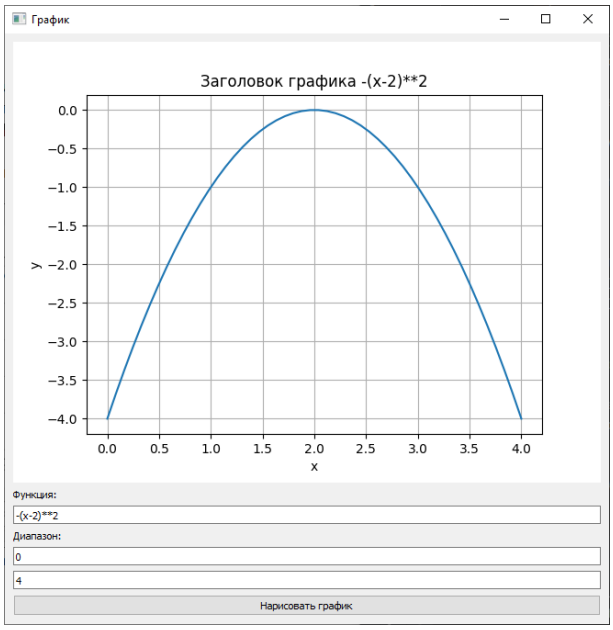

Рис. 6. Окно с графиком, кнопкой и полями ввода диапазона и выражения функции

Задания для самостоятельной работы:

1. Добавить возможность выбора количества точек на графике.
2. Добавить обработку ошибки ввода неверной функции.
3. Добавить кнопку очистки графика.
4. Добавить возможность сохранения точек графика в файл.
5. Добавить возможность добавления функций в виджет списка и выбора из него функций к отрисовке.

Ссылки:

Ссылка на документацию PyQt5
https://www.riverbankcomputing.com/static/Docs/PyQt5/

Ссылка на описание виджетов:
https://www.pythonguis.com/tutorials/pyqt-basic-widgets/
https://fadeevlecturer.github.io/python_lectures/notebooks/qt/widgets.html