* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/165/groups/1257)

[Урок QT Диалоги](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1228)

**PyQT. Диалоги, работа с изображениями**

1. [QPixmap](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1228/materials/2209#1)
2. [Диалоговые окна](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1228/materials/2209#2)
3. [Рисование](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1228/materials/2209#3)
4. [L-системы](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1228/materials/2209#4)

**Аннотация**

*До сих пор мы работали только с текстовой информацией, однако в файлах может храниться не только текст, но и картинки, видео или аудиозаписи. В этом уроке мы познакомимся с возможностями PyQT, позволяющими работать с графикой.*

**QPixmap**

Одним из способов отображения изображения является использование **QPixmap**. Напишем простейшую программу, которая будет демонстрировать заданную картинку.

from PyQt5.QtWidgets import QApplication, QWidget

from PyQt5.QtWidgets import QPushButton, QLineEdit,QLabel

from PyQt5.QtGui import QPixmap

import sys

SCREEN\_SIZE = [400, 400]

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(400, 400, \*SCREEN\_SIZE)

self.setWindowTitle('Отображение картинки')

## Изображение

self.pixmap = QPixmap('orig.png')

self.image = QLabel(self)

self.image.move(80, 60)

self.image.resize(250, 250)

self.image.setPixmap(self.pixmap)

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

app = QApplication(sys.argv)

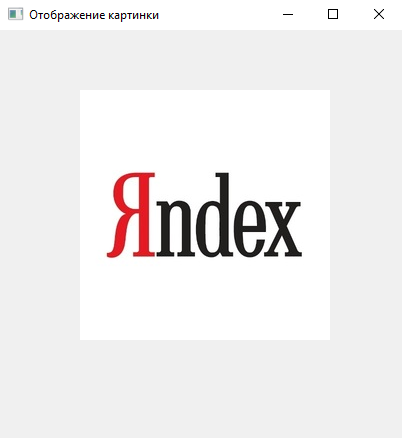
ex = Example()

ex.show()

sys.exit(app.exec())

Хотя раньше мы передавали размеры экрана напрямую в функцию, можно использовать список для хранения этих данных. Это может пригодиться в том случае, если у вас в программе много окон разного размера.

Обратите ваше внимание на следующий факт: QPixmap не используется для «показывания». В нем только хранится объект — картинка. Для отображения используется уже знакомый нам QLabel, в который с помощью функции setPixmap загружается наш объект.



**Диалоговые окна**

Но что нам делать, если мы хотим написать программу, которая позволяет открывать несколько файлов? Менять каждый раз код программы — неоптимально. Использовать input() для ввода имени файла? Но мы же работаем с графическим интерфейсом. Значит, можно использовать так называемые **диалоговые окна**.

Диалоговые окна нужны для того, чтобы получить какую-либо информацию от пользователя. Это может быть текстовая информация, цвет, настройки шрифта и даже файлы. В PyQT уже есть встроенные виджеты, реализующие различные диалоговые окна.

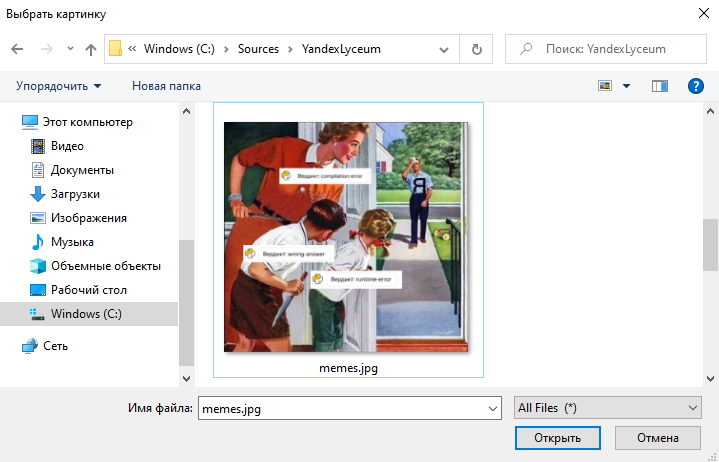
Будем получать путь к файлу с помощью **QFileDialog**. Важно не забыть импортировать этот виджет из **PyQt5.QtWidgets**.

Добавим в функцию загрузки из первой программы следующую строку:

fname = QFileDialog.getOpenFileName(self, 'Выбрать картинку', '')[0]

И будем передавать в QPixmap не имя файла, а переменную fname. Посмотрим, что будет происходить.

Перед запуском программы открывается окно выбора файла.



После выбора файла (после нажатия кнопки «Открыть»), окно закроется и откроется другое — уже знакомое нам по прошлой программе.

Давайте разберем, какие параметры можно передать в функцию создания диалога, для получения имени файла. Важно, что эта функция возвращает кортеж, состоящий из полного пути к файлу, а также его фильтр. Что такое фильтр?

Фильтр, который может быть передан в конструктор, — это набор типов, которые могут быть открыты с помощью данного диалогового окна. Например, мы хотим открывать только картинки формата .png. Чтобы не проверять каждый раз, подходит нам картинка или нет, мы можем передать в конструктор необходимый нам формат:

fname = QFileDialog.getOpenFileName(self, 'Выбрать картинку',

'', "Картинка(\*.png)")[0]

Кроме этого, можно передать заголовок окна, в нашем случае это «Выбрать картинку». Интересный факт: этот заголовок будет отображаться только в Windows, а в macOS будет игнорироваться.

А еще необходимо указать начальный путь для поиска файлов. Если передать пустые кавычки, поиск начнется в той же папке, где лежит запускаемый нами скрипт.

Кроме диалога, позволяющего выбрать файл, существуют и другие.

Рассмотрим виджет **QInputDialog**. Особенность этого виджета в том, что его можно настроить на получение различных типов данных: строки (одной или нескольких), числа или одного значения из списка.

Напишем программу, которая получает имя пользователя с помощью диалогового окна.

import sys

from PyQt5.QtWidgets import QWidget, QApplication, QPushButton

from PyQt5.QtWidgets import QInputDialog

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 150, 150)

self.setWindowTitle('Диалоговые окна')

self.button\_1 = QPushButton(self)

self.button\_1.move(20, 40)

self.button\_1.setText("Кнопка")

self.button\_1.clicked.connect(self.run)

self.show()

def run(self):

i, okBtnPressed = QInputDialog.getText(self, "Введите имя",

"Как тебя зовут?")

if okBtnPressed:

self.button\_1.setText(i)

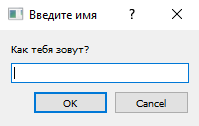
if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

ex = Example()

sys.exit(app.exec\_())

В функции, вызываемой нажатием на кнопку, создается диалоговое окно (QInputDialog) и указывается, какой тип оно будет возвращать (getText). В качестве параметров передаются родительское окно (self), заголовок нового окна и сообщение для пользователя. Возвращает эта функция кортеж данных, где на первом месте записаны введенные пользователем данные (или пустая строка, если нажата кнопка «Отмена»), а на втором — состояние кнопки «Ok».



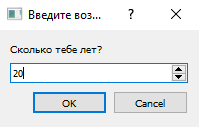
В случае если мы хотим получить целое число, указываем текущее значение, минимальное-максимальное значение и  шаг.

i, okBtnPressed = QInputDialog.getInt(self, "Введите возраст",

"Сколько тебе лет?",

20, 18, 27, 1)

В этом примере значение по умолчанию — 20, минимальное значение — 18, максимальное — 27, а шаг — 1.



Иногда нам нужно, чтобы пользователь выбрал какое-то значение из предоставленных. Для этого есть метод getItem.

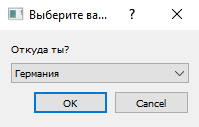
i, okBtnPressed = QInputDialog.getItem(self, "Выберите вашу страну",

"Откуда ты?",

("Россия", "Германия", "США"),

1, False)

В качестве параметров необходимо указать итерируемый объект и индекс значения по умолчанию. А чтобы пользователь не смог что-то самостоятельно ввести, укажем следующим параметром False.



Для выбора цвета существует другой виджет — **QColorDialog**. Вот пример программы с его использованием:

import sys

from PyQt5.QtWidgets import QWidget, QApplication, QPushButton

from PyQt5.QtWidgets import QColorDialog

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 150, 150)

self.setWindowTitle('Диалоговые окна')

self.button\_1 = QPushButton(self)

self.button\_1.move(20, 40)

self.button\_1.setText("Кнопка")

self.button\_1.clicked.connect(self.run)

self.show()

def run(self):

color = QColorDialog.getColor()

if color.isValid():

self.button\_1.setStyleSheet("background-color: {}".format(

color.name()))

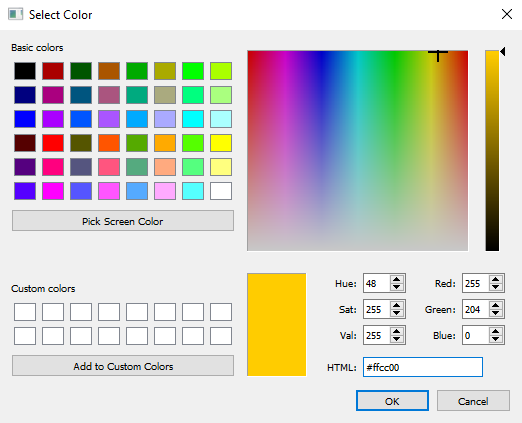
if \_\_name\_\_ == '\_\_main\_\_':

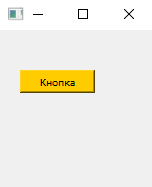
app = QApplication(sys.argv)

ex = Example()

sys.exit(app.exec\_())

Если цвет валиден, то фон кнопки окрасится в него. Функция .name() возвращает цвет в шестнадцатитиричном формате.





**Рисование**

До этого момента, чтобы создать и вывести какое-либо изображение, надо было создавать картинку с помощью PIL, а затем выводить ее, используя виджет QPixmap. Но в PyQT есть модули, которые позволяют рисовать прямо на поле. Посмотрим, как с ними работать.

import sys

from PyQt5.QtWidgets import QWidget, QApplication

from PyQt5.QtGui import QPainter, QColor

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 200, 200)

self.setWindowTitle('Рисование')

self.show()

def paintEvent(self, event):

qp = QPainter()

qp.begin(self)

self.drawFlag(qp)

qp.end()

def drawFlag(self,qp):

qp.setBrush(QColor(255, 0, 0))

qp.drawRect(30, 30, 120, 30)

qp.setBrush(QColor(0, 255, 0))

qp.drawRect(30, 60, 120, 30)

qp.setBrush(QColor(0, 0, 255))

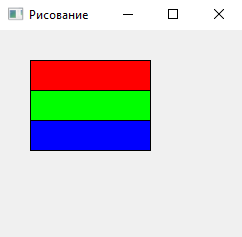
qp.drawRect(30, 90, 120, 30)

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

app = QApplication(sys.argv)

ex = Example()

sys.exit(app.exec\_())



В методе paintEvent происходит инициализация экземпляра класса QPainter, который отвечает за рисование на виджетах. Рисовать нужно между вызовами методов begin и end.

В функции drawFlag происходит непосредственно рисование. С помощью метода setBrush мы задаем цвет кисти в формате RGB. Для того чтобы преобразовать цвет в нужный нам тип, используем метод QColor. Чтобы нарисовать прямоугольник, применяем метод drawRect. В качестве параметров ему передаются координаты левого верхнего угла, длина и высота.

Кроме прямоугольников, получится нарисовать еще массу всего. Узнать о других методах можно в документации.

Во всех функциях в качестве параметра присутствует **event** — событие. Это событие, на которое подвешивается обработчик. Это может быть движение, нажатие или отпускание кнопки мыши, нажатие или отпускание клавиши и многое другое.

Напишем программу, которая будет рисовать пятиконечную звездочку. Для этого необходимо рассчитать точки. Пятиконечная звезда получается из правильного пятиугольника, если соединять точки через одну. Так что сначала нарисуем пятиугольник, а затем — звезду. Важно помнить, что наши координаты, в которых мы привыкли работать, могут не совпадать с экранными.

Для начала напишем функции, которые будут пересчитывать координаты в экранные.

def xs(self,x):

return x + SCREEN\_SIZE[0] // 2

def ys(self,y):

return SCREEN\_SIZE[1] // 2 - y

А теперь вместо функции drawFlag из прошлой программы напишем функцию drawStar. Для ее работы, кроме тех модулей, про которые мы говорили выше, нам понадобятся объекты следующих модулей: Qt, QPen, math. Так что важно не забыть импортировать их.

from PyQt5.QtGui import QPen

from math import pi, cos, sin

from PyQt5.QtCore import Qt

def drawStar(self,qp):

# Задаем длину стороны и количество углов

RAD = 100

p = 5

# Считаем координаты и переводим их в экранные

nodes = [(RAD \* cos(i \* 2 \* pi / p), RAD \* sin(i \* 2 \* pi / p)) for i in range(p)]

nodes2 = [(self.xs(node[0]), self.ys(node[1])) for node in nodes]

# Рисуем пятиугольник

for i in range(-1, len(nodes2) - 1):

qp.drawLine(\*nodes2[i], \*nodes2[i + 1])

# Изменяем цвет линии

pen = QPen(Qt.red, 2)

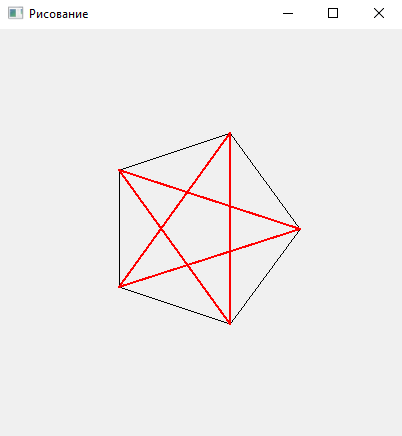
qp.setPen(pen)

# Рисуем звезду

for i in range(-2, len(nodes2) - 2):

qp.drawLine(\*nodes2[i], \*nodes2[i + 2])

Сначала мы высчитываем координаты и переводим их в экранные. Затем мы используем функцию drawLine, в нее поступают координаты двух точек, между которыми строится линия. Пройдясь по всем точкам, мы получим пятиугольник. По умолчанию рисование происходит с помощью «кисти» черного цвета и с толщиной, равной единице. Создадим новую кисть pen = QPen(Qt.red, 2) и применим ее. А затем, пробегаясь по тем же точкам, но через одну, получим изображение звезды.



**L-системы**

В 1968 г. венгерский биолог и ботаник Аристид Линденмайер (Aristid Lindenmayer) предложил математическую модель для изучения развития простых многоклеточных организмов, которая позже была расширена и используется для моделирования сложных ветвящихся структур — разнообразных деревьев и цветов.

Эта модель получила название Lindenmayer System, или просто L-System. Свои идеи Линденмайер изложил в книге «Алгоритмическая красота растений» (The Algorithmic Beauty of Plants).

Идею можно объяснить так: сложный объект получается из простого с помощью перезаписи частей этого простого объекта по определенным ранее правилам. L-система является частным случаем фракталов.

Представьте, что вы — художник. Но у вас есть только карандаш, поле для творчества — лист бумаги, и вы умеете выполнять только определенный набор команд (например, «поставить карандаш на бумагу», «поднять карандаш в воздух», «нарисовать линию», «повернуться на угол» и т.д.) Для того чтобы нарисовать на листе бумаги картину, вы получаете задание — строку символов, которую вам готовит L-автомат. Назовем эту строку **L-строкой**.

Итак, первый этап решения задачи — формирование L-строки, второй — построение по этой строке рисунка.

L-автомат не может сформировать L-строку «из неоткуда». Поэтому, у него есть **аксиома** — строка, которая определяет начало эволюции. Для дальнейшего развития L-автомат использует **теоремы** — правила, по которым и будет происходить преобразование L-строки.

Чтобы построить L-систему мы определим символы, которыми кодируется L-строка и опишем действия, которые производит художник, обрабатывая любой из символов.

|  |  |
| --- | --- |
| **Символ** | **Действие** |
| F | Начертить отрезок из точки, в которой находится карандаш под текущим углом, и остаться в новой точке. |
| f | Перенести карандаш из точки под текущим углом и остаться в новой точке, но не чертить отрезок. |
| + | Повернуться на заданный угол по часовой стрелке. |
| - | Повернуться на заданный угол против часовой стрелки. |
| [ | Сохранить текущее состояние карандаша. |
| ] | Вернуться в предыдущее сохраненное состояние карандаша. |
| | | Повернуться на 180 градусов. |

Необходимо пояснить некоторые моменты. Что такое **текущий угол** и **состояние карандаша**? Дело в том, что перед построением рисунка мы должны определиться со значением угла, на который мы будем поворачиваться в процессе работы, а под состоянием карандаша мы будем понимать набор из координат карандаша на листе и значения угла. Договоримся, что изначально угол равен нулю, а карандаш расположен в начале координат. Кроме того, надо знать длину шага для рисования.

Мы уже упомянули, что каждая фигура определяется своей аксиомой и набором теорем.

Например, аксиома может быть задана так:

F

Это означает, что наша система изначально представлена L-строкой F. То есть для того чтобы ее построить, мы возьмем из списка правил действие для строки F и выполним его, то есть построим линию.

Допустим, у нас есть только одна теорема и она выглядит так:

F -> F-F++F-F

Это означает, что на каждом шаге для получения новой эволюции L-строки мы должны каждый символ F заменить на последовательность: F-F++F-F.

Посмотрите на то, что будет представлять L-строка на первых 3-х шагах алгоритма.

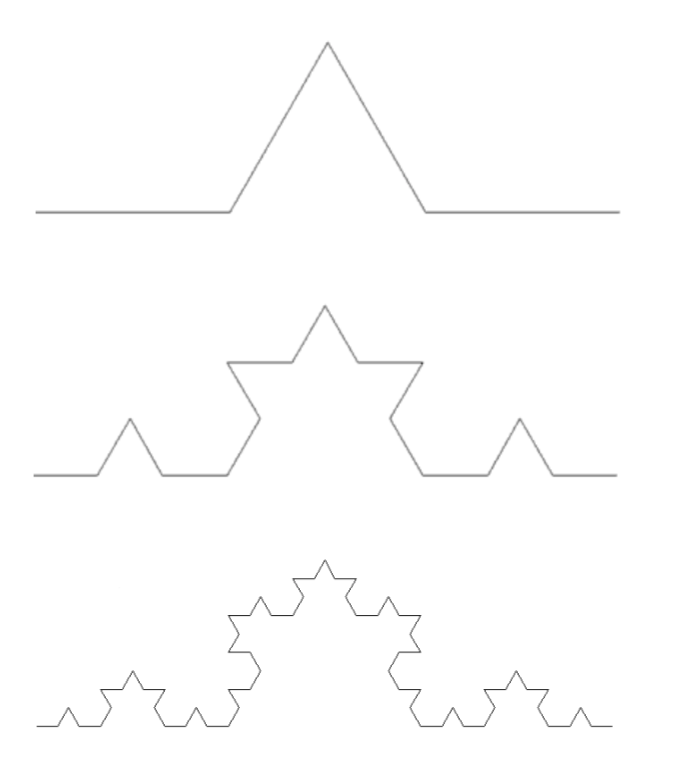
|  |  |
| --- | --- |
| **Этап** | **L-строка** |
| 0 | F |
| 1 | F-F++F-F |
| 2 | F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F |
| 3 | F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F-F-F++F-F-F-F++F-F++F-F++F-F-F- F++F-F |

В результате L-автомат будет содержать L-строку, которая описывает заданный фрактал — кривую Коха.

Вспомним про начальные условия и зададим угол поворота в 60 градусов.

Нулевой шаг алгоритма, L=F — рисуем отрезок.

Первый шаг алгоритма, L=F-F++F-F — рисуем отрезок, затем поворачиваемся против часовой стрелки на 60 градусов, затем опять рисуем отрезок, затем следует поворот на 120 градусов по часовой стрелке, рисование отрезка, поворот на 60 градусов против часовой стрелки и — ура — последний отрезок.



Рассмотрим более сложную L-систему:

Угол: 90 градусов

Аксиома: X

Теоремы:

X => -YF+XFX+FY-

Y => +XF-YFY-FX+

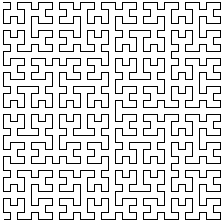
В этой L-системе присутствуют символы, которые не входят в перечень действий карандаша. В этом нет ничего страшного. При создании L-строки такие символы будут использованы, а при построении фрактала — нет, они будут просто проигнорированы.

|  |  |
| --- | --- |
| **Этап** | **L-строка** |
| 0 | X |
| 1 | -YF+XFX+FY- |
| 2 | -+XF-YFY-FX+F+-YF+XFX+FY-F-YF+XFX+FY-+F+XF-YFY-FX+- |

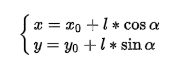
При построении первого шага мы:

1. Повернемся на 90 градусов против часовой стрелки
2. Y – проигнорируем
3. Нарисуем отрезок
4. Повернемся на 90 градусов по часовой стрелке
5. Х – проигнорируем
6. Нарисуем отрезок
7. Опять проигнорируем Х
8. Повернемся на 90 градусов по часовой стрелке
9. Нарисуем отрезок
10. Y – проигнорируем
11. Повернемся на 90 градусов против часовой стрелки

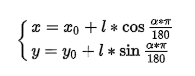
И в результате получим ... кривую Гильберта. Нужно отметить, что фрактал нулевой глубины в данном случае не имеет картинки, поскольку L-строка содержит символ «X», который никак не отображается. А вот для фрактала пятой глубины получим:



Остановимся немного на поворотах. Допустим ваш карандаш находится в точке с координатами (x0, y0), а текущий угол — α. Пусть длина шага равна — l. Тогда координаты карандаша после окончания рисования можно вычислить по следующим формулам:



При этом углы должны быть указаны в **радианах**. Если же мы указываем углы в градусах, то формулы приобретают следующий вид:



Поворот на определенный угол сводится к следующему:

alpha = (alpha + phi) % 360

Все необходимые функции: синус, косинус, перевод градусов в радианы и обратно — можно найти в модуле math, который входит в стандартную библиотеку.

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты