Тема 2.
Событийно-ориентированное программирование.

Лекция 6. Аппаратно–независимый ввод/вывод

Обо мне

- ФОТО

- Харченко Владислав Алексеевич, 26 лет.

- 2012 – 2017 гг. ВГТУ. Факультет Радиотехники и Электроники.
- 2014 – 2018 гг. Фриланс.
- 2017 – 2018 гг. АО «НИИ СВТ». Программист.
- 2018 – 2021 гг. АО «НИИ СВТ». Старший программист.

Учебные вопросы

1. Окно загрузки
2. Работа с графикой
3. Графическая сцена

Источники

- Официальная документация: https://doc.qt.io/qtforpython
- Прохоренок Н. А., Дронов В. А. Python 3 и PyQt 5. Разработка приложений. 2019 г.

Используемые в курсе инструменты для разработки		
IDE	PyCharm CE	https://www.jetbrains.com/pycharm/download
Окружение	Virtualenv	https://docs.python.org/3/library/venv.html
VSC (рекомендовано)	GIT	https://git-scm.com
Фреймворк	PySide2	https://doc.qt.io/qtforpython/

1. Окно загрузки.

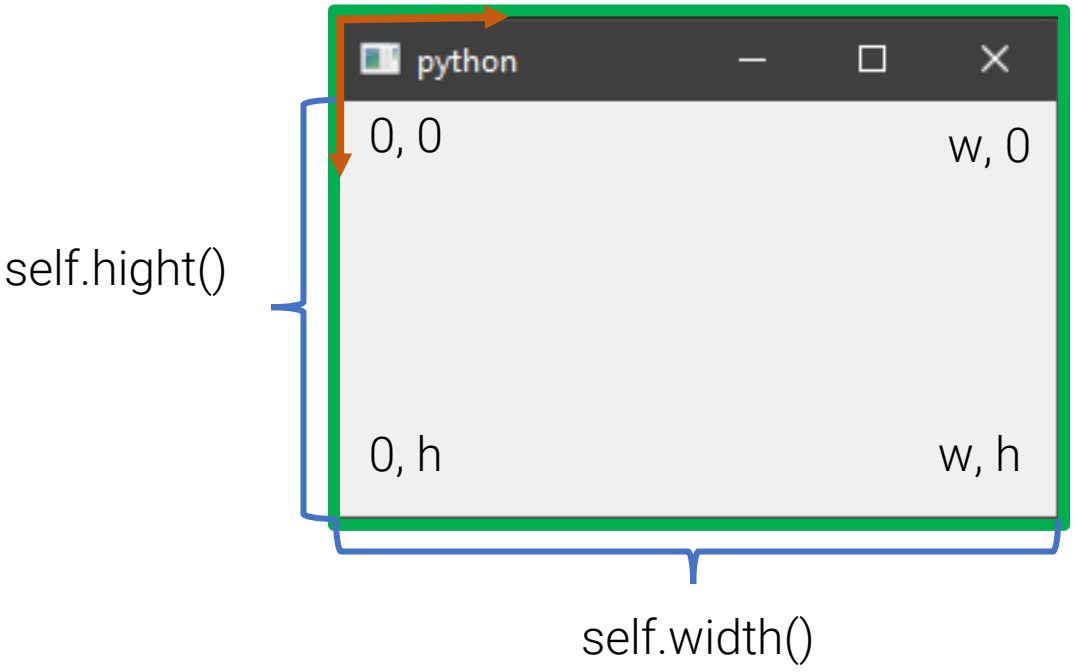
Вывод окна при длительной загрузке приложения

- Для вывода всплывающего окна перед появлением GUI самого, необходимо:
- В конструкции `if __name__ == "__main__":` необходимо создать экземпляр класса QSplashScreen в конструктор которого через конструкцию `QtGui.QPixmap("1.png")` передаём нужное для отображения изображение;
- Пример: `splash = QtWidgets.QSplashScreen(QtGui.QPixmap("1.png"))`
- Методы QSplachScreen:
  - show() – служит для отображения QSplahScreen;
  - finish() – необходим для закрытия QSplashScreen;
  - showMessage() – отображение всплывающего сообщения;
  - setPixmap() – установить изображение не через конструктор;
  - clearMessage() – стирает недпись;

2. Работа с графикой

Класс QPainter

- Класс QPainter выполняет рисование на виджетах;
- Основное применение:
  - Рисование простых фигур;
  - Создание собственных виджетов.
- Координатная сетка:



Координатная сетка исчисляется слева направо, сверху вниз

Простейшая графика

- drawLine(x1, y1, x2, y2) – обычная линия;

```
def paintEvent(self, event: QtGui.QPaintEvent) -> None:
    painter = QtGui.QPainter(self)
    x1, y1 = 10, 10
    x2, y2 = 100, 60
    painter.drawLine(x1, y1, x2, y2)
    p1 = QtCore.QPoint(10, 10)
    p2 = QtCore.QPoint(100, 60)
    painter.drawLine(p1, p2)
```

Через задание координат

Через класс QPoint

- drawLines(list: QLine) – несколько простых линий;

```
def paintEvent(self, event: QtGui.QPaintEvent) -> None:
    painter = QtGui.QPainter(self)
    painter.drawLines([QtCore.QLine(10, 10, 100, 60),
                       QtCore.QLine(100, 60, 20, 30)])
```

Простейшая графика (продолжение)

- drawRect(x1, y1, w, h) – прямоуголник;

```
def paintEvent(self, event: QtGui.QPaintEvent) -> None:
    painter = QtGui.QPainter(self)
    painter.drawRect(10, 10, 100, 60)
```

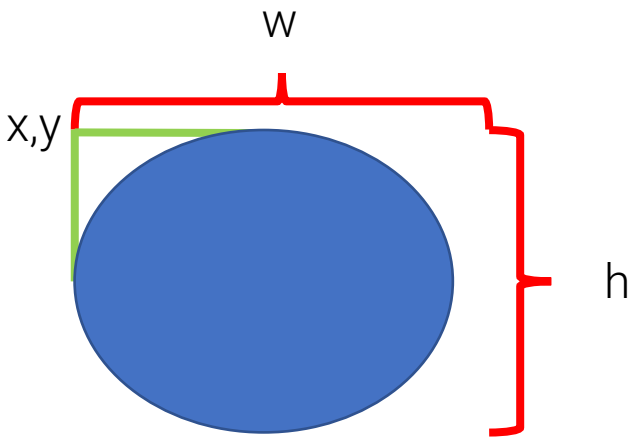
- drawRects(list: QRect) – несколько прямоугольников;

```
def paintEvent(self, event: QtGui.QPaintEvent) -> None:
    painter = QtGui.QPainter(self)
    rects = [QtCore.QRect(10, 10, 100, 60),
             QtCore.QRect(20, 20, 100, 60)]
    painter.drawRects(rects)
```

Простейшая графика (продолжение)

- drawArc(x, y, w, h, a, alen) – дуга;
  - x, y, w, h – <int> – область рисования дуги;
  - a – начальный угол (1/16 градуса);
  - alen – длина дуги (1/16 градуса);

0 = 0
90 = 16*90 = 1440
180 = 16*180 = 2880
270 = 16*270 = 4320
360 = 16*360 = 5760




```
def paintEvent(self, event: QtGui.QPaintEvent) -> None:
    painter = QtGui.QPainter(self)
    painter.drawArc(50, 20, 100, 70, 16*0, 16*211)
```

*1/16 – для большей точности

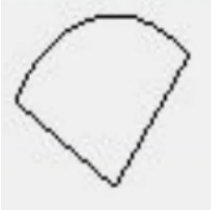
Простейшая графика (продолжение)

- drawPolygon(list: QPoint) – многоугольник;

- drawPoint() – точка;
- drawChord() – дуга с хордой;

- drawPolyline() – множество линий;

- drawPie() – сектор.



Классы QPen и QBrush

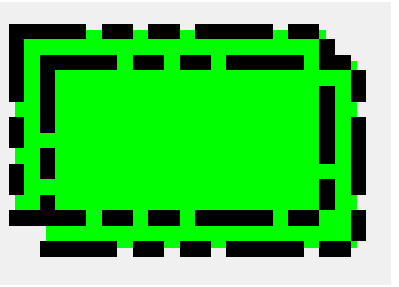
```
def paintEvent(self, event: QtGui.QPaintEvent) -> None:
    painter = QtGui.QPainter(self)
    pen = QtGui.QPen()
    pen.setWidth(5)
    pen.setStyle(QtCore.Qt.DashDotDotLine)
    brush = QtGui.QBrush(QtCore.Qt.green)

    painter.setPen(pen)
    painter.setBrush(brush)

    rects = [QtCore.QRect(10, 10, 100, 60),
             QtCore.QRect(20, 20, 100, 60)]
    painter.drawRects(rects)
```

QPen – тип рисуемой фигуры

QBrush – тип заполнения фигуры



Стили контура фигуры

Qt.SolidLine

Qt.DashLine

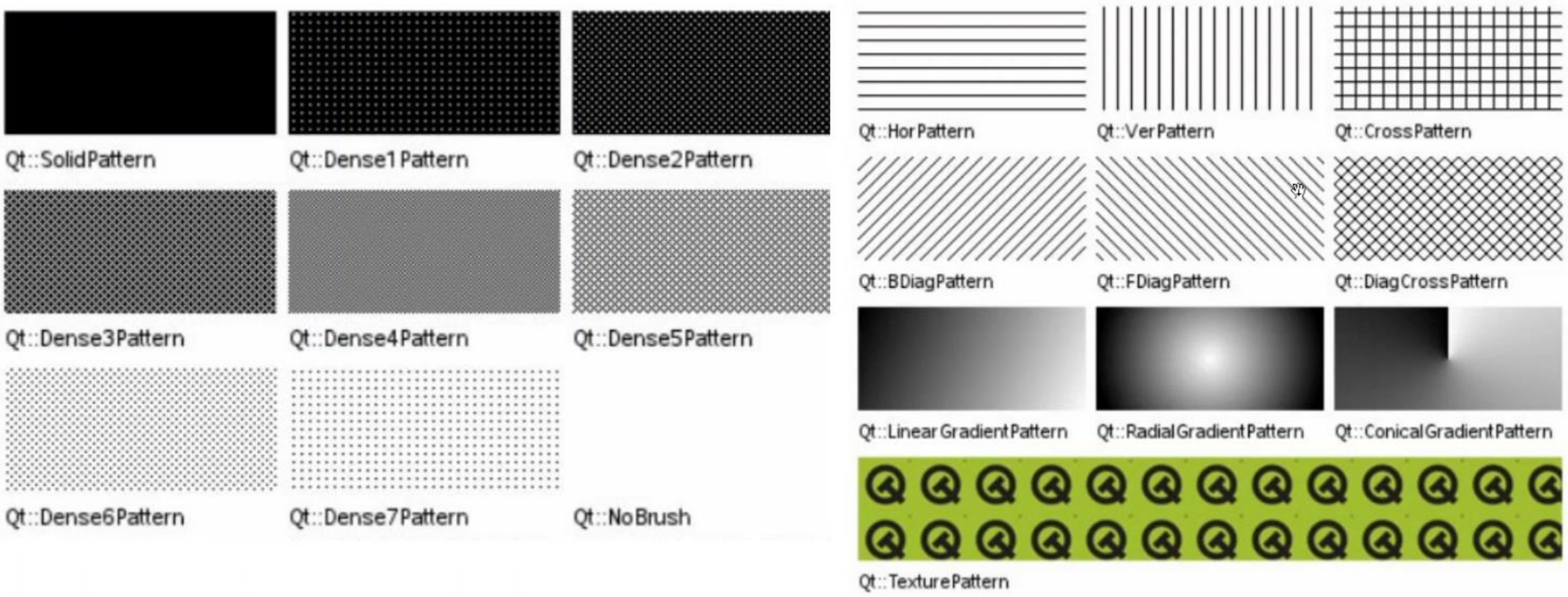
Qt.DotLine

Qt.DashDotLine

Qt.DashDotDotLine

Qt.CustomDashLine

Стили QBrush



3. Графическая сцена

Класс QPixmap

- Список поддерживаемых форматов:

```
for i in QtGui.QImageReader.supportedImageFormats():
    print(str(i, "ascii"), end=" ")
```

```
bmp cur gif icns ico jpeg jpg pbm pdf pgm png ppm svg svgz tga tif tiff wbmp webp xbm xpm
```

*gif – можно прочитать, но нельзя сохранить, т.к. алгоритм сжатия gif, защищён патентом

- Форматы конструктора:

```
pxm = QtGui.QPixmap()
pxm = QtGui.QPixmap(w=10, h=10)
pxm = QtGui.QPixmap(QtCore.QSize(100, 100))
pxm = QtGui.QPixmap(fileName="somePath.jpg", format="JPG")
pxm = QtGui.QPixmap(somePxm)
```

- Основные методы:
  - load()
  - loadFromData()
  - save()
  - convertFromImage()
  - scaled()

- Qbitmap – подкласс QPixmap. Обеспечивает обработку монохромных изображений глубиной 1 бит.

Класс QImage

- Обеспечивает аппаратно – независимое представление для ввода/вывода изображений, а также для прямого доступа к пикселям и манипуляциям с ними.
- Формат конструктора:

```
img = QtGui.QImage()
img = QtGui.QImage(width=10, height=10)
img = QtGui.QImage(QtCore.QSize(100, 100),
                   QtGui.QImage.Format_ARGB32_Premultiplied)
img = QtGui.QImage(fileName="someImg.png")
img = QtGui.QImage(QtGui.QImage())
```

- Методы класса, практически схожи с QPixmap, т.к. авторы Qt хотели, чтобы два класса были унифицированы друг с другом.

Основные отличия QPixmap и QImage

QPixmap	QImage
Служит только для отображения изображений в программе	Служит для работы с данными изображения, для доступа к пикселям и манипуляции с ними
Если изображение будет отображаться более 2х раз рекомендуется использовать QPixmap	Можно управлять QImage вне GUI потока

Спасибо за внимание!