PYQT6 Course

QPainter Class

QPainter provides highly optimized functions to do most of the drawing GUI programs require. It can draw everything from simple lines to complex shapes like pies and chords. It can also draw aligned text and pixmaps. Normally, it draws in a "natural" coordinate system, but it can also do view and world transformation. QPainter can operate on any object that inherits the QPaintDevice class.
The common use of QPainter is inside a widget's paint event: Construct and customize (e.g. set the pen or the brush) the painter.

The core functionality of QPainter is drawing, but the class also provide several functions that allows you to customize QPainter's settings and its rendering quality, and others that enable clipping. In addition, you can control how different shapes are merged together by specifying the painter's composition mode.

There are several settings that you can customize to make QPainter draw according to your preferences:

- font() is the font used for drawing text. If the painter isActive(), you can retrieve information about the currently set font, and its metrics, using the fontInfo() and fontMetrics() functions respectively.
- brush() defines the color or pattern that is used for filling shapes.
- pen() defines the color or stipple that is used for drawing lines or boundaries.
- backgroundMode() defines whether there is a background() or not, it is either
  either Qt.BGMode.OpaqueMode or Qt.BGMode.TransparentMode.
-
- background() only applies when backgroundMode()
  is Qt.BGMode.OpaqueMode and pen() is a stipple. In that case, it describes the color of the background pixels in the stipple.
- brushOrigin() defines the origin of the tiled brushes, normally the origin of widget's background.
- viewport(), window(), worldTransform() make up the painter's coordinate transformation system.
- hasClipping() tells whether the painter clips at all. (The paint device clips, too.) If the painter clips, it clips to clipRegion().
- layoutDirection() defines the layout direction used by the painter when drawing text.

- worldMatrixEnabled() tells whether world transformation is enabled.
- viewTransformEnabled() tells whether view transformation is enabled.

QPainter provides functions to draw most primitives: drawPoint(), drawPoints(), drawLine(), drawRect(), drawRoundedRect(), drawEllipse(), drawArc(), drawPie(), drawChord(), drawPolyline(), drawPolygon(), drawConvexPolygon() and drawCubicBezier(). The two convenience functions, drawRects() and drawLines(), draw the given number of rectangles or lines in the given array of QRects or QLines using the current pen and brush.

QGradient Class

The QGradient class is used in combination with QBrush to specify gradient fills, Qt currently supports three types of gradient fills:

- *Linear* gradients interpolate colors between start and end points.
- *Simple* radial gradients interpolate colors between a focal point and end points on a circle surrounding it.
- *Extended* radial gradients interpolate colors between a center and a focal circle.

- *Conical* gradients interpolate colors around a center point.

QGraphicsView Class

The QGraphicsView class provides a widget for displaying the contents of a QGraphicsScene, QGraphicsView visualizes the contents of a QGraphicsScene in a scrollable viewport. To create a scene with geometrical items, QGraphicsView is part of the Graphics View Framework.
To visualize a scene, you start by constructing a QGraphicsView object, passing the address of the scene you want to visualize to QGraphicsView's constructor. Alternatively, you can call setScene() to set the scene at a later point. After you call show(), the view will by default scroll to the center of the scene and display any items that are visible at this point.

QGraphicsScene Class

The QGraphicsScene class provides a surface for managing a large number of 2D graphical items, the class serves as a container for QGraphicsItems. It is used together with QGraphicsView for visualizing graphical items, such as lines, rectangles, text, or even

custom items, on a 2D surface. QGraphicsScene is part of the Graphics View Framework.
QGraphicsScene also provides functionality that lets you efficiently determine both the location of items, and for determining what items are visible within an arbitrary area on the scene. With the QGraphicsView widget, you can either visualize the whole scene, or zoom in and view only parts of the scene.

QGraphicsRectItem Class

The QGraphicsRectItem class provides a rectangle item that you can add to a QGraphicsScene, To set the item's rectangle, pass a QRectF to QGraphicsRectItem's constructor, or call the setRect() function. The rect() function returns the current rectangle.