Форматирование в QTextEdit, работа с изображениями

Вставка изображений в текст, эффекты.

[QTextEdit](#_ghd9r0svjpor)

[Основные методы](#_2pz1myuserf9)

[Указание параметров текста и фона](#_15c0vrk3toh9)

[Форматирование текста](#_iauvst8xx1kq)

[Смайлы в тексте](#_we1ct0i4inec)

[Изображения](#_t7bm8ba5mavw)

[Оттенки серого](#_3jnlq6pq82ec)

[Эффект сепии](#_4fjmgyffuybk)

[Эффект негатива](#_2udmqa1uthgi)

[Черно-белое изображение](#_x89ua6gvssyb)

[Загрузка изображений с устройства](#_o4cce0cc4a38)

[Практика](#_a7fiora8azhg)

[Практическое задание](#_1t3h5sf)

[Дополнительные материалы](#_4d34og8)

[Используемая литература](#_2s8eyo1)

# QTextEdit

**QTextEdit** поддерживает технологию ***drag and drop***, работу с буфером обмена и стандартными комбинациями клавиш быстрого доступа. Компонент — многострочное текстовое поле, предназначенное для ввода и редактирования текста. Он может быть в форматеHTML или простого текста.

У конструктора два формата вызова:

|  |
| --- |
| <Объект> = QTextEdit([parent=<Родитель>]) <Объект> = QTextEdit (<Текст> [, parent=<Родитель>]) |

## Основные методы

1. **setText (<Текст>)** — помещает в поле текст, который может быть простым или в формате HTML.
2. **setPlainText (<Текст>)** — помещает простой текст в поле.
3. **setHtml (<Текст>)** — помещает в поле текст в формате HTML.
4. **insertPlainText (<Текст>)** — вставляет простой текст в текущую позицию текстового курсора. Если в поле был выделен фрагмент текста, он будет удален.
5. **insertHtml (<Текст>)** — вставляет текст в формате HTML в текущую позицию курсора. Если был выделен фрагмент текста, он удалится.
6. **append (<Текст>)** — добавляет новый абзац с указанным текстом в формате HTML в конец поля.
7. **toPlainText ()** — возвращает простой текст, содержащийся в текстовом поле.
8. **toHtml ()** — возвращает текст в формате HTML.
9. **clear ()** — удаляет весь текст из поля.
10. **selectAll ()** — выделяет весь текст в поле.
11. **cut ()** — копирует выделенный текст в буфер обмена и удаляет его из поля.
12. **сору ()** — копирует выделенный текст в буфер обмена.
13. **paste ()** — вставляет текст из буфера обмена в текущую позицию текстового курсора, если поле доступно для редактирования.

## Указание параметров текста и фона

1. **setCurrentFont (<QFont>)** — задает шрифт. В качестве параметра указывается экземпляр класса **QFont** из модуля **QtGui**. Конструктор этого класса выглядит так:

|  |
| --- |
| <Шрифт> = QFont(<Название шрифта>[, pointSize=-1][, weight=-1] [, italic=False]) |

В первом параметре в виде строки задается название шрифта, а необязательный параметр **pointsize** устанавливает его размер. В параметре **weight** можно указать степень жирности шрифта — числом от 0 до 99 или значением атрибутов **Light, Normal, DemiBold, Bold или Black** класса **QFont**. Если в параметре **italic** указано **True**, шрифт будет курсивным.

1. **currentFont ()** — возвращает экземпляр класса **QFont** с текущими характеристиками шрифта.
2. **setFontFamily (<Название шрифта>)** — задает название шрифта.
3. **setFontPointSize (<Размер>)** — задает размер шрифта.
4. **fontPointsize ()** — возвращает размер шрифта.
5. **setFontWeight (<Жирность>)** — задает жирность шрифта.
6. **fontweight ()** — возвращает жирность шрифта.
7. **setFontitalic (<Флаг>)** — если в параметре указано **True**, шрифт будет курсивным.
8. **fontltalic()** — возвращает **True**, если шрифт курсивный, и **False** — если нет.
9. **setFontUnderline (<Флаг>)** — если в параметре указано **True**, текст будет подчеркнутым.
10. **fontUnderline ()** — возвращает **True**, если текст подчеркнутый, и False — если нет.
11. **setTextColor (<QColor>)** — задает цвет текста. В качестве значения можно указать атрибут класса **QtCore.Qt** (например, **black**, **white**) или экземпляр класса **QColor** из модуля **QtGui** (**QColor ("red")**, **QColor ("#ff0000")**, **QColor(255, 0, 0)** и другие).
12. **textColor ()** — возвращает экземпляр класса **QColor** с цветом текущего текста.
13. **setTextBackgroundColor (<QColor>)** — задает цвет фона. В качестве значения можно указать атрибут из класса **QtCore.Qt** или экземпляр класса **QColor**.
14. **textBackgroundColor ()** — возвращает экземпляр класса **QColor** с цветом фона.

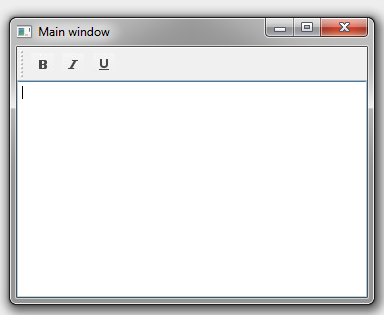
## 

## Форматирование текста

Создадим простое форматирование в поле **QTextEdit**. Нажимая кнопки, будем задавать стиль текста в строке. Создадим базовый каркас приложения. В функции **initUI()** напишем следующее:

|  |
| --- |
| self.textEdit = QTextEdit() self.setCentralWidget(self.textEdit)  bold = QAction(QIcon('b.jpg'),'Bold', self)  italic = QAction(QIcon('i.jpg'), 'Italic', self)  underlined = QAction(QIcon('u.jpg'), 'Underlined', self)  toolbar = self.addToolBar('Formatting') toolbar.addAction(bold) toolbar.addAction(italic) toolbar.addAction(underlined)  self.setGeometry(300, 300, 350, 250) self.setWindowTitle('Main window') self.show() |

Результат:



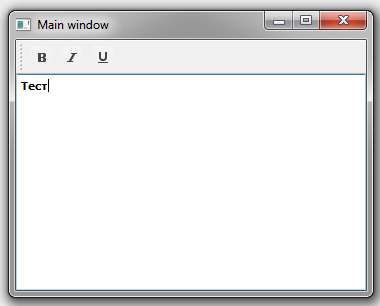
Создали текстовое поле и тулбар с кнопками, изображения для которых находятся в файлах примеров. Напишем обработчики событий для кнопок:

|  |
| --- |
| bold.triggered.connect(self.actionBold) italic.triggered.connect(self.actionItalic) underlined.triggered.connect(self.actionUnderlined) |

Добавим функции, которые будут отвечать за форматирование текста:

|  |
| --- |
| def actionBold(self):  myFont = QFont()  myFont.setBold(True)  self.textEdit.setFont(myFont)  def actionItalic(self):  myFont = QFont()  myFont.setItalic(True)  self.textEdit.setFont(myFont)  def actionUnderlined(self):  myFont = QFont()  myFont.setUnderline(True)  self.textEdit.setFont(myFont) |

В этих функциях задаем стиль шрифта. Нажав кнопку **B** (bold, или «жирный»), увидим:



## 

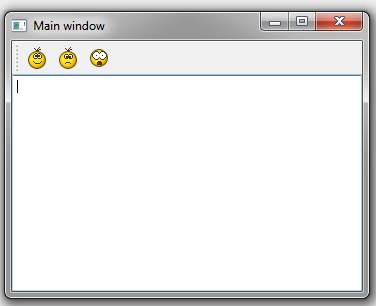
## Смайлы в тексте

Чтобы добавить смайлы в поле **QTextEdit**, будем пользоваться функцией **setHtml**.

Сначала напишем каркас для приложения:

|  |
| --- |
| self.textEdit = QTextEdit()   self.setCentralWidget(self.textEdit)   smile = QAction(QIcon('ab.gif'),'Smile', self)   melancholy = QAction(QIcon('ac.gif'), 'Melancholy', self)   surprise = QAction(QIcon('ai.gif'), 'Surprise', self)   toolbar = self.addToolBar('Formatting')  toolbar.addAction(smile)  toolbar.addAction(melancholy)  toolbar.addAction(surprise)   self.setGeometry(300, 300, 350, 250)  self.setWindowTitle('Main window')  self.show() |

В итоге получится:



Напишем обработку нажатий на смайлы:

|  |
| --- |
| smile.triggered.connect(self.actionSmile)  melancholy.triggered.connect(self.actionMelancholy)  surprise.triggered.connect(self.actionSurprise) |

Добавим функции, которые отвечают за обработку событий:

|  |
| --- |
| def actionSmile(self):  url = 'ab.gif'  self.textEdit.setHtml('<img src="%s" />' % url)  def actionMelancholy(self):  url = 'ac.gif'  self.textEdit.setHtml('<img src="%s" />' % url)  def actionSurprise(self):  url = 'ai.gif'  self.textEdit.setHtml('<img src="%s" />' % url) |

В итоге сможем добавлять в поле смайлы:

# 

# 

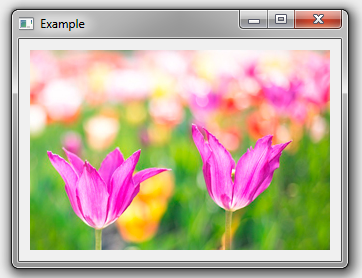
# Изображения

Чтобы преобразовывать изображения, будем использовать библиотеку **PIL**.

Сделаем каркас приложения:

|  |
| --- |
| hbox = QHBoxLayout(self)  pixmap = QPixmap("image.jpg")   lbl = QLabel(self)  lbl.setPixmap(pixmap)   hbox.addWidget(lbl)  self.setLayout(hbox)   self.move(300, 200)  self.setWindowTitle('Example')  self.show() |

Получилось:



## 

## Оттенки серого

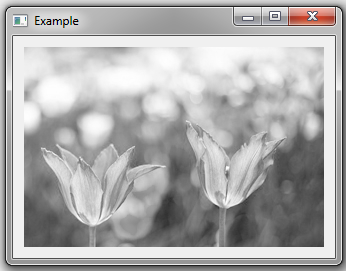
Чтобы получить оттенки серого, напишем код:

|  |
| --- |
| image = Image.open("image.jpg")  draw = ImageDraw.Draw(image)  width = image.size[0]  height = image.size[1]  pix = image.load()    for i in range(width):  for j in range(height):  a = pix[i, j][0]  b = pix[i, j][1]  c = pix[i, j][2]  S = (a + b + c)   draw.point((i, j), (S, S, S))   img\_tmp = ImageQt(image.convert('RGBA'))   hbox = QHBoxLayout(self)  pixmap = QPixmap.fromImage(img\_tmp) |

Подгрузим изображение и усредним каждый его пиксель. Преобразуем из **Image** в **QImage**:

|  |
| --- |
| img\_tmp = ImageQt(image.convert('RGBA')) |

Результат:

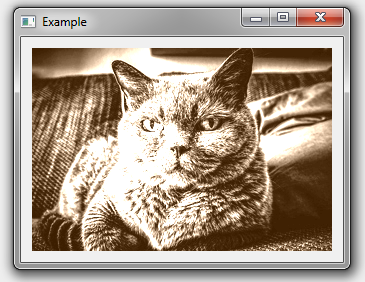


## Эффект сепии

Напишем код:

|  |
| --- |
| image = Image.open("image.jpg")  draw = ImageDraw.Draw(image)  width = image.size[0]  height = image.size[1]  pix = image.load()   depth = 30  for i in range(width):  for j in range(height):  a = pix[i, j][0]  b = pix[i, j][1]  c = pix[i, j][2]  S = (a + b + c)  a = S + depth \* 2  b = S + depth  c = S  if (a > 255):  a = 255  if (b > 255):  b = 255  if (c > 255):  c = 255  draw.point((i, j), (a, b, c))   img\_tmp = ImageQt(image.convert('RGBA')) |

Чтобы получить эффект сепии, усредняем каждый пиксель и к получившемуся значению прибавляем коэффициент — нашем случае 30:

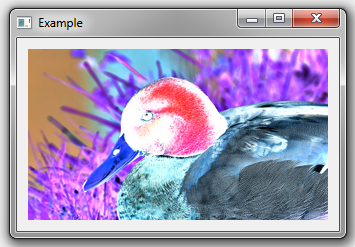


## Эффект негатива

Код:

|  |
| --- |
| image = Image.open("image.jpg")  draw = ImageDraw.Draw(image)  width = image.size[0]  height = image.size[1]  pix = image.load()   for i in range(width):  for j in range(height):  a = pix[i, j][0]  b = pix[i, j][1]  c = pix[i, j][2]  draw.point((i, j), (255 - a, 255 - b, 255 - c))   img\_tmp = ImageQt(image.convert('RGBA'))   hbox = QHBoxLayout(self)  pixmap = QPixmap.fromImage(img\_tmp) |

Это один из наиболее простых эффектов. Чтобы его получить, вычитаем из 255 значение каждого пикселя. Должно получиться примерно так:



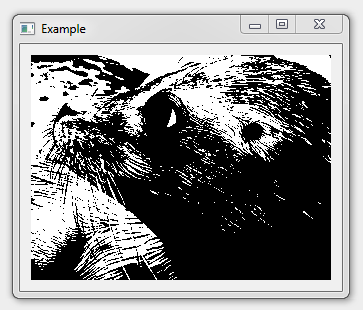
## 

## Черно-белое изображение

Напишем код:

|  |
| --- |
| image = Image.open("image.jpg")  draw = ImageDraw.Draw(image)  width = image.size[0]  height = image.size[1]  pix = image.load()   factor = 50  for i in range(width):  for j in range(height):  a = pix[i, j][0]  b = pix[i, j][1]  c = pix[i, j][2]  S = a + b + c  if (S > (((255 + factor) // 2) \* 3)):  a, b, c = 255, 255, 255  else:  a, b, c = 0, 0, 0  draw.point((i, j), (a, b, c))   img\_tmp = ImageQt(image.convert('RGBA')) |

Чтобы сделать изображение черно-белым, разбиваем пиксели на две группы: черные и белые. Будем смотреть, к какому цвету пиксель ближе, и распределять по группам. Получится так:



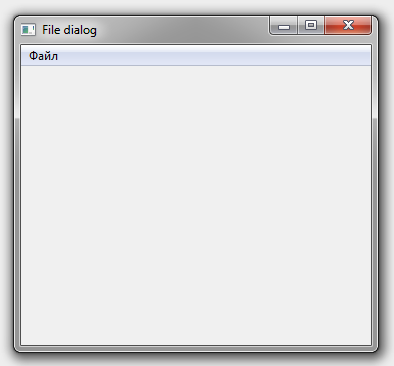
## 

## Загрузка изображений с устройства

Чтобы загрузить изображения, напишем такой код:

|  |
| --- |
| self.lbl = QLabel(self)   openFile = QAction(QIcon('open.png'), 'Open', self)  openFile.setShortcut('Ctrl+O')  openFile.setStatusTip('Открыть файл')   menubar = self.menuBar()  fileMenu = menubar.addMenu('&Файл')  fileMenu.addAction(openFile)   self.setGeometry(300, 300, 350, 300)  self.setWindowTitle('File dialog')  self.show() |

Создали окно с меню:



Напишем обработчик:

|  |
| --- |
| openFile.triggered.connect(self.showDialog) |

В нем вызываем функцию **showDialog**, в которой будет код загрузки изображения:

|  |
| --- |
| fname = QFileDialog.getOpenFileName(self, 'Open file', '/home')[0]  pixmap = QPixmap(fname)  self.lbl.resize(300,300)  self.lbl.setPixmap(pixmap) |

Функция **resize** масштабирует **QLabel** до размеров 300 на 300:

|  |
| --- |
| self.lbl.resize(300,300) |

Получилась программа, которая загружает изображения из компьютера.

# Практика

Если на уроке остается время, решаем следующие задачи:

1. Разработать простой текстовый редактор, который поддерживает функции форматирования текста: **жирный,** *курсив,* подчеркивание.
2. Добавить возможность в текстовом редакторе вставлять в документ картинку.

# Практическое задание

В этом курсе мы продолжаем разрабатывать приложение-мессенджер. Все задания следует делать локально, не используя базы данных. Сохранять можно в обычный текстовый файл.

1. Написать программу, которая будет загружать изображения с компьютера и добавлять к ним эффекты.
2. Добавить форматирование сообщений в мессенджере.
3. Реализовать возможность добавлять фотографии в ваш профиль.
4. \* Добавить смайлы в мессенджер.
5. \* Применить эффекты к изображению в профиле.

# Дополнительные материалы

1. Статья на Habr [«Играемся с изображениями в Python»](https://habrahabr.ru/post/163663/).
2. [Меню и тулбары в PyQt5](https://pythonworld.ru/gui/pyqt5-menustoolbars.html).
3. [Управление макетом в PyQt5](https://pythonworld.ru/gui/pyqt5-layout.html).

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [Учим Python качественно](https://habrahabr.ru/post/150302/).
2. [Самоучитель по Python](http://pythonworld.ru/samouchitel-python)
3. [Марк Лутц. Изучаем Python (4-е издание)](https://proklondike.net/books/python/lutz_python_2011.html).