

## Работа с библиотекой PyQt.

Для программ, которые не предполагают того, что их будет использовать неподготовленный к работе с командной строкой пользователь, текстового интерфейса хватает. Существует большое количество консольных утилит, предназначенных для программистов или системных администраторов, но часто в жизни разработчика наступает момент, когда разработанную программу надо передать незнакомому с консолью пользователю. Графический интерфейс (GUI — Graphical User Interface) более «дружелюбный» к пользователям, а если в программе необходимо отображать не только текст, но и графическую или мультимедийную информацию, его использование становится необходимостью.

Рассмотрим основные понятия концепции GUI. Допустим, есть знакомый нам текстовый редактор. Когда мы нажимаем клавишу X, возникает событие «Нажата клавиша X». В то же время внутри программы запускается обработчик, который проверяет, какая клавиша нажата, какая раскладка выбрана и так далее. Затем он выполняет действие: выводит нужный символ на экран. Общий принцип работы можно представить в виде схемы:



Для языка программирования Python есть много способов создания приложений с графическим интерфейсом, в частности, уже знакомая вам библиотека tkinter. Она используется в большом числе кроссплатформенных приложений, написанных на Python, но возможности библиотеки PyQt значительно богаче.

QT - это написанная на C++ библиотека с классами для создания графического интерфейса. Библиотека получилась настолько удачной, начала собирать вокруг себя большое сообщество программистов, которые разрабатывали приложения не только на C++, но и на других языках программирования. Это привело к тому, что и для других языков программирования стали появляться свои библиотеки-«обертки» для QT. Для Python это PyQt.

В терминологии PyQt (и достаточно большого числа других библиотек создания GUI) все графические приложения состоят из виджетов.

Виджет - минимальный элемент графического интерфейса пользователя.

В библиотеке PyQt5 существует множество модулей, но чаще других используется QtWidgets. Именно в нем находятся классы, соответствующие различным элементам интерфейса.

Напишем простейшую программу с использованием библиотеки PyQt5:

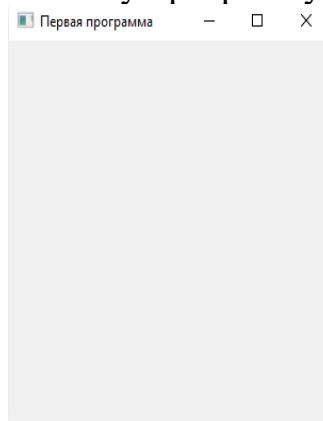
```
# Импортируем из PyQt5.QtWidgets классы для создания приложения и виджета
from PyQt5.QtWidgets import QApplication, QWidget

# Унаследуем наш класс от простейшего графического примитива QWidget
class Example(QWidget):
    def __init__(self):
        # Надо не забыть вызвать инициализатор базового класса
        super().__init__()
        # В метод initUI() будем выносить всю настройку интерфейса,
        # чтобы не перегружать инициализатор
        self.initUI()

    def initUI(self):
        # Зададим размер и положение нашего виджета,
        self.setGeometry(300, 300, 300, 300)
        # А также его заголовок
        self.setWindowTitle('Первая программа')

if __name__ == '__main__':
    # Создадим класс приложения PyQt
    app = QApplication(sys.argv)
    # А теперь создадим и покажем пользователю экземпляр
    # нашего виджета класса Example
    ex = Example()
    ex.show()
    # Будем ждать, пока пользователь не завершил исполнение QApplication,
    # а потом завершим и нашу программу
    sys.exit(app.exec())
```

Если мы запустим эту программу, увидим такое окно:



Обратите внимание на класс - Example. Он наследуется от базового класса QWidget, который определяет простейшее окно. От него наследуется много встроенных виджетов.

Первое, что можно увидеть в классе, — перегруженный конструктор.

`super().__init__()` - эта строка вызывает конструктор родительского класса. Потом вызывается метод класса с названием `initUI`.

Разумеется, инициализацию интерфейса можно реализовать и в инициализаторе класса, но хорошей практикой считается вынос этой функциональности в отдельный метод класса с названием `initUI`.

Обратите внимание: метод `initUI` называется с нарушением правил PEP 8. Как мы говорили ранее, такое отступление возможно для сохранения совместимости с используемой библиотекой, а так как библиотека QT

изначально написана на C++ с именованием методов в стиле camelCase, то `initUI` — допустимое название метода в классе, который унаследован от `QWidget`.

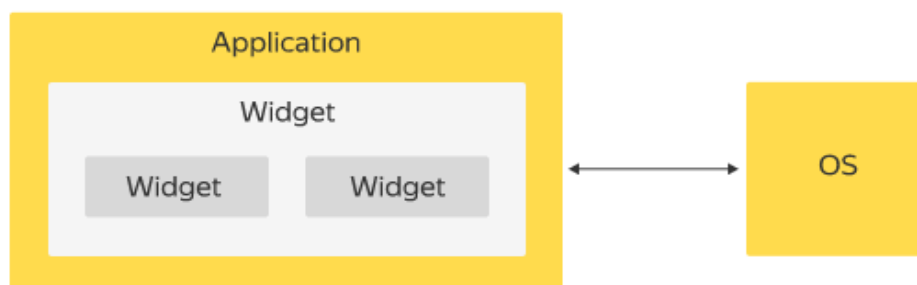
В `initUI` определяем положение и размеры окна методом `setGeometry`. Первые два параметра - X и Y - координаты левого верхнего угла формы относительно левого верхнего угла нашего монитора. Оставшиеся - ширина и высота виджета. Методом `setWindowTitle` задаем заголовок окна.

Мы создали класс, но пока его не используем. Чтобы начать с ним работать, необходимо куда-то разместить наш виджет.

Для этого нужно создать приложение - объект класса `QApplication`, строка `app = QApplication(sys.argv)` как раз отвечает за это. Несмотря на то, что после инициализации переменная `app` используется только один раз, она необходима. Все скрытое от пользователя и разработчика взаимодействие программы с операционной системой возможно только благодаря этому классу. Передавать значение `sys.argv` в конструктор `QApplication` не обязательно, можно передать любой список, даже пустой. Но приведенная запись является хорошим тоном создания PyQt-приложений, так как она позволит корректно обрабатывать запуск приложения с параметрами командной строки (о них мы поговорим позднее).

Затем создаем экземпляр класса: `ex = Example()`. Все готово, можно запускать. Метод `show()` отображает наш виджет в приложение. Но отобразить недостаточно, надо запустить цикл обработки событий. Для этого вызываем метод `app.exec()`.

В последней строке программы этот вызов «обернут» в `sys.exit`. Это сделано для корректного завершения программы.



Начнем добавлять в окно виджеты. Первый на очереди — кнопка. Класс, который необходим для работы с ней, называется `QPushButton`.

```

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 300, 300)
        self.setWindowTitle('Вторая программа')
        # Создаем кнопку.
        # Передаем 2 параметра:
        # надпись и виджет, на котором будет размещена кнопка
        btn = QPushButton('Кнопка', self)
        # Изменяем размер кнопки. Теперь он 100 на 100 пикселей
        btn.resize(100, 100)
        # Размещаем кнопку на родительском виджете
        # по координатам (100, 100)
        btn.move(100, 100)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    sys.exit(app.exec())

```

У любого виджета, кроме базового, должен быть «родитель». Когда мы добавляем кнопку, «родителем» выступает наш виджет окна. Поэтому при объявлении кнопки мы указываем не только текст, но и экземпляр класса QWidget (или, как в нашем случае, его наследника).

Метод `resize` позволяет изменить размеры кнопки. А с помощью метода `move` указываем расположение нашей кнопки в виджете-«родителе». Запустим программу и убедимся, что кнопка появилась.

На кнопку можно даже понажимать, но пока безрезультатно. Сделаем ее полезной - добавим функциональность. Т.е. добавим обработчик события «нажатие на кнопку».

```

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 300, 300)
        self.setWindowTitle('Третья программа')

        self.btn = QPushButton('Кнопка', self)
        self.btn.resize(100, 100)
        self.btn.move(100, 100)
        # присоединим к событию нажатия на кнопку обработчик self.hello()
        self.btn.clicked.connect(self.hello)

    def hello(self):
        # метод setText() используется для задания надписи на кнопке
        self.btn.setText('Привет')

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    sys.exit(app.exec())

```

Что изменилось по сравнению с предыдущей программой? Во-первых, теперь `btn` - это поле класса, а не просто локальная переменная метода, как было в прошлый раз. Поэтому добраться до кнопки мы теперь можем не только из метода `initUI`, а из любого места внутри класса формы (и даже из любого кода, который имеет доступ к объекту нашей формы). А во-вторых, добавилась функциональность: при нажатии на кнопку надпись на ней изменяется на строку «Привет».

`self.btn.clicked.connect(self.hello)` - что значит эта фраза в переводе на человеческий язык? «Если получишь событие `clicked` от объекта `self.btn`, вызови обработчик `self.hello()`». В нем с помощью метода `setText` мы меняем текст на кнопке.

Но эта кнопка фактически «одноразовая», поскольку текст на кнопке изменится только после первого нажатия. При последующих нажатиях он, конечно же, тоже меняется, ведь метод `hello()` вызывается, но мы, как пользователи, этого не видим.

Если говорить про библиотеку QT, в ней концепция «Событие → Обработчик → Действие» реализована с некоторыми особенностями, поэтому введена своя терминология - сигналы и слоты.

Сигнал вырабатывается, когда происходит определенное событие. Слот - это функция, которая ловит определенный сигнал. Все классы, наследуемые от `QObject` или его дочерних классов (например, уже знакомые нам `QWidget` и `QPushButton`) могут содержать сигналы и слоты. Сигналы вырабатываются объектами, когда они изменяют свое состояние так, что это может заинтересовать другие объекты. При этом сами объекты не знают и не заботятся о том, что у его сигнала может не быть получателя. Только класс, который определяет сигнал, (или его дочерние классы) могут вырабатывать сигнал.

Слоты - обычные функции (или методы класса), которые могут быть использованы для получения сигналов (хотя никто не запрещает вам вызывать эти функции и просто так). Точно так же, как объект не знает ничего о получателях своих сигналов, функция-слот ничего не знает о сигналах, которые к ней подключены.

Мы можем подключать к одному слоту сколько угодно сигналов, а один слот может быть подключен к неограниченному количеству сигналов (даже несколько раз к одному и тому же сигналу, тогда при появлении сигнала функция-слот выполнится столько раз, сколько была подключена). Кроме того, возможно подключать сигнал к другому сигналу, что вызовет выработку второго сигнала немедленно после появления первого.

В качестве аналогии можно привести радиотрансляцию с радиостанции. Сигнал - это такая трансляция, и радиостанции неизвестно, сколько радиоприемников-слотов будут настроены на волну, когда будет трансляция. Но все настроенные на нужную волну приемники получают сигнал.

Выведем на кнопке не один и тот же текст каждый раз, а, например, количество нажатий на нее. Никаких дополнительных атрибутов в классе нам не понадобится - хранить информацию будем прямо в «кнопке».

```

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 300, 300)
        self.setWindowTitle('Четвёртая программа')

        self.btn = QPushButton('0', self)
        self.btn.resize(100, 100)
        self.btn.move(100, 100)
        # Подпишем функцию-слот self.count() на сигнал clicked кнопки btn
        self.btn.clicked.connect(self.count)

    def count(self):
        # Не забываем, что надпись на кнопке - это текст.
        self.btn.setText(f"{int(self.btn.text()) + 1}")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    sys.exit(app.exec())

```

Метод `text()` возвращает строку — текущую надпись на кнопке.

Для отображения данных в PyQT есть и более подходящие виджеты. Для текстовых данных лучше использовать `QLabel`, а для цифр есть красивый виджет `QLCDNumber`, который имитирует дисплей калькулятора.

```

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton
from PyQt5.QtWidgets import QLCDNumber, QLabel

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 400, 400)
        self.setWindowTitle('Пятая программа')

        self.btn = QPushButton('Кнопка', self)
        # Подстроим размер кнопки под надпись на ней
        self.btn.resize(self.btn.sizeHint())
        self.btn.move(100, 150)
        # Обратите внимание: функцию не надо вызывать :)
        self.btn.clicked.connect(self.inc_click)

        self.label = QLabel(self)
        # Текст задается также, как и для кнопки
        self.label.setText("Количество нажатий на кнопку")
        self.label.move(80, 30)

        self.LCD_count = QLCDNumber(self)
        self.LCD_count.move(110, 80)

        self.count = 0

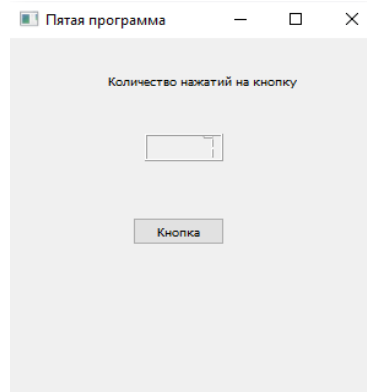
```

```
def inc_click(self):
    self.count += 1
    # В QLCDNumber для отображения данных используется метод display()
    self.LCD_count.display(self.count)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    sys.exit(app.exec())
```

Теперь для хранения будем использовать отдельное поле - `count`. Для задания значения у `QLCDNumber` используется метод `display`, а для `QLabel` - `setText`, как и у `QPushButton`.

Код `self.btn.resize(self.btn.sizeHint())` подстраивает размеры кнопки под размер надписи на ней, чтобы не возникла ситуация, что часть надписи заходила за границу этого виджета.



`QLCDNumber`, кроме чисел, умеет показывать следующие символы: 0 (нулем), S (с помощью 5), g (с помощью 9), минус, точку, A, B, C, D, E, F, h, H, L, o, P, r, u, U, Y, кавычку, пробел. То есть вполне возможно написать на `QLCDNumber` какое-либо сообщение, например `Error`. Те символы, которые `QLCDNumber` отобразить не может, он просто заменяет на пробелы.

Отображать данные пользователю нашей программы - это только половина дела, не менее важно получать от него данные. Для этого существуют несколько виджетов. Если пользователь должен ввести одну строку, то для получения ее получения прекрасно подойдет `QLineEdit`.



```

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton
from PyQt5.QtWidgets import QLabel, QLineEdit

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 400, 400)
        self.setWindowTitle('Шестая программа')

        self.btn = QPushButton('Кнопка', self)
        self.btn.resize(self.btn.sizeHint())
        self.btn.move(100, 150)
        self.btn.clicked.connect(self.hello)

        self.label = QLabel(self)
        self.label.setText("Привет, неопознанный лев")
        self.label.move(40, 30)

        self.name_label = QLabel(self)
        self.name_label.setText("Введите имя: ")
        self.name_label.move(40, 90)

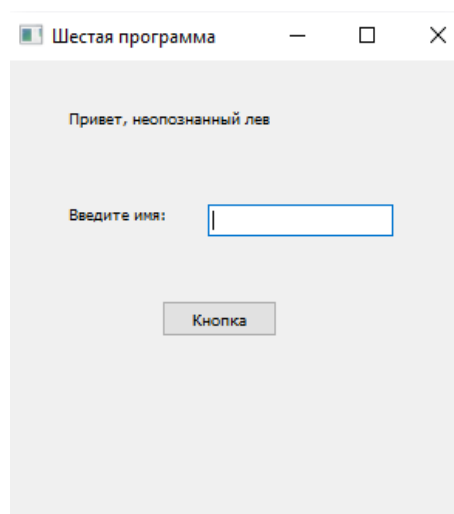
        self.name_input = QLineEdit(self)
        self.name_input.move(150, 90)

    def hello(self):
        name = self.name_input.text() # Получим текст из поля ввода
        self.label.setText(f"Привет, {name}")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    sys.exit(app.exec())

```

Метод `text()` виджета `QLineEdit` позволяет получить введенную пользователем строку.



Рассмотрим более детально, как происходит взаимодействие между виджетами. Когда пользователь нажимает на кнопку, возникает сигнал, который обрабатывается некоторой функцией-слотом. Но если у нас одна функция-обработчик для нескольких кнопок, как понять, на какую из них нажал пользователь?

Чтобы определить, кто является источником сигнала, у виджета есть метод `.sender()`.

Рассмотрим пример такой программы.

```
import sys

from PyQt5.QtWidgets import QWidget, QApplication, QPushButton, QLabel

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('Кто отправил сигнал')

        self.button_1 = QPushButton(self)
        self.button_1.move(90, 40)
        self.button_1.setText("Кнопка 1")
        self.button_1.clicked.connect(self.run)

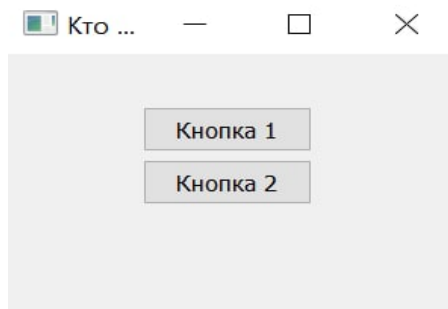
        self.button_2 = QPushButton(self)
        self.button_2.move(90, 80)
        self.button_2.setText("Кнопка 2")
        self.button_2.clicked.connect(self.run)

        self.label = QLabel(self)
        self.label.setText("Пока никто не отправлял")
        self.label.move(50, 120)

        self.show()

    def run(self):
        self.label.setText(self.sender().text())
        print(self.sender().text())

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    sys.exit(app.exec())
```



В этой программе слот `run()` привязан к сигналам от двух кнопок. Независимо от того, кто его вызвал, метод печатает в консоль текст нажатой

кнопки, которую получаем с помощью `sender()` .

Разумеется, далеко не всегда приложение ограничивается одной формой. В настоящих программах, как минимум, есть еще одна - «О программе», а также формы с настройками, диалоги открытия и сохранения файлов и так далее. Конечно, PyQT дает возможность в приложении создавать формы из других форм. Обычно (но далеко не всегда) для главной формы приложения выбирают класс `QMainWindow`, а для дочерних форм - класс `QWidget`. Чтобы создать форму из другой формы, достаточно сделать объект нужного нам класса и вызвать у него метод `show`.

```
import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton
from PyQt5.QtWidgets import QMainWindow, QLabel

class FirstForm(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 300, 300)
        self.setWindowTitle('Главная форма')

        self.btn = QPushButton('Другая форма', self)
        self.btn.resize(self.btn.sizeHint())
        self.btn.move(100, 100)

        self.btn.clicked.connect(self.open_second_form)

    def open_second_form(self):
        self.second_form = SecondForm(self, "Данные для второй формы")
        self.second_form.show()

class SecondForm(QWidget):
    def __init__(self, *args):
        super().__init__()
        self.initUI(args)

    def initUI(self, args):
        self.setGeometry(300, 300, 300, 300)
        self.setWindowTitle('Вторая форма')
        self.lbl = QLabel(args[-1], self)
        self.lbl.adjustSize()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = FirstForm()
    ex.show()
    sys.exit(app.exec())
```

Обратите внимание на два момента:

1. Мы сохраняем созданную форму в атрибут родительской формы для того, чтобы иметь возможность управлять ею из других методов. И для того, чтобы сборщик мусора Python не удалил ее случайно как объект,

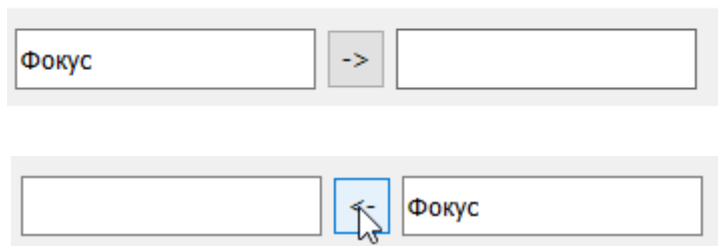
на который нет ссылок.

2. В инициализаторе дочерней формы, помимо `self`, есть еще `*args`, куда мы можем помещать информацию, которую хотим передать из родительской формы в дочернюю. Тут мы передаем ссылку на объект-родитель (его обычно передают первым) и сообщение, которое мы будем отображать на второй форме в `QLabel`.

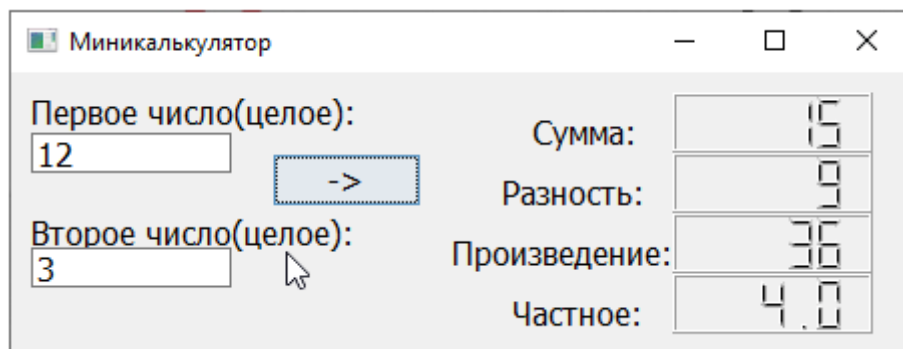
Информацию о других методах можно посмотреть на официальном сайте QT. Обратите внимание: документация написана для языка C++, так что нужно обращать внимание лишь на названия методов и параметры, но не на синтаксис примеров. <https://doc.qt.io/qt-5/qtwidgets-module.html>

### Задания для самостоятельного выполнения

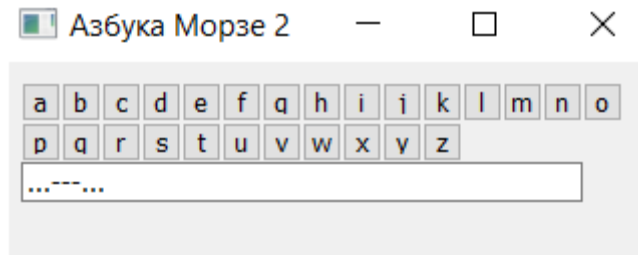
**Задание 1.** На форме разместите два поля для ввода и кнопку. На кнопке должна быть показана стрелка от первого поля ко второму. В первое поле вводится строка, по нажатию кнопки эта строка перебрасывается в другое поле, при этом на кнопке меняется стрелка на противоположную. При повторном нажатии строка летит обратно, а стрелка на кнопке меняется на изначальную. И так далее.



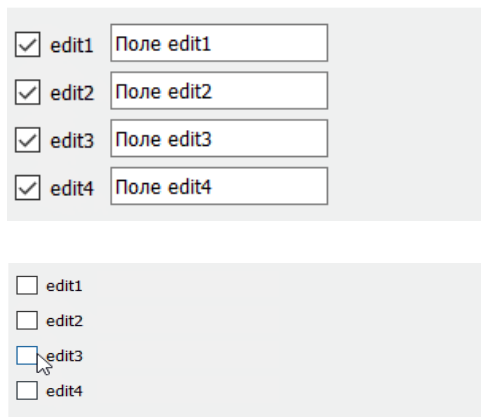
**Задание 2.** Напишите программу с графическим пользовательским интерфейсом на PyQt, в которой в два текстовых поля вводятся целые числа. После нажатия кнопки «Рассчитать» программа должна вычислить сумму, разность, частное и произведение введенных чисел и вывести результат каждой операции в отдельные виджеты `QLCDNumber`. В случае попытки деления на 0 программа должна выводить какое-либо сообщение.



**Задание 3.** Разместите на форме кнопки с латинскими буквами, а так же поле для ввода. При нажатии на кнопку добавляйте в поле код этой буквы из азбуки Морзе. Не создавайте каждую кнопку вручную, воспользуйтесь циклом.

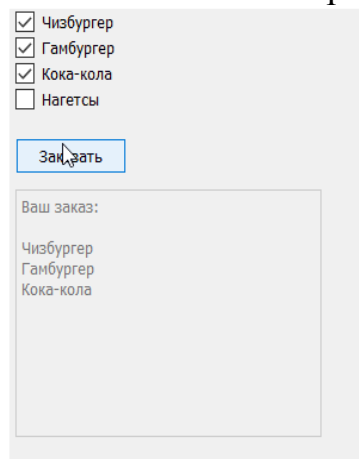


**Задание 4.** Разместите на форме три или более произвольных виджета, а рядом с ними чекбоксы. Напишите обработчики, которые прячут и показывают виджеты в зависимости от состояния чекбокса. Попробуйте не писать по обработчику для каждого чекбокса, а сделать один универсальный.



**Задание 5.** Напишите программу с графическим пользовательским интерфейсом на PyQt.

Пользователь должен иметь возможность выбирать одно или несколько блюд. После нажатия на кнопку «Заказать» в отдельном виджете должен отображаться «чек» с выбранными позициями.



Добавьте возможность указать не только блюдо, но и количество порций. У каждого блюда есть цена. По умолчанию, если блюдо выбрано,

количество становится равным 1. В чеке должна быть отображена следующая информация: блюдо, количество, итоговая стоимость блюда каждого типа и суммарная стоимость заказа.

<input checked="" type="checkbox"/>	Чизбургер	1
<input checked="" type="checkbox"/>	Гамбургер	12
<input type="checkbox"/>	Кока-кола	0
<input type="checkbox"/>	Нагетсы	0

Заказать

Ваш заказ

Чизбургер-----1-----10  
Гамбургер-----12-----240  
  
Итого: 250