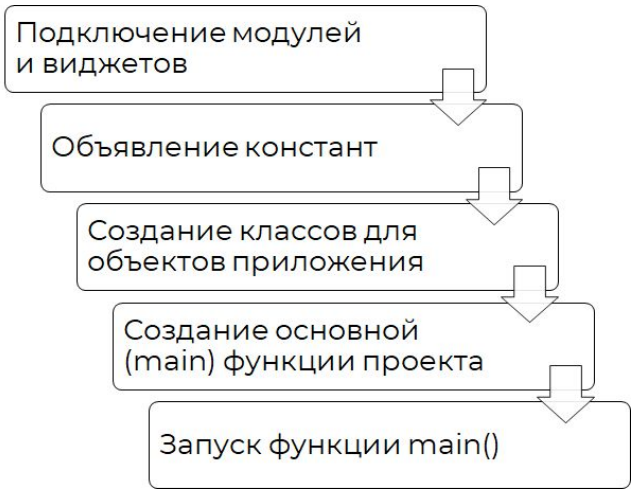
Теория

Применение ООП в разработке десктопного приложения

Верхнеуровневую схему разработки проекта можно представить следующим образом:



1. **Подключение модулей и виджетов**. Подключение необходимых библиотек, модулей, виджетов и пр. всегда пишется в самом верху программы.

```
#подключение модулей и виджетов
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget, QHBoxLayout,
QPushButton, QLabel, QLineEdit
```

2. **Объявление констант.** Константы — это данные, которые будет использовать программа. Они не подразумевают изменений. Например, размер будущего окна, текст на надписях и кнопках и прочее. Такие данные легче всего создать вначале программы в одном месте, а далее использовать их в виде переменных. Так всегда будет очевидно, где искать эти данные для внесения правок. Пример:

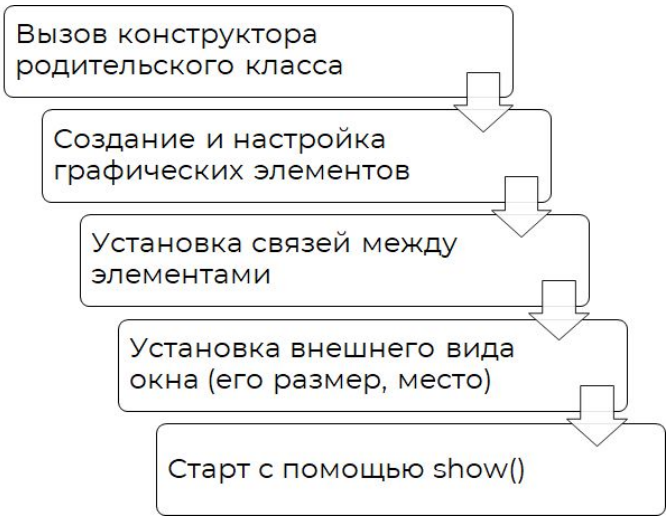
```
#подключение модулей и виджетов
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget, QHBoxLayout,
QPushButton, QLabel, QLineEdit

#объявление констант
win_width, win_height = 800, 300
win_x, win_y = 200, 200
txt_title = "О т п р а в к а  т е к с т а"
txt_send = "О т п р а в и т ь"
txt_line = "П о л е  в в о д а"
```

3. **Создание классов для объектов приложения.** Объекты зависят от поставленной задачи. Это может быть класс “Человек”, если нам нужно хранить данные о каких-то людях, или “Меню”, но также есть классы, которые точно будут реализованы в любом оконном приложении — классы окон. Поскольку окно содержит много виджетов, должно правильно отображать данные, удобно создавать отдельный класс для каждого окна. К тому же, библиотека PyQT5 предоставляет возможность наследования такого класса от встроенного QWidget, т. е. таким образом программа автоматически будет распознавать новый класс как виджет.

```
#подключение модулей и виджетов
…
#объявление констант
…
class MainWindow(QWidget):
...
```

При создании класса окна важно также грамотно продумать структуру самого класса. Делается это следующим образом: алгоритм создания окна делится на несколько шагов, каждый из которых описывается в отдельном методе класса. Эти методы вызываются последовательно в конструкторе класса.

Пример алгоритма:



Важно понимать, что, если вы наследуете свой класс окна от QWidget и создаёте в нём конструктор, вы переопределяете конструктор класса QWidget и при создании экземпляра нашего класса уже не будет создаваться виджет. Чтобы такого не произошло, необходимо вызывать конструктор с определёнными параметрами и обязательно внутри вручную вызывать конструктор родительского класса.

Пример программы:

```
#подключение модулей и виджетов
…
#объявление констант
…
class MainWindow(QWidget):
def __init__(self, parent=None, flags=Qt.WindowFlags()):
        #вызов конструктора родительского класса
        super().__init__(parent=parent, flags=flags)

        # создаём и настраиваем графические элементы:
        self.initUI()

        #устанавливает связи между элементами
        self.connects()

    #устанавливает, как будет выглядеть окно (надпись, размер, место)
```

```
        self.set_appear()

        # старт:
        self.show()

    def initUI(self):
        ''' создает графические элементы '''
        self.btn_send = QPushButton(txt_send, self)
        self.line = QLineEdit(txt_line)
        self.lable_finish = QLabel()

        self.layout_line = QHBoxLayout()
        self.layout_line.addWidget(self.line, alignment = Qt.AlignLeft)
        self.layout_line.addWidget(self.btn_send, alignment =
Qt.AlignLeft)
        self.layout_line.addWidget(self.lable_finish, alignment =
Qt.AlignCenter)
        self.setLayout(self.layout_line)

    def next_click(self):
        self.lable_finish.setText(self.line.text())

    def connects(self):
        self.btn_send.clicked.connect(self.next_click)

''' устанавливает, как будет выглядеть окно (надпись, размер, место) '''
    def set_appear(self):
        self.setWindowTitle(txt_title)
        self.resize(win_width, win_height)
        self.move(win_x, win_y)
```

4. **Создание основной (main) функции и запуск проекта.** После реализации класса основного окна, остаётся лишь создать объект приложения и объект разработанного нами класса окна, чтобы программа начала свою работу. Это принято делать в функции main() т. е. основной (стартовой) функции проекта.

```
#подключение модулей и виджетов
```

```
...
#объявление констант
...
#создание классов для объектов приложения
class MainWindow(QWidget):
def __init__(self, parent=None, flags=Qt.WindowFlags()):
        #вызов конструктора родительского класса
        super().__init__(parent=parent, flags=flags)

     # создаём и настраиваем графические элементы:
        self.initUI()

        #устанавливает связи между элементами
        self.connects()

        #устанавливает, как будет выглядеть окно (надпись, размер, место)
        self.set_appear()

        # старт:
        self.show()
    ...
#создание основной (main) функции проекта
def main():
    app = QApplication([])
    mw = MainWindow()
    app.exec_()
#запуск проекта
main()
```