Курс программирования Python Pro
Модуль 5. Десктопные приложения.
Знакомство с PyQT5.

Теория

Содержание:

Общие понятия

Оконное приложение — это программа, использующая элементы графического интерфейса для взаимодействия с пользователем (например, кнопки, окна, переключатели). При помощи устройств ввода (клавиатура/мышь/тачпад и прочие), пользователь имеет возможность взаимодействовать с объектами оконного приложения: перемещать, активировать, прокручивать.

Виджет — это специальный элемент пользовательского интерфейса, который отображает необходимую информацию или даёт возможность взаимодействовать с операционной системой или приложением.

Примеры виджетов: надпись, кнопка, радиокнопка.

PyQT5 — это кроссплатформенная библиотека для создания оконных приложений. Кроссплатформенность библиотеки означает, что созданное с помощью PyQt5 приложение будет одинаково хорошо открываться на любой операционной системе.

Подключение модулей

В библиотеке PyQt5 есть много готовых модулей и функций. Чтобы подключить модуль для какого-либо виджета, нужно воспользоваться знакомой вам командой:

```
from PyQt5.QtWidgets import виджет1, виджет2
```

Некоторые варианты виджетов, которые можно подключить:

Виджет	**Обозначение**
Приложение	QApplication
Окно приложения	QWidget
Надпись	QLabel
Кнопка	QPushButton
Радиокнопка	QRadioButton
Группа кнопок*	QButtonGroup

* Кнопки могут быть разных видов.

Создание приложения

В начале разработки необходимо создать само приложение. Для этого есть специальный метод QApplication([]), который возвращает объект типа “приложение”:

```
app = QApplication([])
```

Сразу после создания приложения важно написать в конце программы команду app.exec_(). Это служебная команда, которая оставит приложение открытым до тех пор, пока пользователь не нажмёт на кнопку выхода («красный крестик»).

Окно приложения

Для того, чтобы создать окно приложения, необходимо использовать команду *QWidget()*, которая возвращает объект типа “окно”:

```
my_win = QWidget()
```

У окна есть ряд параметров, которые можно изменить:

Метод	Назначение
`my_win.setWindowTitle('Название')`	Установить заголовок окна
`my_win.move(900, 70)`	Появление окна в указанной точке (а не по центру экрана)
`my_win.resize(400, 200)`	Изменение размеров окна
`my_win.show()`	**Сделать окно видимым**
`my_win.hide()`	Скрыть окно

Создание виджетов

Чтобы добавить виджет на окно, нужно последовательно выполнить несколько действий. В первую очередь необходимо создать соответствующий объект виджета:

Виджет	Метод
Надпись	`title = QLabel('Я родился!')`
Кнопка	`button = QPushButton('Подтвердить')`
Радиокнопка	`radiobutton = QRadioButton('Вариант 1')`
Группа кнопок	`buttongroup = QButtonGroup()`
Текстовое поле	`line = QLineEdit('Подсказка для пользователя')`

После создания соответствующего объекта переходим к его размещению на окне (смотреть далее).

Размещение виджетов на окне

Созданные элементы приложения нужно расположить в окне. Это удобно сделать с помощью направляющих линий. Для этого необходимо подключить соответствующие модули:

```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import другие виджеты, QHBoxLayout, QVBoxLayout
```

где QHBoxLayout — горизонтальная направляющая, QVBoxLayout — вертикальная направляющая.

Например, чтобы расположить элементы приложения по вертикали, нужно создать направляющую вертикальную линию, добавить к ней виджеты (по желанию, можно выровнять их по центру, по левому краю и т. д.) и добавить линию к окну приложения:

Метод	*Назначение*

`v_line = QVBoxLayout()`	**Создание вертикальной линии**, по которой можно будет выровнять элементы приложения
`h_line = QHBoxLayout()`	**Создание горизонтальной линии** (аналогично)

Добавление объекта к линии:

Метод	*Назначение*
`v_line.addWidget(title, alignment = Qt.AlignCenter)`	Добавить надпись к вертикальной линии; расположить надпись по центру

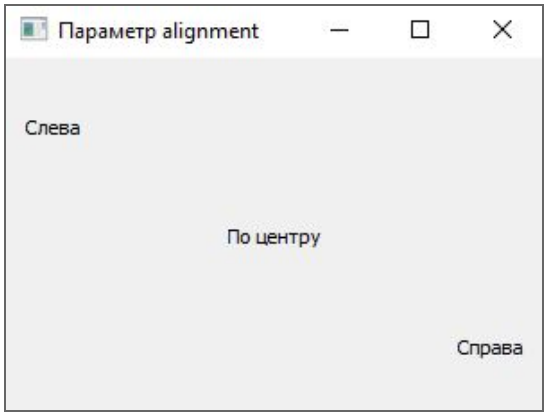
Параметр *alignment* в методе *addWidget* задаёт выравнивание компонента по направляющей.

В этом параметре можно указать следующие атрибуты:

- ★ **AlignLeft** — горизонтальное выравнивание по левому краю;
- ★ **AlignRight** — горизонтальное выравнивание по правому краю;
- ★ **AlignCenter** — горизонтальное выравнивание по центру;
- ★ **AlignBottom** — выравнивание по нижнему краю;
- ★ **AlignTop** — выравнивание по верхнему краю.

Пример:

```
l = QHBoxLayout()
l.addWidget(label1, alignment = Qt.AlignLeft)
l.addWidget(label2, alignment = Qt.AlignRight)
l.addWidget(label3, alignment = Qt.AlignCenter)
```

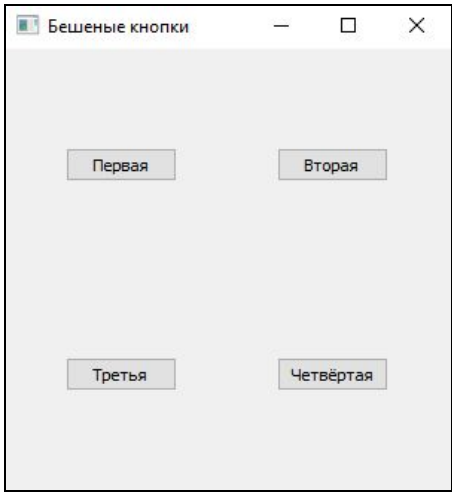


Разместить в окне направляющую со всеми привязанными к ней виджетами:

Метод	*Назначение*
`my_win.setLayout(v_line)`	Добавить получившуюся линию и её объекты в окно приложения

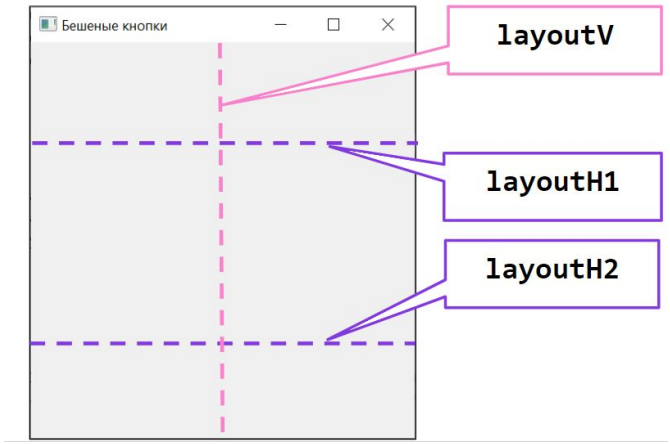
Для создания окон со сложным дизайном, можно использовать несколько направляющих, которые в процессе создания окна привязываются друг к другу. Для этого нужно создать базовые направляющие, разместить на них объекты, а после этого привязать базовые направляющие вместе с размещёнными на них объектами к главной направляющей. Главная направляющая, в свою очередь, размещается в окне.

Рассмотрим пример, в котором необходимо расположить объекты как на картинке:



Опускаем часть с подключением модулей, созданием приложения и окна и сразу переходим к созданию объектов:

```
#Создаём объекты кнопок
button1 = QPushButton('Первая')
button2 = QPushButton('Вторая')
button3 = QPushButton('Третья')
button4 = QPushButton('Четвёртая')

#Создаём 1 вертикальную и 2 горизонтальные направляющие
layoutV = QVBoxLayout()
layoutH1 = QHBoxLayout()
layoutH2 = QHBoxLayout()
```



Расположим кнопки по горизонтальным направляющим:

```
#Добавить объект button_ к направляющей линии по центру
layoutH1.addWidget(button1, alignment = Qt.AlignCenter)
layoutH1.addWidget(button2, alignment = Qt.AlignCenter)
layoutH2.addWidget(button3, alignment = Qt.AlignCenter)
layoutH2.addWidget(button4, alignment = Qt.AlignCenter)
```

Кнопки button1 и button2 расположены по горизонтальной направляющей layoutH1. Кнопки button3 и button4 расположены по горизонтальной направляющей layoutH2:

Далее необходимо привязать горизонтальные направляющие к вертикальной:

```
layoutV.addLayout(layoutH1)
layoutV.addLayout(layoutH2)
window.setLayout(layoutV)
window.show()
```

Создание радиокнопок

Радиокнопка (от англ. radio button) или переключатель — элемент интерфейса, который позволяет пользователю выбрать одну опцию (пункт) из предопределённого набора (группы).

Для работы с радиокнопками необходимо подключить соответствующий модуль (см. Создание виджетов) и создать объекты радиокнопок. Важно

отметить, что объект радиокнопки — это одна кнопка. Соответственно, если нужно создать, например, три варианта ответа в тесте, необходимо создать три объекта радиокнопок:

```
# создаём объекты радиокнопок
radio_button_1 = QRadioButton('1')
radio_button_2 = QRadioButton('2')
radio_button_3 = QRadioButton('3')
```

При запуске программы часто необходимо, чтобы какая-то кнопка уже была выбрана. Для этого используется метод setChecked, который установит указанной радиокнопке состояние “выбрана”:

```
# создаём объекты радиокнопок
radio_button_1 = QRadioButton('1')
radio_button_2 = QRadioButton('2')
radio_button_3 = QRadioButton('3')
# устанавливаем, какая радиокнопка будет выбрана при запуске программы
radio_button_1.setChecked(True)
```

Для того, чтобы объединить объекты радиокнопок и реализовать выбор только одного варианта ответа из предложенных, радиокнопки необходимо объединить в группу кнопок. Для этого необходимо подключить модуль QButtonGroup, создать объект “Группа кнопок” и добавить в группу нужные радиокнопки. Обращаем внимание на поле id: в нём можно указать уникальный номер (идентификатор) каждой кнопки, чтобы в дальнейшем было легко обратить к каждой радиокнопке в любом месте программы:

```
# создаём группу радиокнопок и добавляем туда созданные нами ранее объекты радиокнопок
button_group = QButtonGroup()
button_group.addButton(radio_button_1, id = 1)
button_group.addButton(radio_button_2, id = 2)
button_group.addButton(radio_button_3, id = 3)
```

Располагаем радиокнопки на необходимых направляющих, для того, чтобы увидеть результат на экране.

Для того, чтобы узнать, какая радиокнопка выбрана в данный момент, используется следующий метод, возвращающий уникальный идентификатор (номер, указанный в поле id) соответствующей радиокнопки:

```
название_группы_кнопок.checkedId()
```

Изменение и получение текста для виджета

Для того, чтобы изменить текст виджета уже в ходе выполнения программы, используется следующий метод:

```
название_объекта_виджета.setText("Новый текст")
```

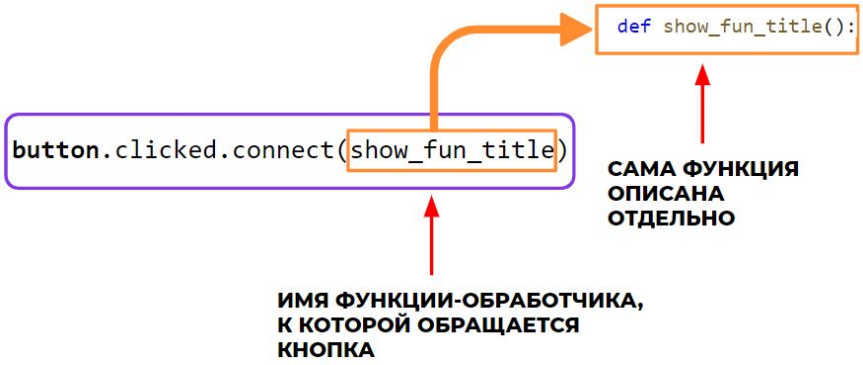
Для того, чтобы получить текст виджета, например, то, что ввёл пользователь в текстовое поле, используется следующий метод:

```
s = название_объекта_виджета.text()
```

Обработка события нажатия на кнопку

После создания кнопки, нужно сделать её активной, иначе, если запустить приложение и нажать на кнопку, то ничего не произойдёт. Реагирование элемента управления (кнопки) на событие внешнего мира (клик мышкой пользователя) называется обработкой события.

Часто действия программы после нажатия на кнопку описываются в отдельной функции. Представим, что действия программы описаны в функции show_fun_title(). Будем называть её функцией-обработчиком. Чтобы при нажатии на кнопку запустить функцию show_fun_title(), воспользуемся командой button.clicked.connect(show_fun_title). То есть “при нажатии на кнопку button запусти функцию show_fun_title”.



Работа со шрифтами

Для того, чтобы работать со шрифтами, необходимо подключить к проекту модуль QFont:

```
from PyQt5.QtGui import QFont
```

Изменение размера, толщины и шрифта текста виджета реализовывается следующим образом:

```
название_объекта_виджета.setFont(QFont("Times", 36, QFont.Bold))
```

Изменение цвета текста виджета реализовывается следующим образом, где (0,0,0) — код цвета в кодировке RGB:

```
название_объекта_виджета.setStyleSheet("color: rgb(0,0,0)")
```

Установка ограничений на ввод текста в текстовое поле

Для того, чтобы работать с текстовыми полями, необходимо подключить к проекту модуль QLineEdit:

```
from PyQt5.QtWidgets import QApplication, QWidget,..., QLineEdit

line = QLineEdit("Подсказка")
```

Qt содержит класс QValidator, проверяющий корректность введённых данных. Данных класс не может быть использован напрямую. Для проверки данных придётся воспользоваться готовыми подклассами QIntValidator (целые числа), QDoubleValidator (дробные числа). Также будет полезен модуль QLocale, который позволит привести данные к нужному языку. Тогда ограничить ввод можно следующим образом:

```
from PyQt5.QtWidgets import QApplication, QWidget,..., QLineEdit
from PyQt5.QtGui import QDoubleValidator, QIntValidator
from PyQt5.QtCore import QLocale

# установка языка, страны
loc = QLocale(QLocale.English, QLocale.UnitedStates)
validator = QDoubleValidator()
validator.setLocale(self.loc)

line = QLineEdit("Подсказка")
# устанавливаем ограничение - тест только числа
line.setValidator(validator)
# указываем диапазон допустимых значений
line.setValidator(QIntValidator(0, 150))
```

Таймер

Для того, чтобы работать с таймером и временем необходимо подключить соответствующие модули:

```
from PyQt5.QtCore import QTimer, QTime
```

Далее создать объект класса QTime, с помощью которого можно установить стартовое время. Первое число в параметрах — часы, второе — минуты, третье — секунды:

```
from PyQt5.QtCore import QTimer, QTime
time = QTime(0, 0, 15)
```

Следующий шаг — создать объект класса таймер. Этот инструмент будет запускать указанную функцию через указанные промежутки времени:

```
from PyQt5.QtCore import QTimer, QTime
time = QTime(0, 0, 15)

# создаём объект типа таймер
timer = QTimer()
# связываем его с функцией
timer.timeout.connect(timerEvent)
# устанавливаем промежутки запуска функции
timer.start(1000)
```

В связанной с таймером функции можно указать, что делать таймеру при каждом её вызове. Это обязательно должно быть изменение времени, а также важно указать условие остановки таймера:

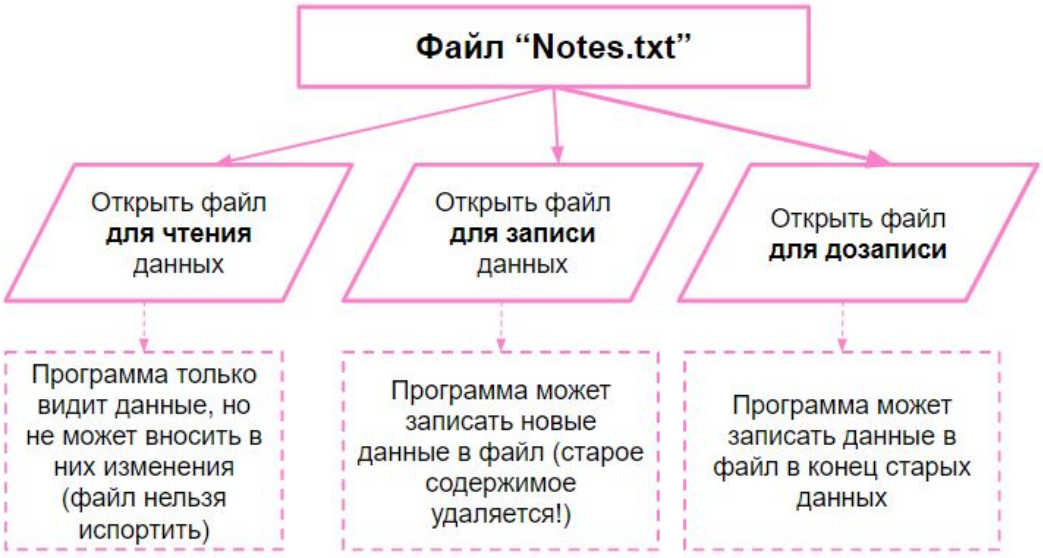
```
from PyQt5.QtCore import QTimer, QTime

def timerEvent(self):
    # отнимаем секунду от времени при каждом вызове функции
    time = time.addSecs(-1)
    # условие остановки - когда время станет равно 00:00:00
    if time.toString("hh:mm:ss") == "00:00:00":
       self.timer.stop()

time = QTime(0, 0, 15)
# создаём объект типа таймер
timer = QTimer()
# связываем его с функцией
timer.timeout.connect(timerEvent)
# устанавливаем промежутки запуска функции - 1 секунда
timer.start(1000)
```

Ввод и вывод данных (формат .txt)

Ввод и вывод данных можно организовать с помощью текстовых файлов. Текстовый файл можно создать с помощью текстового редактора «Блокнот» или среды разработки VS Code. Файл должен находиться в одной папке с программой. Сама программа может использовать файл по-разному: как источник информации, как хранилище данных или и то, и другое.

Взаимодействие программы и текстового файла можно отразить в виде следующей схемы:

К файлу может быть настроен различный вид доступа:



Используя разные атрибуты доступа, можно открывать файлы для

определённых целей:

Назначение функции	*Функция в Python*
Открыть файл для чтения	`file = open(“notes.txt”, “r”)`
Открыть файл для записи (старое содержимое удаляется!)	`file = open(“notes.txt”, “w”)`
Открыть файл на дозапись (информация добавляется в конец файла)	`file = open(“notes.txt”, “a”)`

Для того, чтобы прочитать данные из файла, понадобятся следующие функции:

Назначение функции	*Функция в Python*
Открытие файла на чтение	`file = open(“notes.txt”, “r”)`
Чтение данных файла	`data = file.read()`
Чтение данных файла (по частям)	`data = file.read(1024)`
Закрыть файл по окончании работы	`file.close()`

Пример. В файле poem.txt записано стихотворение:

files_program.py

```
file = open("poem.txt", "r")
data = file.read(1)
print(data)
data = file.read()
print(data)
file.close()
```

poem.txt

Унылая пора! Очей очарованье!
Приятна мне твоя прощальная краса —
Люблю я пышное природы увяданье,
В багрец и в золото одетые леса,
В их сенях ветра шум и свежее дыханье,
И мглой волнистою покрыты небеса,
И редкий солнца луч, и первые морозы,
И отдаленные седой зимы угрозы.

После запуска кода слева содержимое файла будет выведено на экран в следующем виде:

Результат работы

```
у
нылая пора! Очей очарованье!
Приятна мне твоя прощальная краса –
Люблю я пышное природы увяданье,
В багрец и в золото одетые леса,
В их сенях ветра шум и свежее дыханье,
И мглой волнистою покрыты небеса,
И редкий солнца луч, и первые морозы,
И отдаленные седой зимы угрозы.
bash-3.2$
```

Для записи информации в файл потребуются следующие функции:

Назначение функции	*Функция в Python*
Открытие файла на запись	`file = open(“notes.txt”, “w”)`
Запись данных в файл	`file.write(“Information”)`
Закрытие файла	`file.close()`

Пример:

files_program.py

```
#допишем автора стихотворения
file = open("poem.txt", "w")
file.write("\nA. C. Пушкин")
file.close()
file = open("poem.txt", "r")
data = file.read()
print(data)
```

poem.txt

Унылая пора! Очей очарованье!

Приятна мне твоя прощальная краса –

Люблю я пышное природы увяданье,

В багрец и в золото одетые леса,

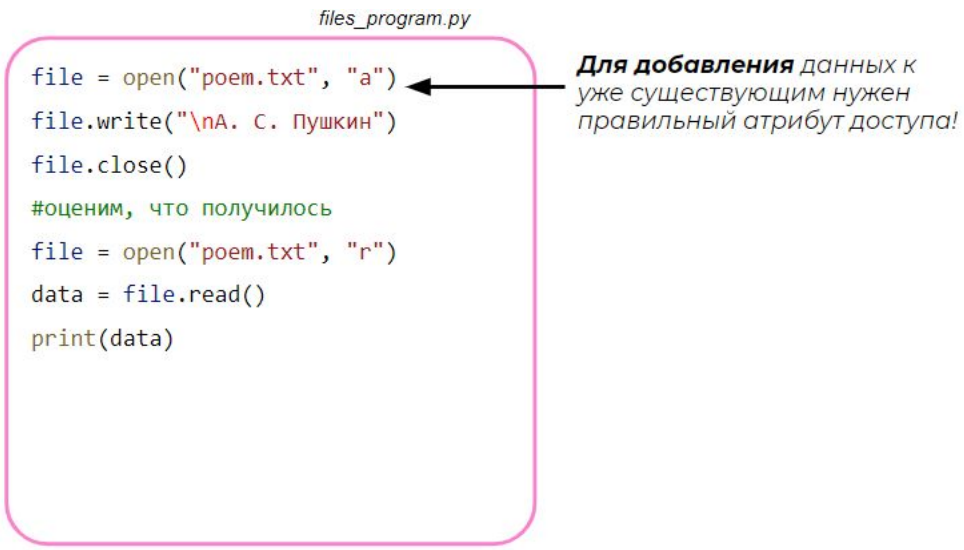
В их сенях ветра шум и свежее дыханье,

И мглой волнистою покрыты небеса,

И редкий солнца луч, и первые морозы,

И отдаленные седой зимы угрозы.

При выполнении данного кода содержимое файла poem.txt сотрётся и будет заменено на указанную строку. Если необходимо дополнить файл новой информацией, необходимо изменить тип доступа к файлу:

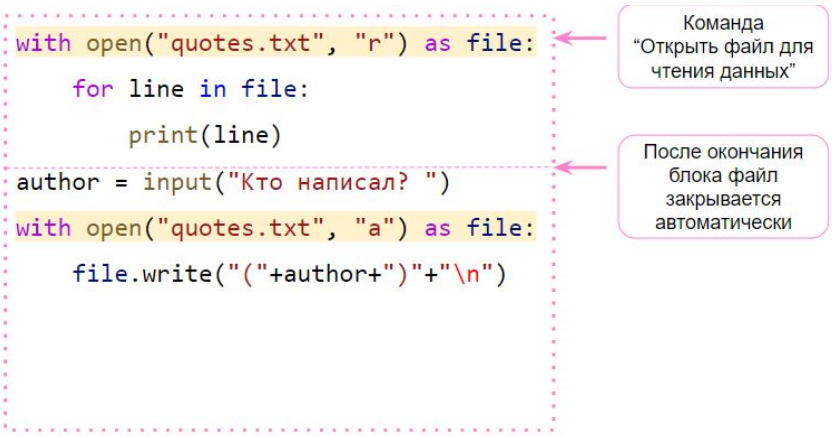


Код открытия / закрытия файла и чтения / записи информации занимает много места.

Пример:

```
file = open("quotes.txt", "r")
data = file.read()
file.close()
print(data)
author = input("Кто написал? ")
file = open("quotes.txt", "a")
file.write("("+author+")"+"\n")
file.close()
```

Существует альтернативная команда для **открытия и автоматического закрытия** файла. Так вышеуказанный код программы можно записать следующим образом:



Нужно обратить внимание на удобный способ построчного чтения файла с помощью цикла for.