

# **Тема 2.**

## **Событийно-ориентированное программирование.**

**Лекция 5. Взаимодействие элементов управления посредством  
сигналов/слотов**



# Обо мне

• ФОТО

- Харченко Владислав Алексеевич, 26 лет.
- 2012 – 2017 гг. ВГТУ. Факультет Радиотехники и Электроники.
- 2014 – 2018 гг. Фриланс.
- 2017 – 2018 гг. АО «НИИ СВТ». Программист.
- 2018 – 2021 гг. АО «НИИ СВТ». Старший программист.

# Учебные вопросы

1. [Понятие сигналов и слотов](#)
2. [Потоки QThread](#)
3. [Генерация сигналов](#)

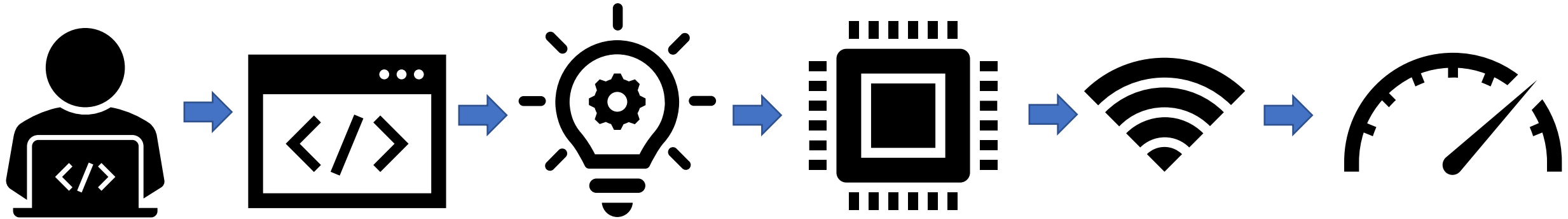
# Источники

- Официальная документация: <https://doc.qt.io/qtforpython>
- Прохоренок Н. А., Дронов В. А. Python 3 и PyQt 5. Разработка приложений. 2019 г.

Используемые в курсе инструменты для разработки		
<b>IDE</b>	PyCharm CE	<a href="https://www.jetbrains.com/pycharm/download">https://www.jetbrains.com/pycharm/download</a>
<b>Окружение</b>	Virtualenv	<a href="https://docs.python.org/3/library/venv.html">https://docs.python.org/3/library/venv.html</a>
<b>VSC</b> (рекомендовано)	GIT	<a href="https://git-scm.com">https://git-scm.com</a>
<b>Фреймворк</b>	PySide2	<a href="https://doc.qt.io/qtforpython/">https://doc.qt.io/qtforpython/</a>

# **1. Понятие сигналов и слотов**

# Понятие событийно-ориентированного программирования



# Назначение обработчиков

- Для генерации сигнала необходимо:
  1. Указать одноименное событие (clicked, triggered, textChanged) для необходимого виджета;
  2. Назначить сигналу обработчик (слот) с помощью метода connect()
- Пример:

```
self.button1.clicked.connect(lambda: print(f"button1 отправлен сигнал"))
self.button2.clicked.connect(self.pressed)
self.someComboBox.currentTextChanged.connect(
    lambda: print(f"Установлено значение {self.someComboBox.currentText()}"))
```

# Назначение обработчиков (продолжение)

- Обработчиком можно назначить:
  - Ссылку на функцию;
  - Метод класса;
  - Экземпляр класса, в котором определен метод `__call__()`;
  - Анонимную функцию;
  - Ссылку на слот класса.
- Данные в обработчик можно передать:
  - Через метод `__call__`;
  - Через анонимную функцию;
  - Через метод `partial()` из библиотеки `functools()`

Виды сигналов можно изучить для каждого элемента в [оф. документации](#)



# Система слотов

- Слот вызывается когда вырабатывается сигнал, с которым он связан.
- Слот это обычная функция в python и может вызываться обычным способом; единственная его особенность, что с ним можно соединять сигналы.
- \*Для того, чтобы функцию сделать слотом, необходимо указать для этой функции декоратор @Slot.

\*Примечание: данное действие не обязательно, но желательно, т.к. функция на которую ссылается сигнал, автоматически является слотом, однако при указании декоратора вызов слота будет выполняться быстрее чем метода

## **2. Потоки QThread**

# Понятие многопоточного программирования

- При запуске слота, основной цикл приложения блокируется. Если операция выполняется быстро, то пользователь не замечает блокирования GUI, однако, если операция подразумевает работу с данными или подсчёты, которые занимают время, приложение будет «зависшим» всё время выполнения расчётов.
- **Решение:**
  - Использование метода `processEvent()`
  - Использование потоков `QThread()`

# Использование processEvent()

- В значимые моменты выполнения кода, есть возможность вызвать стандартный метод **processEvent()**, во время выполнения этой конструкции основной поток снова перехватывает управление, давая возможность обновить GUI

```
@QtCore.Slot()
def myTimer(self):
    for _ in range(10, 0, -1):
        self.lineEdit.setText(str(_))
        time.sleep(1)
        QtWidgets.QApplication.processEvents()
```

Как отработает данный слот, если закомментировать указанную строку?

И какое поведение будет если запустить приложение с указанной строкой?

# Использование класса QThread

- Для долгих операций, целесообразно использовать класс QThread.

```
class TestThread(QtCore.QThread):  
  
    def run(self) -> None:  
        for _ in range(10, 0, -1):  
            time.sleep(1)  
            print(_)
```

```
self.t = TestThread()  
self.button.clicked.connect(self.t.start)
```

В данном случае блокировки GUI  
происходить не будет

Блокирует ли GIL  
потоки QThread  
написанные на  
C++?

# Передача данных в QThread

- Передача данных в метод `__init__()` при инициализации класса, унаследованного от `QThread`

```
class TestThread(QThread):  
  
    def __init__(self, host: str, port: str):  
        self.host = host  
        self.port = port
```

```
self.t = TestThread("127.0.0.1", "3500")
```

- Создание методов-сеттеров в классе, который унаследован от `QThread`

```
class TestThread(QThread):  
  
    def setConnectParameters(self, host: str, port: str):  
        self.host = host  
        self.port = port
```

```
self.t = TestThread()  
self.t.setConnectParameters("127.0.0.1",  
                             "3500")
```

# Управление QThread


- Сигналы потока:

```
self.t.started.connect(lambda: print("Поток запущен"))  
self.t.finished.connect(lambda: print("Поток завершен"))
```

- Завершение потока:

```
def run(self) -> None:  
    self.status = True  
    count = 1000  
    while self.status:  
        time.sleep(1)  
        print(count)  
        count -= 1
```

```
self.button2.clicked.connect(self.stopThread)  
@QtCore.Slot()  
def stopThread(self):  
    self.t.status = False
```



# **3. Генерация сигналов**



## Создание сигнала и метод emit().

- Для передачи данных между потоками, необходимо создать сигнал.

```
class TestThread(QThread):  
    mysignal = QtCore.Signal(str)
```

- В необходимом месте вызвать emit() и отправить нужные данные:

```
def run(self) -> None:  
    self.status = True  
    count = 1000  
    while self.status:  
        time.sleep(1)  
        self.mysignal.emit(str(count))  
        count -= 1
```

- Обработать поступивший сигнал в основном потоке

```
self.t.mysignal.connect(self.setLineEditText, QtCore.Qt.QueuedConnection)  
  
def setLineEditText(self, text):  
    self.lineEdit.setText(text)
```

**Спасибо за внимание!**