Kostenloses eBook

LERNEN

pyqt5

Free unaffiliated eBook created from
Stack Overflow contributors.

#pyqt5

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: pyqt5

It is an unofficial and free pyqt5 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official pyqt5.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit pyqt5

Bemerkungen

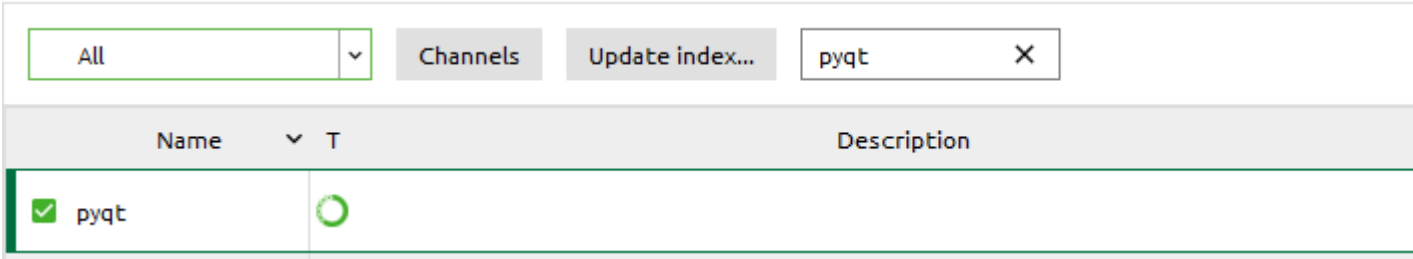
In diesem Abschnitt erhalten Sie einen Überblick darüber, was pyqt5 ist und warum ein Entwickler es verwenden möchte.

Es sollte auch alle großen Themen in pyqt5 erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation für pyqt5 neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

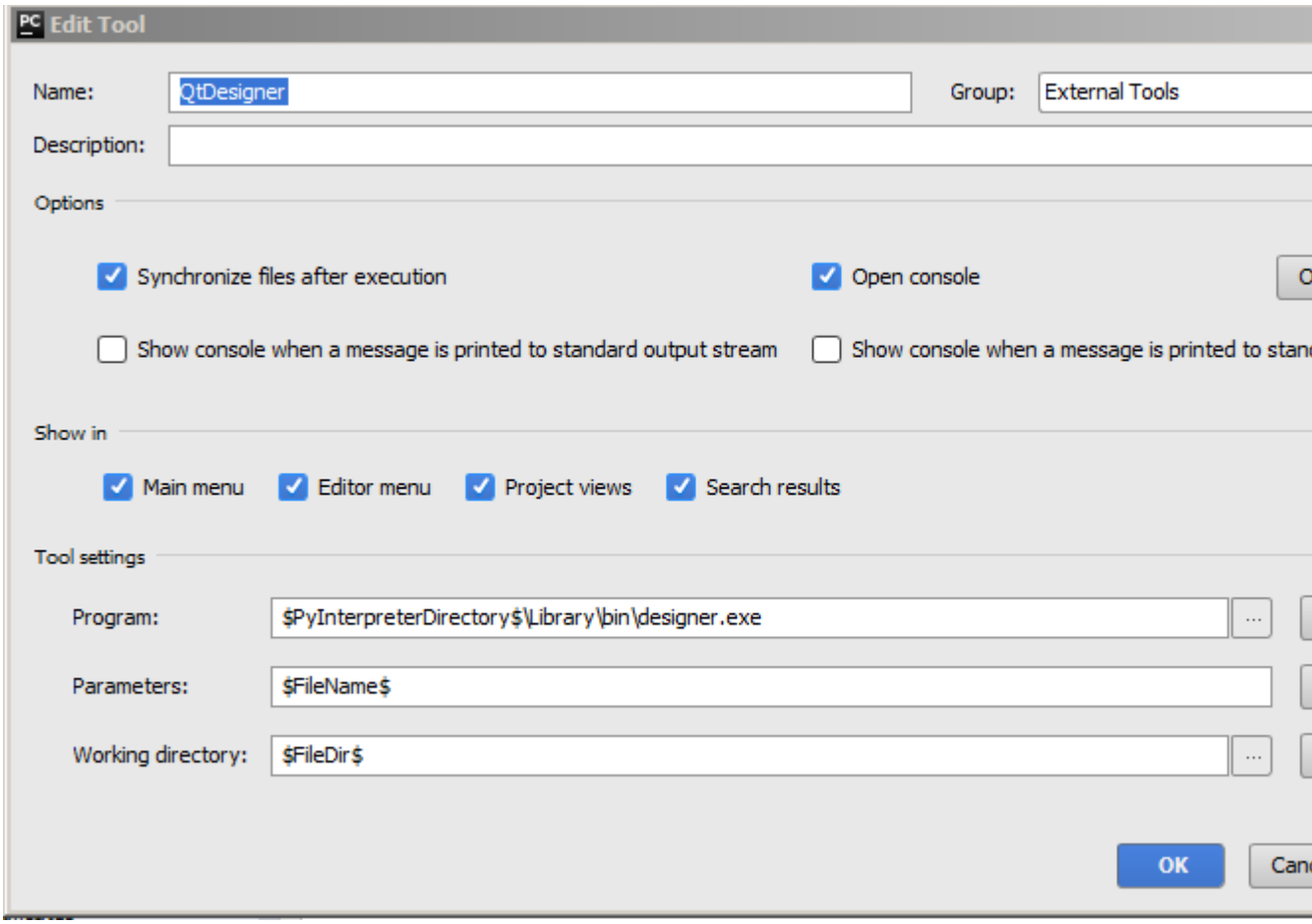
Examples

Installation oder Setup

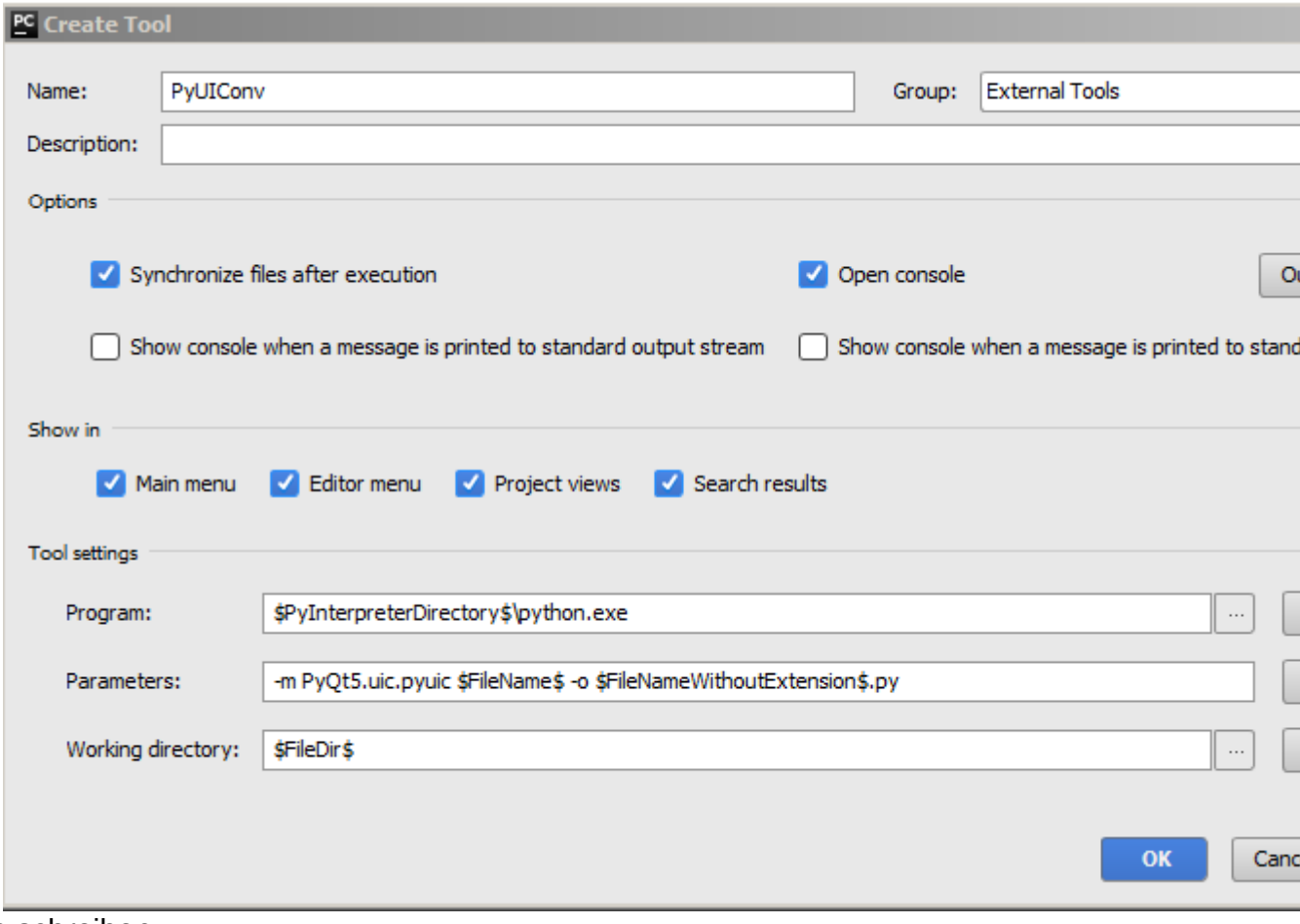
1. Installieren Sie Anaconda (PyQt5 ist eingebaut), insbesondere für Windows-Benutzer.



2. Integration von QtDesigner und QtUIConvert in PyCharm (externe Tools)
    - Öffnen Sie PyCharm- `Settings` > `Tools` > `External Tools`
    - Create Tool (QtDesigner) - zum Bearbeiten von * .ui-Dateien

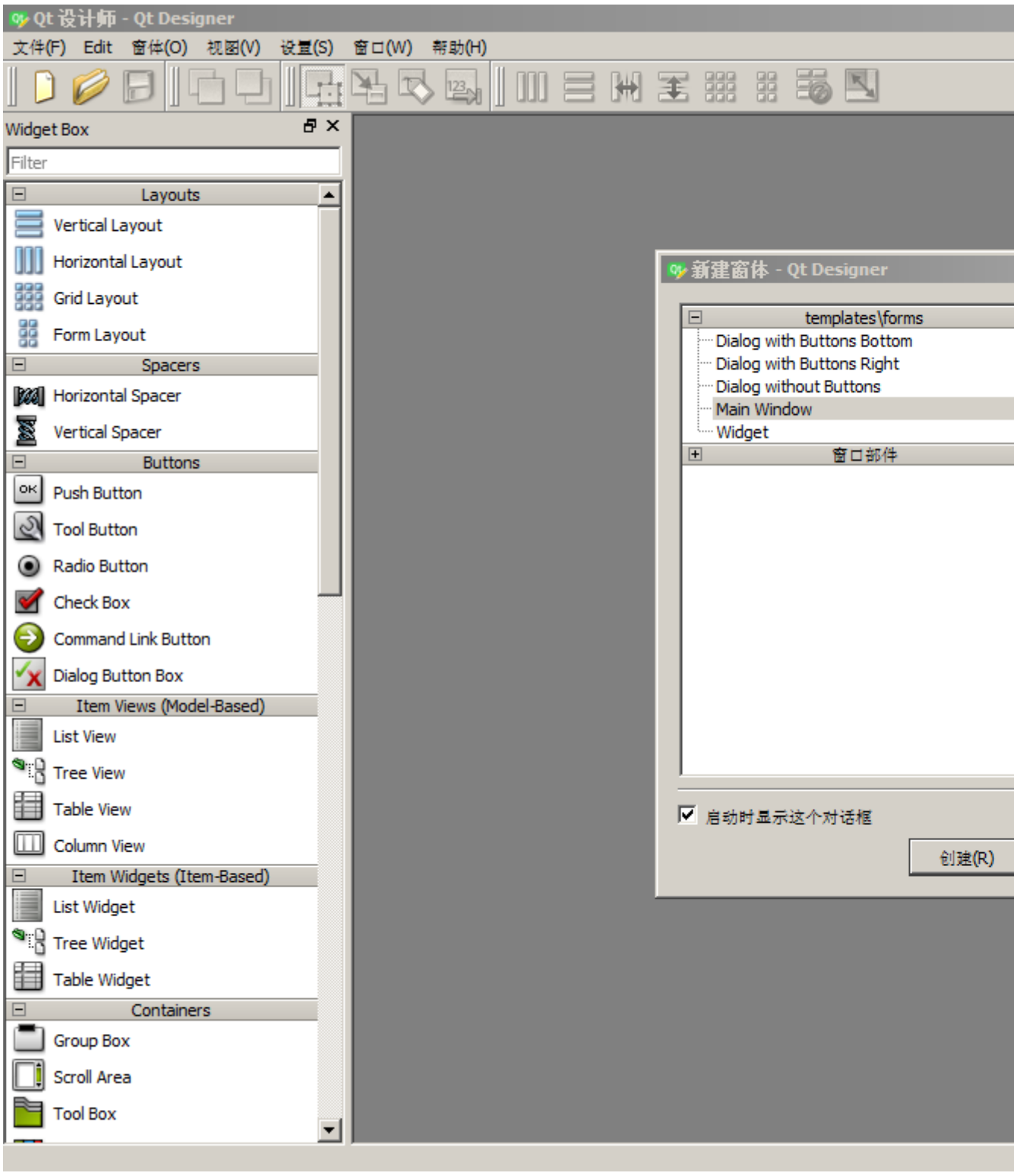


- Create Tool (PyUIConv) - wird verwendet, um * .ui in * .py zu konvertieren

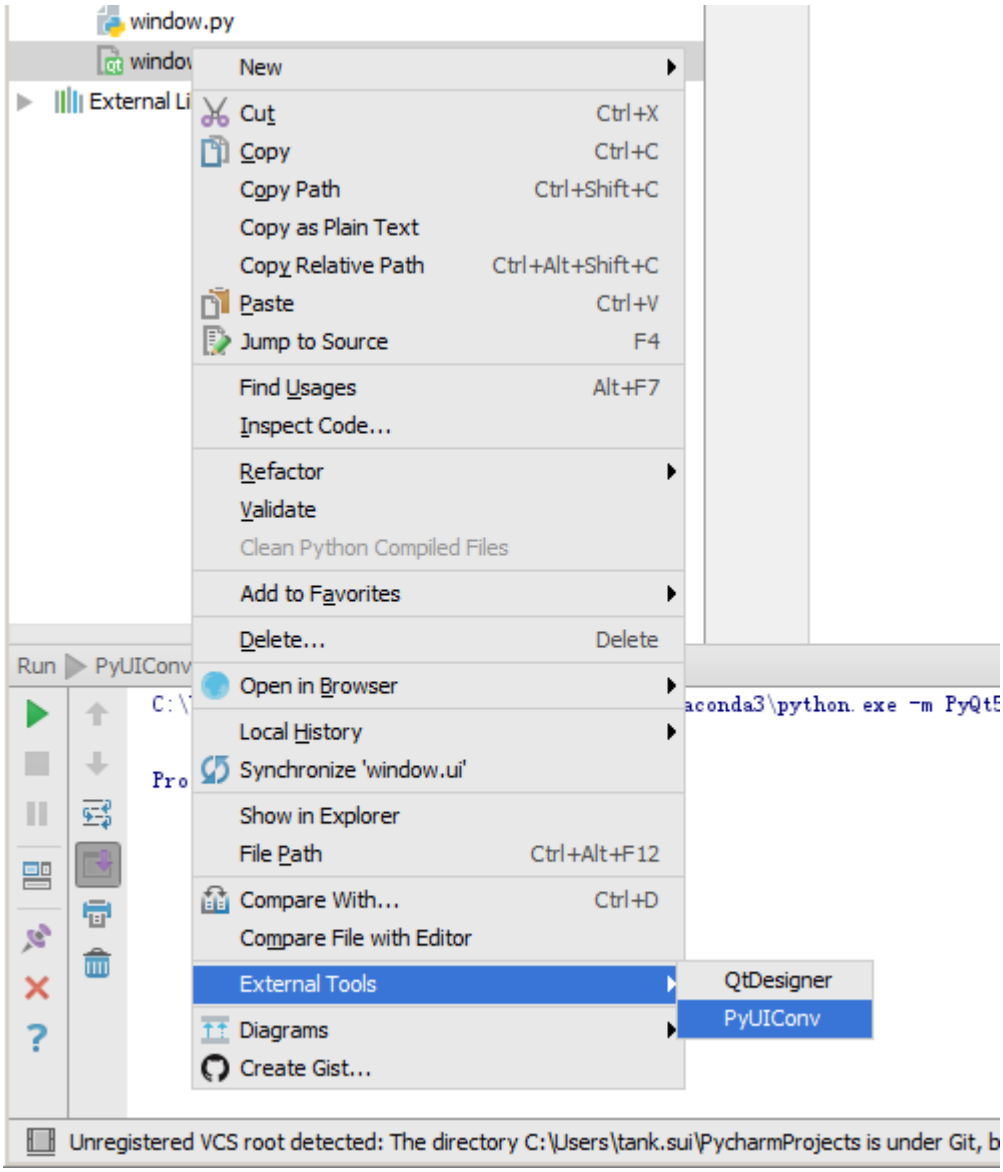


3. Demo schreiben

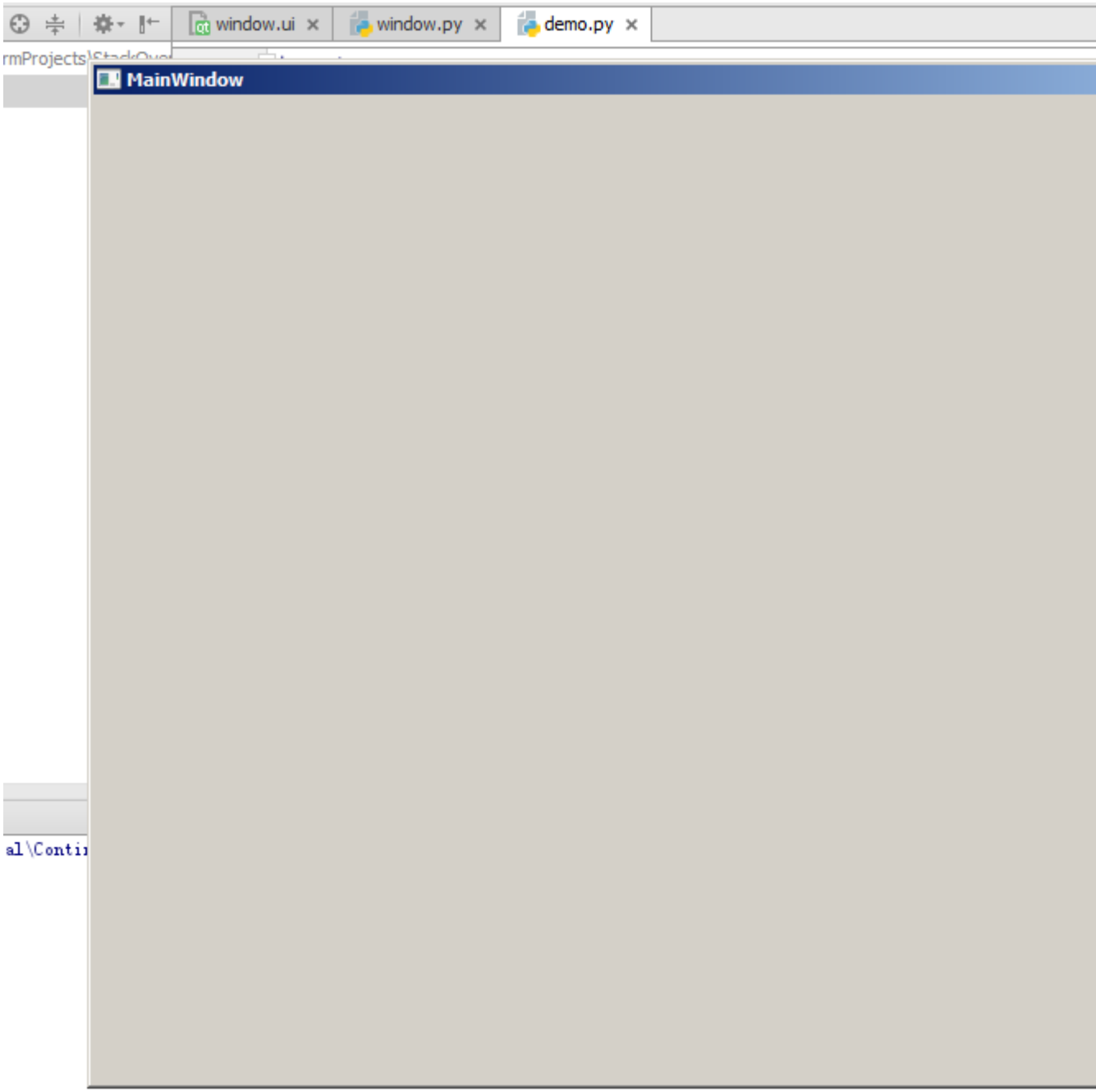
- neues window.ui durch externes Werkzeug (QtDesigner)



- Konvertierung in window.py mit externem Tool (PyUIConv)


window.py
External Li
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Inspect Code...
Refactor
Validate
Clean Python Compiled Files
Add to Favorites
Delete... Delete
Open in Browser
Local History
Synchronize 'window.ui'
Show in Explorer
File Path Ctrl+Alt+F12
Compare With... Ctrl+D
Compare File with Editor
External Tools
Diagrams
Create Gist...
QtDesigner
PyUIConv
Run
PyUIConv
Unregistered VCS root detected: The directory C:\Users\tank.sui\PycharmProjects is under Git, b

- Demo




```
import sys
from PyQt5.QtWidgets import QApplication,QMainWindow
from window import Ui_MainWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    w = QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(w)
    w.show()
    sys.exit(app.exec_())
```

Hallo Weltbeispiel

In diesem Beispiel wird ein einfaches Fenster mit einer Schaltfläche und einer Linienbearbeitung in einem Layout erstellt. Außerdem wird gezeigt, wie ein Signal an einen Steckplatz

angeschlossen wird, sodass durch Klicken auf die Schaltfläche der Zeilenbearbeitung etwas Text hinzugefügt wird.

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


if __name__ == '__main__':

    app = QApplication(sys.argv)

    w = QWidget()
    w.resize(250, 150)
    w.move(300, 300)
    w.setWindowTitle('Hello World')
    w.show()

    sys.exit(app.exec_())
```

Analyse

`app = QtWidgets.QApplication(sys.argv)`

Jede PyQt5-Anwendung muss ein Anwendungsobjekt erstellen. Der Parameter sys.argv ist eine Liste von Argumenten aus einer Befehlszeile. Python-Skripte können von der Shell aus ausgeführt werden.

```
w = QWidget()
```

Das `QWidget` Widget ist die Basisklasse aller Benutzeroberflächenobjekte in PyQt5. Wir stellen den Standardkonstruktor für `QWidget` . Der Standardkonstruktor hat kein übergeordnetes Element. Ein Widget ohne Elternteil wird als Fenster bezeichnet.

```
w.resize(250, 150)
```

Die `resize()` -Methode ändert die `resize()` des Widgets. Es ist 250px breit und 150px hoch.

```
w.move(300, 300)
```

Die `move()` -Methode verschiebt das Widget an eine Position auf dem Bildschirm bei x = 300, y = 300 Koordinaten.

```
w.setWindowTitle('Hello World')
```

Hier legen wir den Titel für unser Fenster fest. Der Titel wird in der Titelleiste angezeigt.

```
w.show()
```

Die `show()` -Methode zeigt das Widget auf dem Bildschirm an. Ein Widget wird zuerst im Speicher erstellt und später auf dem Bildschirm angezeigt.

```
sys.exit(app.exec_())
```

Zum Schluss geben wir die Hauptschleife der Anwendung ein. Die Ereignisbehandlung beginnt an diesem Punkt. Die Hauptschleife empfängt Ereignisse vom Fenstersystem und sendet sie an die Anwendungs-Widgets. Die Hauptschleife endet, wenn wir die `exit()` Methode aufrufen oder das Hauptwidget zerstört wird. Die `sys.exit()` -Methode sorgt für einen sauberen Exit. Die Umgebung wird darüber informiert, wie die Anwendung beendet wurde.

Die Methode `exec_()` hat einen Unterstrich. Dies liegt daran, dass der Exec ein Python-Schlüsselwort ist. Und so wurde stattdessen `exec_()` verwendet.

Anwendungssymbol hinzufügen

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtGui import QIcon


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        self.setGeometry(300, 300, 300, 220)
        self.setWindowTitle('Icon')
        self.setWindowIcon(QIcon('web.png'))

        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Analyse

Funktionsargumente in Python

In Python können benutzerdefinierte Funktionen vier verschiedene Arten von Argumenten annehmen.

1. *Standardargumente:*

- Funktionsdefinition

  ```
  def defaultArg( name, msg = "Hello!"):
  ```

- Funktionsaufruf

```
defaultArg( name)
```

2. *Erforderliche Argumente:*

- Funktionsdefinition

```
def requiredArg (str,num):
```

- Funktionsaufruf:

```
requiredArg ("Hello",12)
```

3. *Keyword-Argumente:*

- Funktionsdefinition

```
def keywordArg( name, role ):
```

- Funktionsaufruf

```
keywordArg( name = "Tom", role = "Manager")
```

oder

```
keywordArg( role = "Manager", name = "Tom")
```

4. *Variable Anzahl von Argumenten:*

- Funktionsdefinition

```
def varlengthArgs(*varargs):
```

- Funktionsaufruf

```
varlengthArgs(30,40,50,60)
```

```
class Example(QWidget):

    def __init__(self):
        super().__init__()
        ...
```

Drei wichtige Dinge in der objektorientierten Programmierung sind Klassen, Daten und Methoden. Hier erstellen wir eine neue Klasse mit dem Namen `Example` . Die `Example` Klasse erbt von der `QWidget` Klasse. Das bedeutet, dass wir zwei Konstruktoren aufrufen: den ersten für die Klasse `Example` und den zweiten für die geerbte Klasse. Die `super()` -Methode gibt das übergeordnete Objekt der `Example` Klasse zurück und wir nennen ihren Konstruktor. Der `self` Variable bezieht sich auf das Objekt selbst.

Warum haben wir `__init__` ?

Überprüfen Sie dies heraus:

```
class A(object):
    def __init__(self):
```

```
        self.lst = []

class B(object):
    lst = []
```

und jetzt versuche es:

```
>>> x = B()
>>> y = B()
>>> x.lst.append(1)
>>> y.lst.append(2)
>>> x.lst
[1, 2]
>>> x.lst is y.lst
True
```

und das:

```
>>> x = A()
>>> y = A()
>>> x.lst.append(1)
>>> y.lst.append(2)
>>> x.lst
[1]
>>> x.lst is y.lst
False
```

> Bedeutet das, dass x in der Klasse B vor der Instantiierung festgelegt wird?

Ja, es ist ein Klassenattribut (es wird von Instanzen gemeinsam genutzt). In Klasse A ist es ein Instanzattribut.

```
self.initUI()
```

Die Erstellung der GUI wird an die Methode `initUI()` delegiert.

```
self.setGeometry(300, 300, 300, 220)
self.setWindowTitle('Icon')
self.setWindowIcon(QIcon('web.png'))
```

Alle drei Methoden wurden von der `QWidget` Klasse geerbt. `setGeometry()` führt zwei Dinge aus: Es lokalisiert das Fenster auf dem Bildschirm und legt die Größe fest. Die ersten beiden Parameter sind die x- und y-Positionen des Fensters. Die dritte ist die Breite und die vierte ist die Höhe des Fensters. Tatsächlich kombiniert es die Methoden `resize()` und `move()` in einer Methode. Die letzte Methode legt das Anwendungssymbol fest. Dafür haben wir ein `QIcon` Objekt erstellt. Das `QIcon` empfängt den Pfad zu unserem Symbol, das angezeigt werden soll.

```
if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Die Anwendungs- und Beispielobjekte werden erstellt. Die Hauptschleife wird gestartet.

Zeigt einen Tooltip an

```
import sys
from PyQt5.QtWidgets import (QWidget, QToolTip,
    QPushButton, QApplication)
from PyQt5.QtGui import QFont


class Example(QWidget):

    def __init__(self):
        super().__init__()

        self.initUI()


    def initUI(self):

        QToolTip.setFont(QFont('SansSerif', 10))

        self.setToolTip('This is a <b>QWidget</b> widget')

        btn = QPushButton('Button', self)
        btn.setToolTip('This is a <b>QPushButton</b> widget')
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('Tooltips')
        self.show()


if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Analyse

```
QToolTip.setFont(QFont('SansSerif', 10))
```

Diese statische Methode legt eine Schriftart zum Rendern von QuickInfos fest. Wir verwenden eine 10px SansSerif-Schrift.

```
self.setToolTip('This is a <b>QWidget</b> widget')
```

Um eine `setTooltip()` zu erstellen, rufen wir die Methode `setTooltip()` . Wir können Rich-Text-Formatierungen verwenden.

```
btn = QPushButton('Button', self)
btn.setToolTip('This is a <b>QPushButton</b> widget')
```

Wir erstellen ein Schaltflächen-Widget und legen einen Tooltip dafür fest.

```
btn.resize(btn.sizeHint())
btn.move(50, 50)
```

Die Schaltfläche wird in der Größe verändert und im Fenster verschoben. Die `sizeHint()` -Methode gibt eine empfohlene Größe für die Schaltfläche an.

Packen Sie Ihr Projekt in excutable / installer

cx_Freeze - ein Tool kann Ihr Projekt einem ausführbaren / installierenden Paket zuordnen

- Nachdem Sie es per Pip installiert haben, um das Paket `demo.py` zu `demo.py` , benötigen wir das folgende `setup.py` .

```
import sys
from cx_Freeze import setup, Executable

# Dependencies are automatically detected, but it might need fine tuning.
build_exe_options = {
    "excludes": ["tkinter"],
    "include_files":[('./platforms','./platforms')] # need qwindows.dll for qt5 application
}

# GUI applications require a different base on Windows (the default is for a
# console application).
base = None
if sys.platform == "win32":
    base = "Win32GUI"

setup(  name = "demo",
        version = "0.1",
        description = "demo",
        options = {"build_exe": build_exe_options},
        executables = [Executable("demo.py", base=base)])
```

- dann bauen

```
python .\setup.py build
```

- dann dist

```
python .\setup.py bdist_msi
```

Erste Schritte mit pyqt5 online lesen: https://riptutorial.com/de/pyqt5/topic/7403/erste-schritte-mit-pyqt5

Kapitel 2: Einführung in Fortschrittsbalken

Einführung

Fortschrittsbalken sind ein wesentlicher Bestandteil der Benutzererfahrung und helfen Benutzern, einen Überblick über die verbleibende Zeit für einen bestimmten Prozess zu erhalten, der auf der GUI ausgeführt wird. Dieses Thema behandelt die Grundlagen der Implementierung einer Fortschrittsleiste in Ihrer eigenen Anwendung.

Dieses Thema wird leicht auf QThread und den neuen Mechanismus für Signale / Slots eingehen. Einige Grundkenntnisse der PyQt5-Widgets werden auch von Lesern erwartet.

Verwenden Sie beim Hinzufügen von Beispielen nur die integrierten PyQt5- und Python-Ins, um die Funktionalität zu demonstrieren.

Nur PyQt5

Bemerkungen

Das Experimentieren mit diesen Beispielen ist der beste Weg, um mit dem Lernen zu beginnen.

Examples

Grundlegende PyQt-Fortschrittsleiste

Dies ist eine sehr grundlegende Fortschrittsanzeige, die nur das Nötigste verwendet.

Es wäre klug, dieses ganze Beispiel bis zum Ende zu lesen.

```
import sys
import time

from PyQt5.QtWidgets import (QApplication, QDialog,
                             QProgressBar, QPushButton)

TIME_LIMIT = 100

class Actions(QDialog):
    """
    Simple dialog that consists of a Progress Bar and a Button.
    Clicking on the button results in the start of a timer and
    updates the progress bar.
    """
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Progress Bar')
        self.progress = QProgressBar(self)
```

```
        self.progress.setGeometry(0, 0, 300, 25)
        self.progress.setMaximum(100)
        self.button = QPushButton('Start', self)
        self.button.move(0, 30)
        self.show()

        self.button.clicked.connect(self.onButtonClick)

    def onButtonClick(self):
        count = 0
        while count < TIME_LIMIT:
            count += 1
            time.sleep(1)
            self.progress.setValue(count)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Actions()
    sys.exit(app.exec_())
```

Die Fortschrittsleiste wird zuerst so importiert wie `from PyQt5.QtWidgets import QProgressBar`

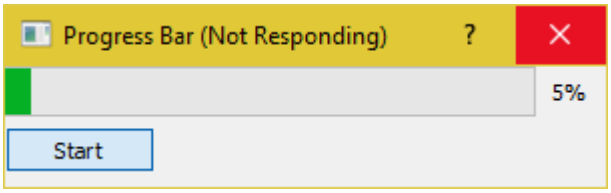
Dann wird es wie jedes andere Widget in `QtWidgets`

Die `self.progress.setGeometry(0, 0, 300, 25)` definiert die `self.progress.setGeometry(0, 0, 300, 25)` `x,y` Positionen im Dialogfeld sowie Breite und Höhe der Fortschrittsleiste.

Dann bewegen wir den Button mit `.move()` um `30px` nach unten, so dass zwischen den beiden Widgets ein Abstand von `5px` .

Hier wird `self.progress.setValue(count)` verwendet, um den Fortschritt zu aktualisieren. Wenn Sie mit `.setMaximum()` einen Maximalwert `.setMaximum()` werden die Werte automatisch für Sie berechnet. Wenn der Maximalwert beispielsweise auf 50 `TIME_LIMIT` ist, `TIME_LIMIT` 100 und `TIME_LIMIT` von 0 auf 2 bis 4 Prozent statt von 0 auf 1 bis 2 pro Sekunde. Sie können auch einen Mindestwert `.setMinimum()` indem Sie `.setMinimum()` , um den Fortschrittsbalken zu zwingen, von einem bestimmten Wert zu starten.

Wenn Sie dieses Programm ausführen, wird eine ähnliche GUI erzeugt.



Wie Sie sehen, wird die GUI definitiv einfrieren und reagiert nicht, bis der Zähler die `TIME_LIMIT` Bedingung erfüllt. Dies liegt daran, dass `time.sleep` das Betriebssystem zu der Überzeugung bringt, dass das Programm in einer Endlosschleife `time.sleep` geblieben ist.

QThread

Wie überwinden wir dieses Problem? Wir können die von PyQt5 bereitgestellte Threading-Klasse verwenden.

```
import sys
import time

from PyQt5.QtCore import QThread, pyqtSignal
from PyQt5.QtWidgets import (QApplication, QDialog,
                             QProgressBar, QPushButton)

TIME_LIMIT = 100

class External(QThread):
    """
    Runs a counter thread.
    """
    countChanged = pyqtSignal(int)

    def run(self):
        count = 0
        while count < TIME_LIMIT:
            count +=1
            time.sleep(1)
            self.countChanged.emit(count)

class Actions(QDialog):
    """
    Simple dialog that consists of a Progress Bar and a Button.
    Clicking on the button results in the start of a timer and
    updates the progress bar.
    """
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Progress Bar')
        self.progress = QProgressBar(self)
        self.progress.setGeometry(0, 0, 300, 25)
        self.progress.setMaximum(100)
        self.button = QPushButton('Start', self)
        self.button.move(0, 30)
        self.show()

        self.button.clicked.connect(self.onButtonClick)

    def onButtonClick(self):
        self.calc = External()
        self.calc.countChanged.connect(self.onCountChanged)
        self.calc.start()

    def onCountChanged(self, value):
        self.progress.setValue(value)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Actions()
    sys.exit(app.exec_())
```

Lassen Sie uns diese Modifikationen abbauen.

```
from PyQt5.QtCore import QThread, pyqtSignal
```

Diese Zeile importiert `Qthread` , eine `PyQt5` Implementierung, um einige Teile (z. B. Funktionen, Klassen) eines Programms im Hintergrund zu teilen und auszuführen (auch als Multithreading bekannt). Diese Teile werden auch als Threads bezeichnet. Alle `PyQt5` Programme verfügen standardmäßig über einen Haupt-Thread und die anderen (Worker-Threads) werden verwendet, um zusätzliche zeitaufwendige und prozessintensive Aufgaben in den Hintergrund zu verlagern, während das Hauptprogramm weiterhin funktioniert.

Das zweite Import- `pyqtSignal` wird verwendet, um Daten (Signale) zwischen `pyqtSignal` und Main-Threads zu senden. In diesem Fall werden wir es verwenden, um dem Haupt-Thread mitzuteilen, dass der Fortschrittsbalken aktualisiert werden soll.

Nun haben wir die while-Schleife für den Zähler in eine separate Klasse namens `External` verschoben.

```
class External(QThread):
    """
    Runs a counter thread.
    """
    countChanged = pyqtSignal(int)

    def run(self):
        count = 0
        while count < TIME_LIMIT:
            count +=1
            time.sleep(1)
            self.countChanged.emit(count)
```

Durch die Unterklassifizierung von `QThread` wir im Wesentlichen `External` in eine Klasse, die in einem separaten Thread ausgeführt werden kann. Darüber hinaus können Threads jederzeit gestartet oder gestoppt werden, was die Vorteile erhöht.

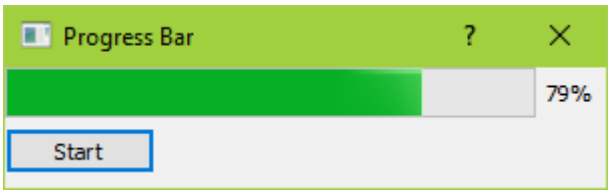
`countChanged` ist der aktuelle Fortschritt, und `pyqtSignal(int)` teilt dem Worker-Thread mit, dass das gesendete Signal vom Typ `int` . `self.countChanged.emit(count)` sendet das Signal einfach an alle Verbindungen im Haupt-Thread (normalerweise kann es auch mit anderen Worker-Threads kommunizieren).

```
def onButtonClick(self):
        self.calc = External()
        self.calc.countChanged.connect(self.onCountChanged)
        self.calc.start()

def onCountChanged(self, value):
    self.progress.setValue(value)
```

Wenn Sie auf die Schaltfläche `self.onButtonClick` wird `self.onButtonClick` ausgeführt und der Thread gestartet. Der Thread wird mit `.start()` gestartet. Es sollte auch beachtet werden, dass wir das `self.calc.countChanged` erstellte Signal `self.calc.countChanged` mit der zum Aktualisieren des Fortschrittsbalkenwerts verwendeten Methode verbunden haben. Bei jeder Aktualisierung von `External::run::count` wird der `int` Wert auch an `onCountChanged` gesendet.

So könnte die GUI nach diesen Änderungen aussehen.



Es sollte sich auch viel ansprechender anfühlen und wird nicht einfrieren.

Einführung in Fortschrittsbalken online lesen:
https://riptutorial.com/de/pyqt5/topic/9544/einfuhrung-in-fortschrittsbalken

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit pyqt5	Ansh Kumar, Community, ekhumoro, suiwenfeng
2	Einführung in Fortschrittsbalken	daegontaven