FREE eBook

LEARNING

pyqt5

Free unaffiliated eBook created from
Stack Overflow contributors.

pyqt5

Sommario

Di

Puoi condividere questo PDF con chiunque ritieni possa trarne vantaggio, scaricando l'ultima versione da: pyqt5

È un ebook pyqt5 non ufficiale e gratuito creato per scopi didattici. Tutto il contenuto è estratto da Stack Overflow Documentazione, che è stato scritto da molte persone laboriose su Stack Overflow. Non è né affiliato con Stack Overflow né pyqt5 ufficiale.

Utilizzare il contenuto presentato in questo libro a proprio rischio; non è garantito che sia corretto o accurato, invia il tuo feedback e le tue correzioni ainfo@zzzprojects.com

Capitolo 1: Iniziare con pyqt5

Osservazioni

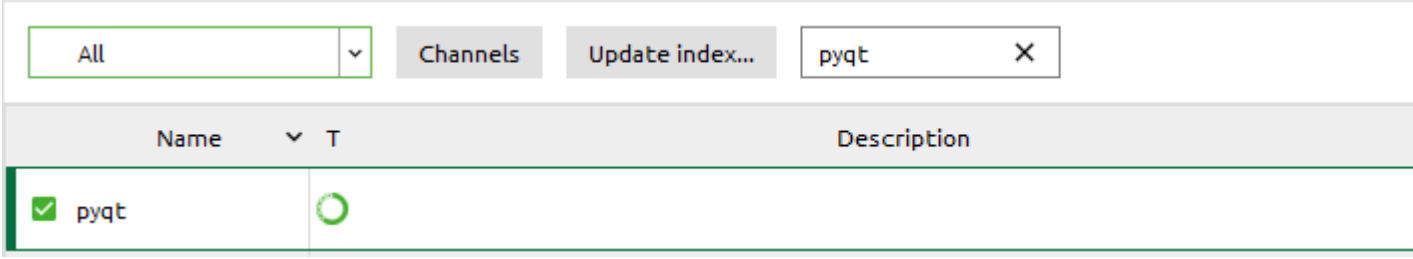
Questa sezione fornisce una panoramica di cos'è pyqt5 e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare eventuali argomenti di grandi dimensioni all'interno di pyqt5 e collegarsi agli argomenti correlati. Dal momento che La documentazione per pyqt5 è nuova, potrebbe essere necessario creare versioni iniziali di questi argomenti correlati.

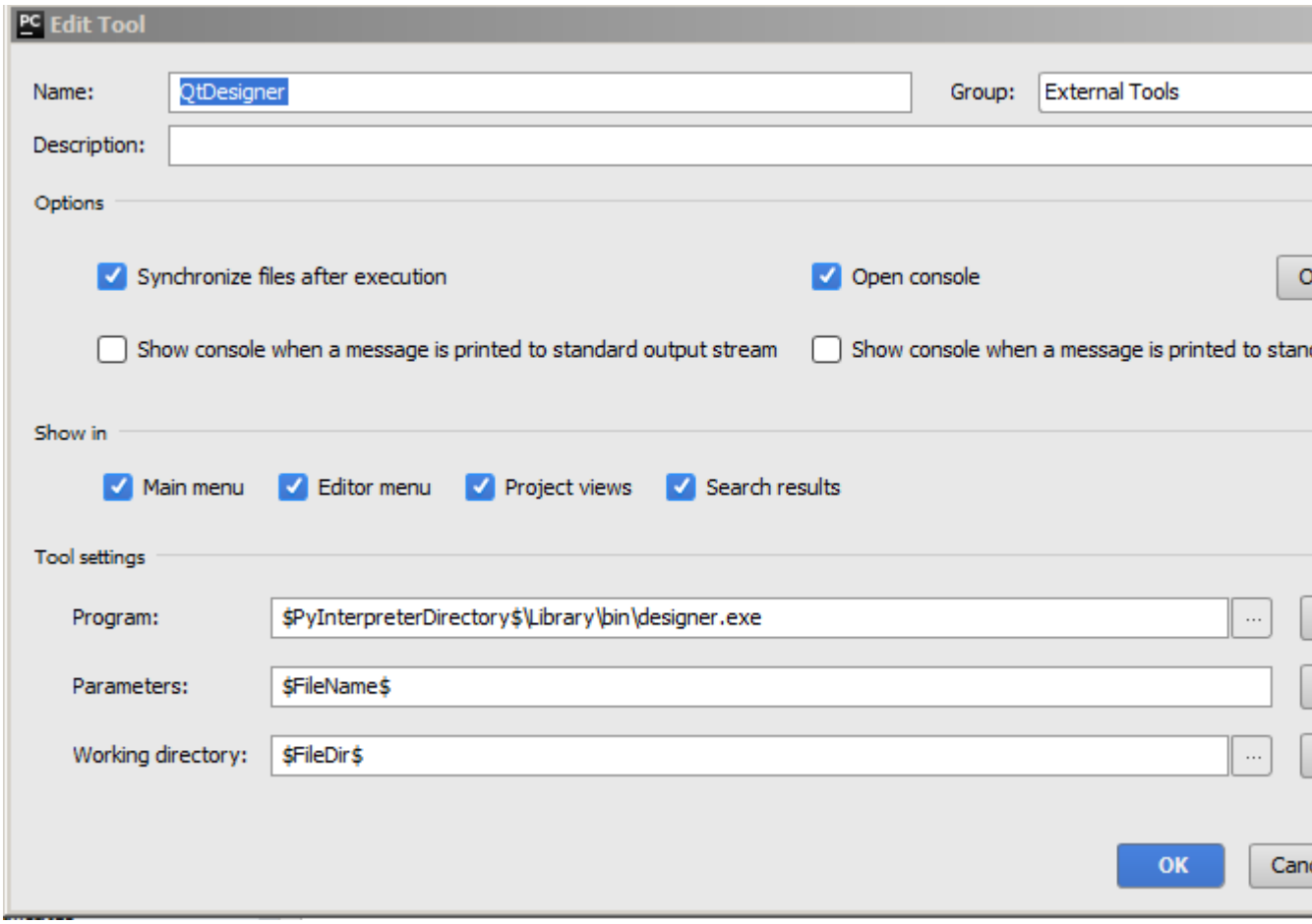
Esempi

Installazione o configurazione

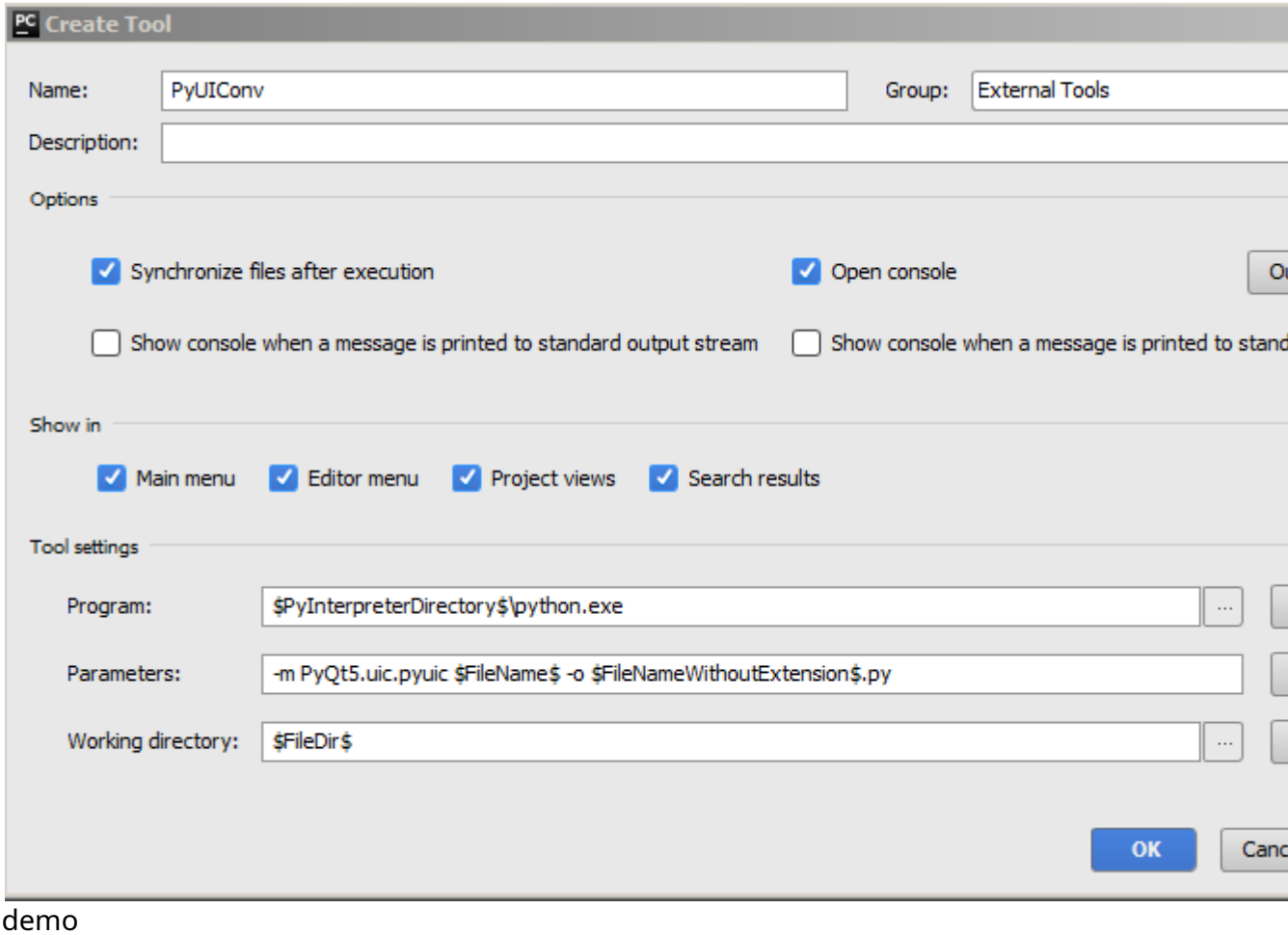
1. Installa Anaconda (PyQt5 è integrato), specialmente per gli utenti di Windows.



2. Integra QtDesigner e QtUIConvert in PyCharm (strumenti esterni)
    - Apri PyCharm Impostazioni > Strumenti > Strumenti esterni
    - Strumento di creazione (QtDesigner) - utilizzato per modificare i file *.ui

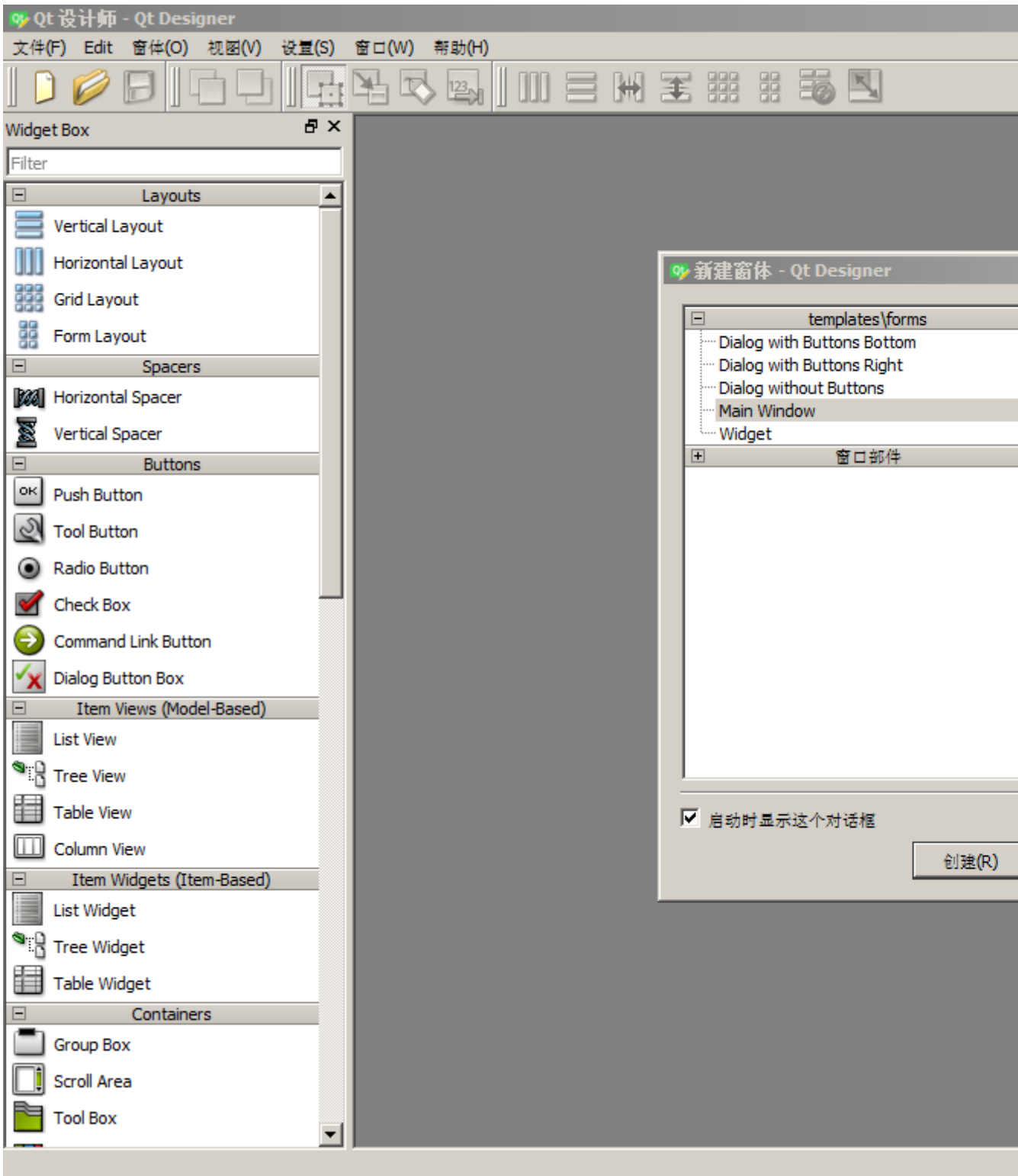


• Strumento di creazione (PyUIConv): utilizzato per convertire *.ui in *.py

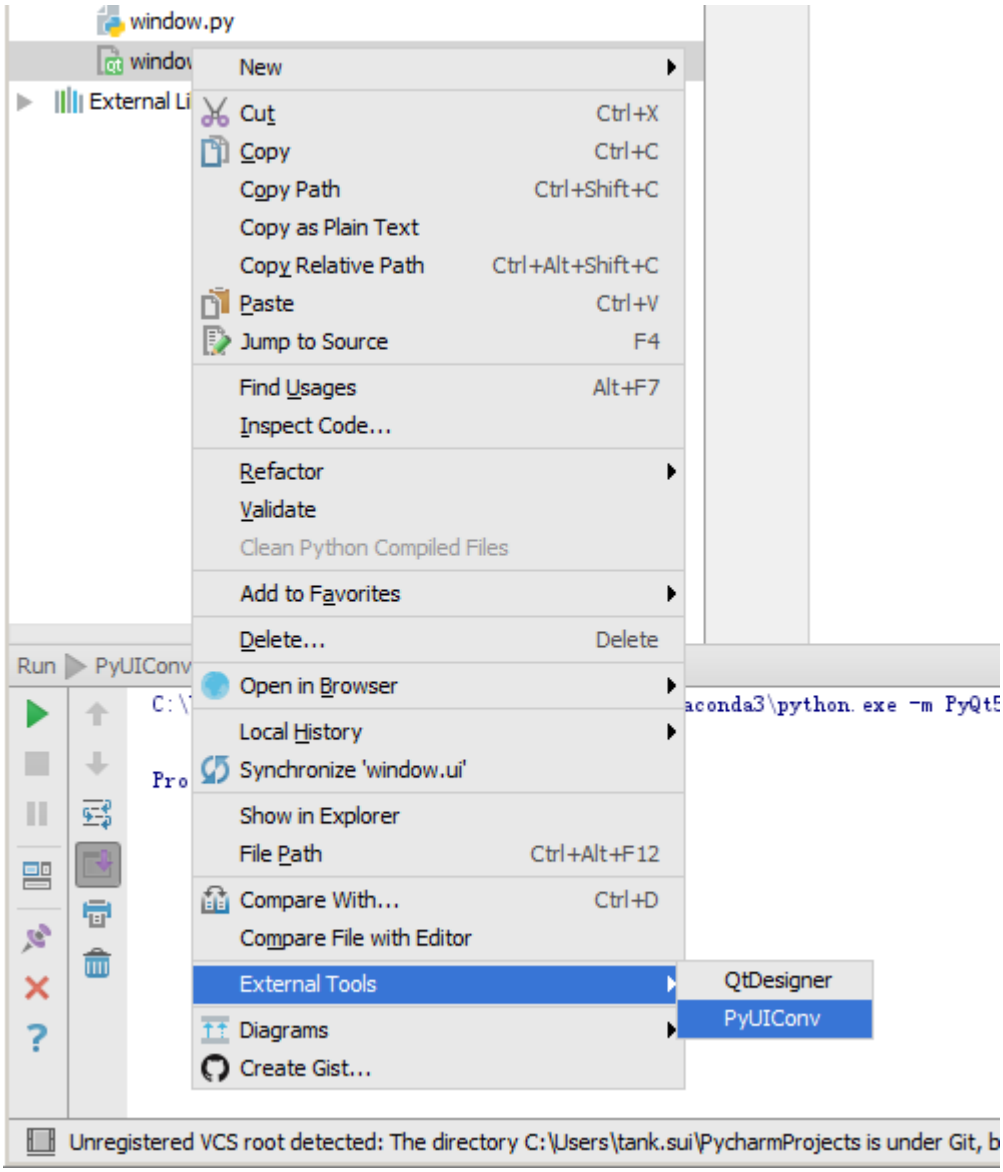


3. Scrivi demo

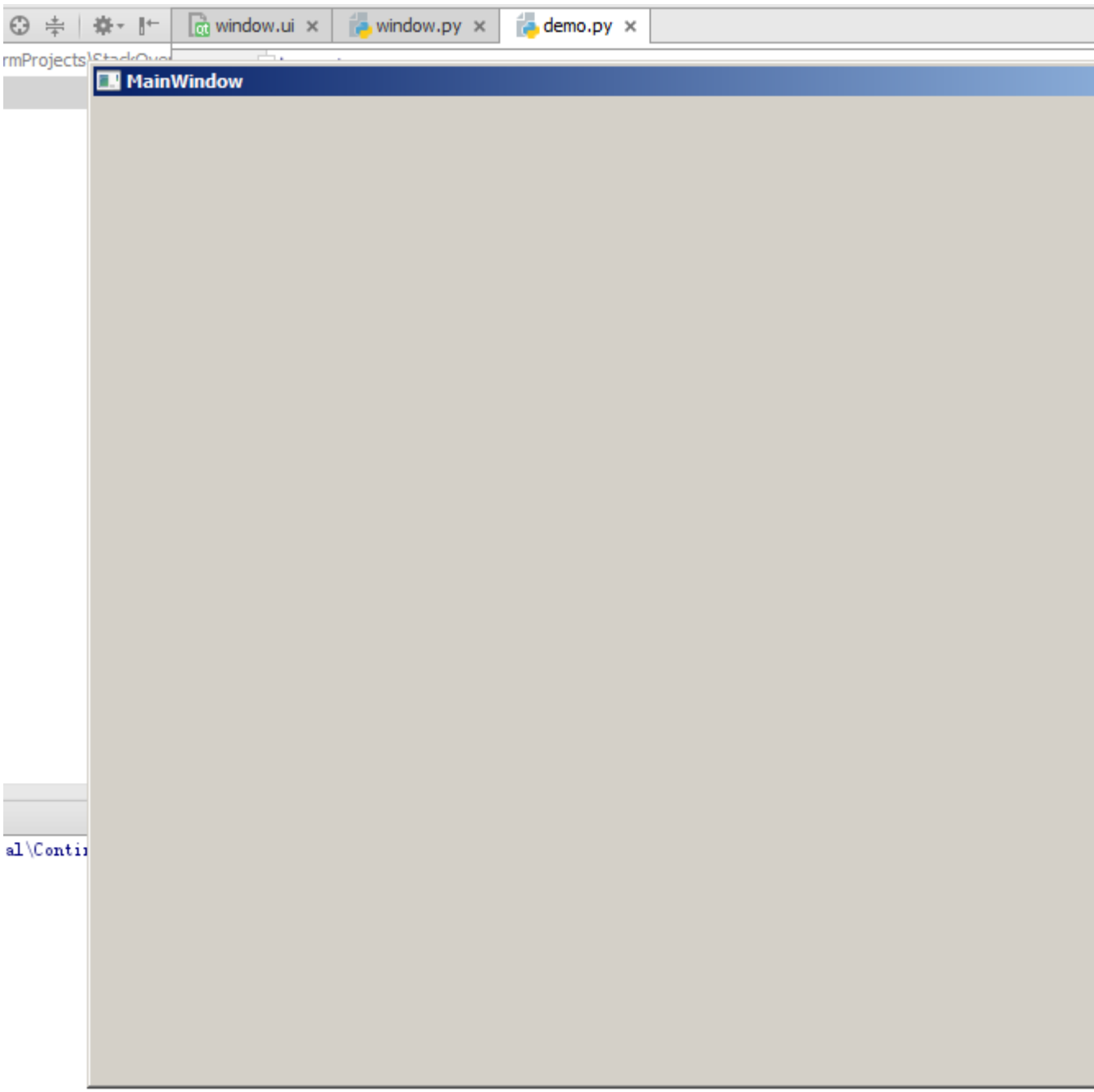
• nuovo window.ui tramite strumento esterno (QtDesigner)



• convertire in window.py tramite uno strumento esterno (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
Run PyUIConv
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
Unregistered VCS root detected: The directory C:\Users\tank.sui\PycharmProjects is under Git, b

• demo



sistema di importazione

```
da PyQt5.QtWidgets import QApplication,QMainWindow dalla finestra
import Ui_MainWindow

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

Ciao esempio mondiale

Questo esempio crea una finestra semplice con un pulsante e una modifica di riga in un layout. Mostra anche come collegare un segnale a uno slot, in modo che facendo clic sul pulsante si aggiunga del testo alla modifica della riga.

```
sistema di importazione
da PyQt5.QtWidgets importa QApplication, QWidget


if __name__ == '__main__':

    app = QApplication(sys.argv)

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

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

Analisi

app = QtWidgets.QApplication(sys.argv)

Ogni applicazione PyQt5 deve creare un oggetto applicazione. Il parametro sys.argv è un elenco di argomenti da una riga di comando. Gli script Python possono essere eseguiti dalla shell.

```
w = QWidget()
```

Il QWidget widget è la classe base di tutti gli oggetti dell'interfaccia utente in PyQt5. Forniamo il costruttore predefinito perQWidget. Il costruttore predefinito non ha padre. Un widget senza genitore è chiamato finestra.

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

Il ridimensiona() Il metodo ridimensiona il widget. È largo 250 px e alto 150 px.

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

Il mossa() Il metodo sposta il widget in una posizione sullo schermo alle coordinate x=300, y=300.

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

Qui impostiamo il titolo per la nostra finestra. Il titolo viene mostrato nella barra del titolo.

```
w.mostra()
```

Il mostrare() Il metodo visualizza il widget sullo schermo. Un widget viene prima creato in memoria e poi mostrato sullo schermo.

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

Infine, entriamo nel mainloop dell'applicazione. La gestione degli eventi inizia da questo punto. Il mainloop riceve gli eventi dal sistema di finestre e li invia ai widget dell'applicazione.

Il mainloop termina se chiamiamo Uscita() metodo o il widget principale viene distrutto. Ilsys.exit() metodo garantisce un'uscita pulita. L'ambiente verrà informato su come è terminata l'applicazione.

Il exec_() il metodo ha un carattere di sottolineatura. È perché l'exec è una parola chiave Python. E quindi, exec_() è stato invece utilizzato.

Aggiunta di un'icona dell'applicazione

```
sistema di importazione
da PyQt5.QtWidgets importa QApplication, QWidget da
PyQt5.QtGui importa QIcon


classe Esempio (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 =
    Esempio()
    sys.exit(app.exec_())
```

Analisi

Argomenti delle funzioni in Python

In Python, le funzioni definite dall'utente possono accettare quattro diversi tipi di argomenti.

1. ***Argomenti predefiniti:***

• Definizione della funzione

```
def defaultArg( nome, msg = "Ciao!"):
```

• Chiamata di funzione

```
defaultArg( nome)
```

2. ***Argomenti richiesti:***

• Definizione della funzione

def RichiestoArg (str,num):

• Chiamata di funzione:

richiestoArg ("Ciao",12)

3. ***Argomenti delle parole chiave:***

• Definizione della funzione

def keywordArg(nome, ruolo):

• Chiamata di funzione

keywordArg(nome = "Tom", ruolo = "Manager")

o

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

4. ***Numero variabile di argomenti:***

• Definizione della funzione

def varlengthArgs(*varargs):

• Chiamata di funzione

lunghezza variabileArgs(30,40,50,60)

```
classe Esempio (QWidget):

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

Tre cose importanti nella programmazione orientata agli oggetti sono classi, dati e metodi. Qui creiamo una nuova classe chiamataEsempio. Il Esempio la classe eredita dal QWidget classe. Ciò significa che chiamiamo due costruttori: il primo per ilEsempio classe e la seconda per la classe ereditata. Il super() il metodo restituisce l'oggetto genitore di Esempio classe e la chiamiamo costruttore. Ilse stesso variabile si riferisce all'oggetto stesso.

Perché abbiamo usato __dentro__?

Controllalo:

```
classe A (oggetto):
        def __init__(self):
                self.lst = []

classe B (oggetto):
        lst = []
```

e ora prova:

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

e questo:

```
> > > x = A()
> > > y = A()
> > > x.lst.append(1)
> > > y.lst.append(2)
> > > x.lst
[1]
> > > x.lst è y.lst
falso
```

Questo significa che x nella classe B viene stabilito prima dell'istanziazione?

Sì, è un attributo di classe (è condiviso tra le istanze). Mentre in classe A è un attributo di istanza.

```
self.initUI()
```

La creazione della GUI è delegata al `initUI()` metodo.

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

Tutti e tre i metodi sono stati ereditati dal `QWidget` classe. Il`setGeometria()` fa due cose: individua la finestra sullo schermo e ne imposta le dimensioni. I primi due parametri sono le posizioni xey della finestra. Il terzo è la larghezza e il quarto è l'altezza della finestra. Infatti, unisce il `ridimensiona()` e `mossa()` metodi in un metodo. L'ultimo metodo imposta l'applicazione icona. Per fare questo, abbiamo creato un`QIcon` oggetto. Il`QIcon` riceve il percorso della nostra icona da visualizzare.

```
if __name__ == '__main__':

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

Vengono creati l'applicazione e gli oggetti di esempio. Viene avviato il ciclo principale.

Visualizzazione di un tooltip

```
sistema di importazione
```

```
dall'importazione di PyQt5.QtWidgets (QWidget, QToolTip,
        QPushButton, QApplication) da
PyQt5.QtGui import QFont


classe Esempio (QWidget):

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

                self.initUI()


        def initUI(self):

                QToolTip.setFont(QFont('SansSerif', 10)) self.setToolTip('Questo è

                un widget <b>QWidget</b>')

                btn = QPushButton('Button', self) btn.setToolTip('Questo è un widget
                <b>QPushButton</b>') btn.resize(btn.sizeHint())

                btn.move(50, 50)

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



if __name__ == '__main__':

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

Analisi

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

Questo metodo statico imposta un carattere utilizzato per il rendering delle descrizioni comandi. Usiamo un font SansSerif da 10px.

```
self.setToolTip('Questo è un widget <b>QWidget</b>')
```

Per creare un tooltip, chiamiamo il `setTooltip()` metodo. Possiamo usare la formattazione rich text.

```
btn = QPushButton('Button', self) btn.setToolTip('Questo è un widget
<b>QPushButton</b>')
```

Creiamo un widget pulsante e impostiamo un tooltip per esso.

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

Il pulsante viene ridimensionato e spostato nella finestra. Il`dimensioneSuggerimento()` Il metodo fornisce una dimensione consigliata per il pulsante.

Comprimi il tuo progetto in eseguibile/installer

cx_Freeze - uno strumento può impacchettare il tuo progetto in eseguibile/installer

- dopo averlo installato tramite pip, per impacchettare `demo.py`, abbiamo bisogno `setup.py` sotto.

```
sistema di importazione
da cx_Freeze impostazione di importazione, eseguibile

# Le dipendenze vengono rilevate automaticamente, ma potrebbe essere necessaria una regolazione fine.
build_exe_options = {
        "esclude": ["tkinter"],
        "include_files":[('./platforms','./platforms')] # necessita di qwindows.dll per l'applicazione qt5
}

# Le applicazioni GUI richiedono una base diversa su Windows (l'impostazione predefinita è per a
# applicazione console).
base = nessuno
if sys.platform == "win32":
        base = "Win32GUI"

setup( nome = "demo",
                versione = "0.1",
                descrizione = "demo",
                options = {"build_exe": build_exe_options}, eseguibili =
                [Executable("demo.py", base=base)])
```

- poi costruire

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

- quindi dist

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

Leggi Iniziare con pyqt5 online: https://riptutorial.com/pyqt5/topic/7403/iniziare-con-pyqt5

Capitolo 2: Introduzione alle barre di avanzamento

introduzione

Le barre di avanzamento sono parte integrante dell'esperienza utente e aiutano gli utenti a farsi un'idea del tempo rimasto per un dato processo che viene eseguito sulla GUI. Questo argomento esaminerà le basi dell'implementazione di una barra di avanzamento nella propria applicazione.

Questo argomento toccherà leggermente QThread e il nuovo meccanismo segnali/slot. Ci si aspetta dai lettori anche una conoscenza di base dei widget PyQt5.

Quando si aggiungono esempi, utilizzare solo PyQt5 e i built-in Python per dimostrare la funzionalità.

Solo PyQt5

Osservazioni

Sperimentare con questi esempi è il modo migliore per iniziare a imparare.

Esempi

Barra di avanzamento di base di PyQt

Questa è una barra di avanzamento molto semplice che utilizza solo ciò che è necessario al minimo

indispensabile. Sarebbe saggio leggere l'intero esempio fino alla fine.

```
sistema di importazione
tempo di importazione

dall'importazione di PyQt5.QtWidgets (QApplication, QDialog,
                                      QProgressBar, QPushButton)

TIME_LIMIT = 100

Azioni di classe (QDialog):
    """
    Finestra di dialogo semplice che consiste in una barra di avanzamento e un
    pulsante. Facendo clic sul pulsante si avvia un timer e si aggiorna la barra di
    avanzamento.
    """
    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):
            conteggio = 0
            mentre conta < TIME_LIMIT:
                contare += 1
                tempo.sonno(1)
                self.progress.setValue(count)

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

La barra di avanzamento viene prima importata in questo modo da PyQt5.QtWidgets importa QProgressBar

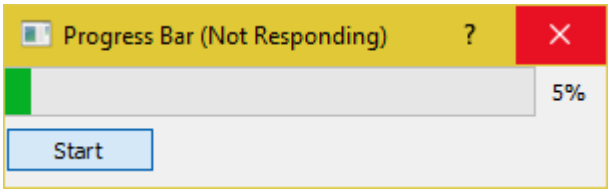
Quindi viene inizializzato come qualsiasi altro widget in QtWidget

La linea self.progress.setGeometry(0, 0, 300, 25) metodo definisce il x, y posizioni sulla finestra di dialogo e larghezza e altezza della barra di avanzamento.

Quindi spostiamo il pulsante usando .mossa() di 30px verso il basso in modo che ci sia uno spazio di 5px tra i due widget.

Qui self.progress.setValue(count) viene utilizzato per aggiornare lo stato di avanzamento. Impostazione di un valore massimo utilizzando .setMaximum() calcolerà automaticamente anche i valori per te. Ad esempio, se il valore massimo è impostato su 50, poichéLIMITE DI TEMPO è 100 salterà da 0 a 2 a 4% invece di 0 a 1 a 2 ogni secondo. Puoi anche impostare un valore minimo usando .setMinimo() forzando la barra di avanzamento a partire da un dato valore.

L'esecuzione di questo programma produrrà una GUI simile a questa.



Come puoi vedere, la GUI si bloccherà sicuramente e non risponderà fino a quando il contatore non soddisfa il LIMITE DI TEMPO condizione. Questo è perchétempo.dormire fa sì che il sistema operativo creda che il programma abbia rimanere bloccato in un ciclo infinito.

QThread

Allora come superiamo questo problema? Possiamo usare la classe di threading fornita da PyQt5.

```
sistema di importazione
tempo di importazione

da PyQt5.QtCore importa QThread, pyqtSignal
```

```
dall'importazione di PyQt5.QtWidgets (QApplication, QDialog,
                                      QProgressBar, QPushButton)

TIME_LIMIT = 100

classe Esterno (QThread):
    """
    Esegue un thread contatore.
    """
    countChanged = pyqtSignal(int)

    def run(self):
        conteggio = 0
        mentre conta < TIME_LIMIT:
            conteggio +=1
            tempo.sonno(1)
            self.countChanged.emit(count)

Azioni di classe (QDialog):
    """
    Finestra di dialogo semplice che consiste in una barra di avanzamento e un
    pulsante. Facendo clic sul pulsante si avvia un timer e si aggiorna la barra di
    avanzamento.
    """
    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 = Esterno()
        self.calc.countChanged.connect(self.onCountChanged) self.calc.start()


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

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

Analizziamo queste modifiche.

```
da PyQt5.QtCore importa QThread, pyqtSignal
```

Questa linea importa `Qthread` il quale è un `PyQt5` implementazione per dividere ed eseguire alcune parti (es: funzioni, classi) di un programma in background (noto anche come multi-threading). Queste parti sono anche chiamate thread. Tutti`PyQt5` i programmi di default hanno un thread principale e gli altri (worker

thread) vengono utilizzati per scaricare in background le attività che richiedono molto tempo ed elaborare attività intensive, pur mantenendo il programma principale in funzione.

La seconda importazione pyqtSignal viene utilizzato per inviare dati (segnali) tra i thread di lavoro e i thread principali. In questo caso lo useremo per dire al thread principale di aggiornare la barra di avanzamento.

Ora abbiamo spostato il ciclo while per il contatore in una classe separata chiamata Esterno.

```
classe Esterno (QThread):
        """
        Esegue un thread contatore.
        """
        countChanged = pyqtSignal(int)

        def run(self):
                conteggio = 0
                mentre conta < TIME_LIMIT:
                        conteggio +=1
                        tempo.sonno(1)
                        self.countChanged.emit(count)
```

Per sottoclasse QThread ci stiamo essenzialmente convertendo Esterno in una classe che può essere eseguita in un thread separato. I thread possono anche essere avviati o interrotti in qualsiasi momento, aumentando i vantaggi.

Qui conteggioCambiato è il progresso attuale e pyqtSignal(int) dice al thread di lavoro che il segnale inviato è di tipo int. Mentre, self.countChanged.emit(count) invia semplicemente il segnale a qualsiasi connessione nel thread principale (normalmente può essere utilizzato anche per comunicare con altri thread di lavoro).

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


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

Quando si fa clic sul pulsante self.onButtonClick verrà eseguito e avvierà anche il thread. Il thread è iniziato con .inizio(). Va anche notato che abbiamo collegato il segnale self.calc.countCambiato abbiamo creato in precedenza il metodo utilizzato per aggiornare il valore della barra di avanzamento. Ogni volta Esterno::run::count è aggiornato il int il valore viene inviato anche a onCountChanged.

Ecco come potrebbe apparire la GUI dopo aver apportato queste modifiche.

Dovrebbe anche sembrare molto più reattivo e non si bloccherà.

Leggi Introduzione alle barre di avanzamento online: https://riptutorial.com/pyqt5/topic/9544/introduction-to-

barre di avanzamento

Titoli di coda

S. No	Capitoli	Contributori
1	Iniziare con pyqt5	Ansh Kumar, Comunità, ekhumoro, suiwenfeng
2	Introduzione a Barre di avanzamento	daegontaven