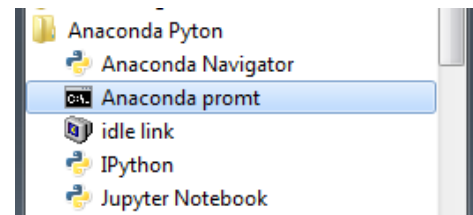


Practicum OPROG – week 1

Hands-on introductie PyQt

Om een gebruikersinterface te maken wordt Qt Designer gebruikt. Start het programma als volgt:

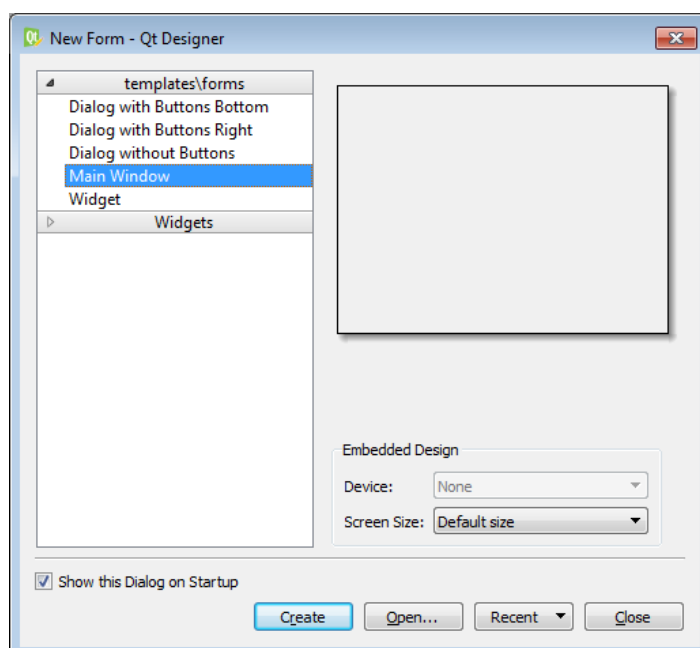
Start de Anaconda prompt vanuit het Startmenu.



In de Anaconda prompt type je:

`designer.exe`

gevolgd door de ENTER-knop op het toetsenbord.

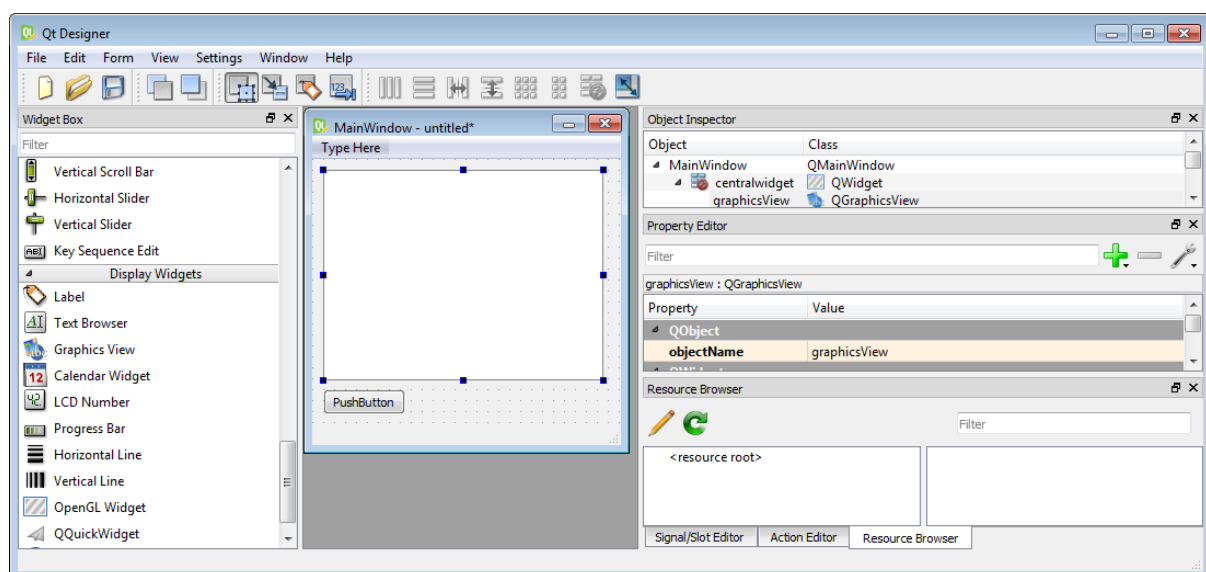


Kies *Main Window* en klik op *Create*.

Je kunt nu componenten op het venster (in VBA noemden we dat een formulier) plaatsen, door widgets vanuit de *Widget Box* (links) naar het ontwerpvenster (midden) te slepen.

Plaats een *Push Button* en een *Graphics View* (zie het voorbeeld hieronder)

In de *Property Editor* (rechts) kun je eigenschappen van de widgets veranderen.

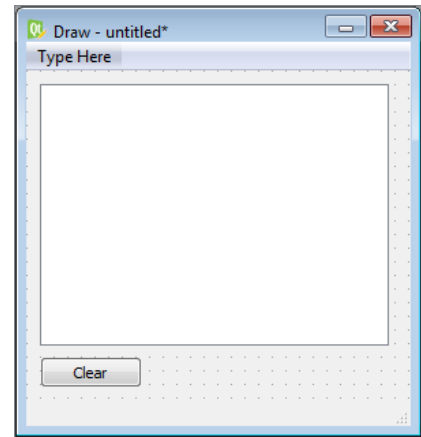


Klik maar eens op het formulier (b.v. naast de PushButton). Je ziet nu de *ObjectName* (onder *QObject*) veranderen in MainWindow. Deze naam kan je eventueel aanpassen. Zoek nu de property *windowTitle* op en verander deze in Draw.

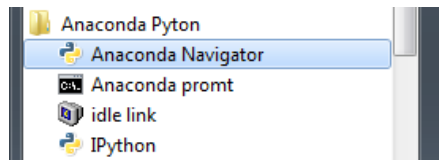
Klik nu op de PushButton. Verander de property *objectName* in btnClear en de property *tekst* in Clear. Verander de *objectName* van de Graphics View in view.

Je ontwerpvenster ziet er nu als volgt uit:

Sla nu je ontwerp op als `Main.ui`.



Start nu het programma  Spyder. Start hiervoor eerst de Anaconda Navigator en start vanuit daar Spyder.



Daarna moet de Anaconda Navigator gesloten worden!

Maak een nieuw Pythonbestand en sla dit op als

`Main.py`.


Je hebt nu een codeframework nodig om het hierboven ontworpen venster te kunnen besturen. Neem de volgende code over:

```
import sys
from PyQt5 import QtWidgets, uic

Ui_MainWindow, QtBaseClass = uic.loadUiType("Main.ui")

class MyApp(QtWidgets.QMainWindow):
    def __init__(self):
        super(MyApp, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

if __name__ == "__main__":
    app = 0 # lost Kernal died probeem op bij herhaald opstarten
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```

Klik vervolgens op Run .

Het zelfontworpen venster verschijnt nu, maar je kunt er nog niets mee doen.

In de code hierboven zie je een klassedefinitie `class MyApp...`

In de klasse is een functie opgenomen `def __init__...` (een functie in een klasse noemen we een methode).

Een klasse kan ook variabelen hebben (in een klasse noemen we dat velden). We zien hier bijvoorbeeld het veld `ui`, dat opgeroepen wordt door de code `self.ui`. Hiermee kunnen we de elementen van de gebruikersinterface (`ui` = user interface) bereiken.

We gaan nu proberen iets te doen als er op de knop geklikt is.

Voeg aan het einde van de `__init__` methode de volgende code toe:

```
self.ui.btnClear.clicked.connect(self.btnClearClicked)
```

We krijgen dan dus:

```
class MyApp(QMainWindow):
    def __init__(self):
        super(MyApp, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.btnClear.clicked.connect(self.btnClearClicked)
```

Hiermee koppelen we de methode `btnClearClicked` aan het klik-event van de knop `btnClear`. Hierbij mag je voor de methode zelf een naam verzinnen. De knopnaam moet de naam zijn die we in Qt Designer aan de knop gegeven hebben.

De methode `btnClearClicked` bestaat nog niet. Deze moeten we zelf schrijven. We voegen dus een nieuwe methode (functie) toe aan de klasse `MyApp`, dus op hetzelfde inspringniveau als de `__init__` methode.

```
class MyApp(QMainWindow):
    def __init__(self):
        ...
    def btnClearClicked(self):
        print("Er is geklikt")
```

Iedere keer dat je op de knop klikt verschijnt nu in de console (van de ontwikkelomgeving) de tekst "Er is geklikt".

We willen nu iets tekenen in de `Graphics View`. Dit is niet direct mogelijk. Een `Graphics View` is slechts een weergave van een bestaande tekening. De werkelijke tekening moet gemaakt worden in een `QGraphicsScene`.

Definieer de scene in de constructor (een betere naam voor de `__init__` methode):

```
def __init__(self):
    super(MyApp, self).__init__()
    self.ui = Ui_MainWindow()
    self.ui.setupUi(self)
    self._scene = QGraphicsScene()
    self.ui.view.setScene(self._scene)
    self._scene.setSceneRect(0,0,100,100)
    self.ui.btnClear.clicked.connect(self.btnClearClicked)
```

Op regel 5 van van bovenstaand codefragment wordt de scene gemaakt. Op regel 6 wordt de scene gekoppeld aan de view (de view laat de tekening in de scene zien). Op regel 7 wordt het werkgebied van de scene gedefinieerd.



We kunnen nu op de scene tekenen. Vervang de code in het klik-event van de knop door:

```
def btnClearClicked(self):  
    item = QtWidgets.QGraphicsRectItem(0, 0, 50, 50)  
    self._scene.addItem(item)
```

Het zou natuurlijk mooi zijn als we een vierkant konden tekenen met de muis.

Definieer daarvoor in de constructor de volgende events:

```
self.ui.view.mousePressEvent = self.viewMouseDown  
self.ui.view.mouseReleaseEvent = self.viewMouseUp
```

En voeg de volgende methodes toe:

```
# Mouse down en mouse up events  
def viewMouseDown(self, event):  
    # mapToScene zet muiscoördinaten om in scene coördinaten  
    point = self.ui.view.mapToScene(event.pos())  
    self._x = point.x()  
    self._y = point.y()  
def viewMouseUp(self, event):  
    point = self.ui.view.mapToScene(event.pos())  
    x = point.x()  
    y = point.y()  
    width = x - self._x  
    height = y - self._y  
    item = QtWidgets.QGraphicsRectItem(self._x, self._y,  
                                       width, height)  
    self._scene.addItem(item)
```

Vervang tenslotte de code in de Clear-methode door:

```
self._scene.clear()
```

Wat gebeurt er als je op de knop klikt?