



Initiation à pyQT



[PyQt](#) est un module libre qui permet de lier le langage Python avec la bibliothèque Qt distribuée sous deux licences : une commerciale et la GNU GPL. Il permet ainsi de créer des interfaces graphiques en Python.

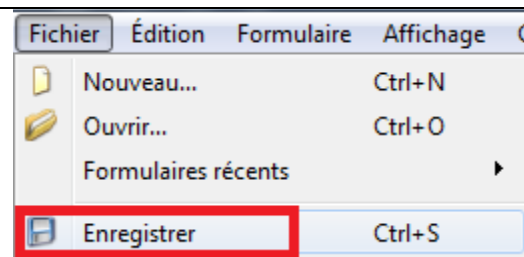
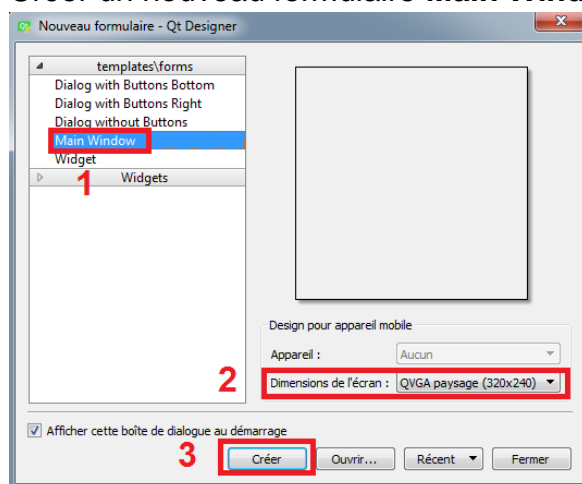
L'environnement de développement Edupython dispose d'une icône « Qt Designer » dans PyScripter.



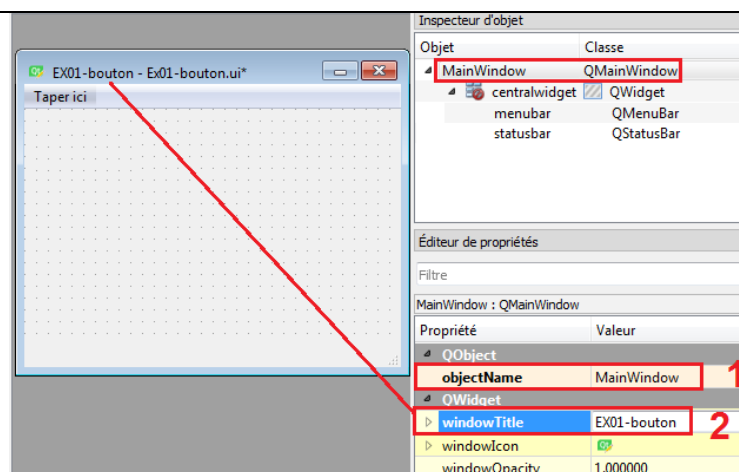
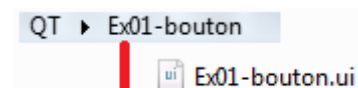
1 Bouton et label (nom des fichiers créés : EX01-bouton.ui, EX01-bouton.py)

1.1 Qt Designer

Créer un nouveau formulaire **Main Windows**



Créer un répertoire **\QT\Ex01-bouton**
Dans lequel sera enregistré le fichier
« **Ex01-bouton.ui** »

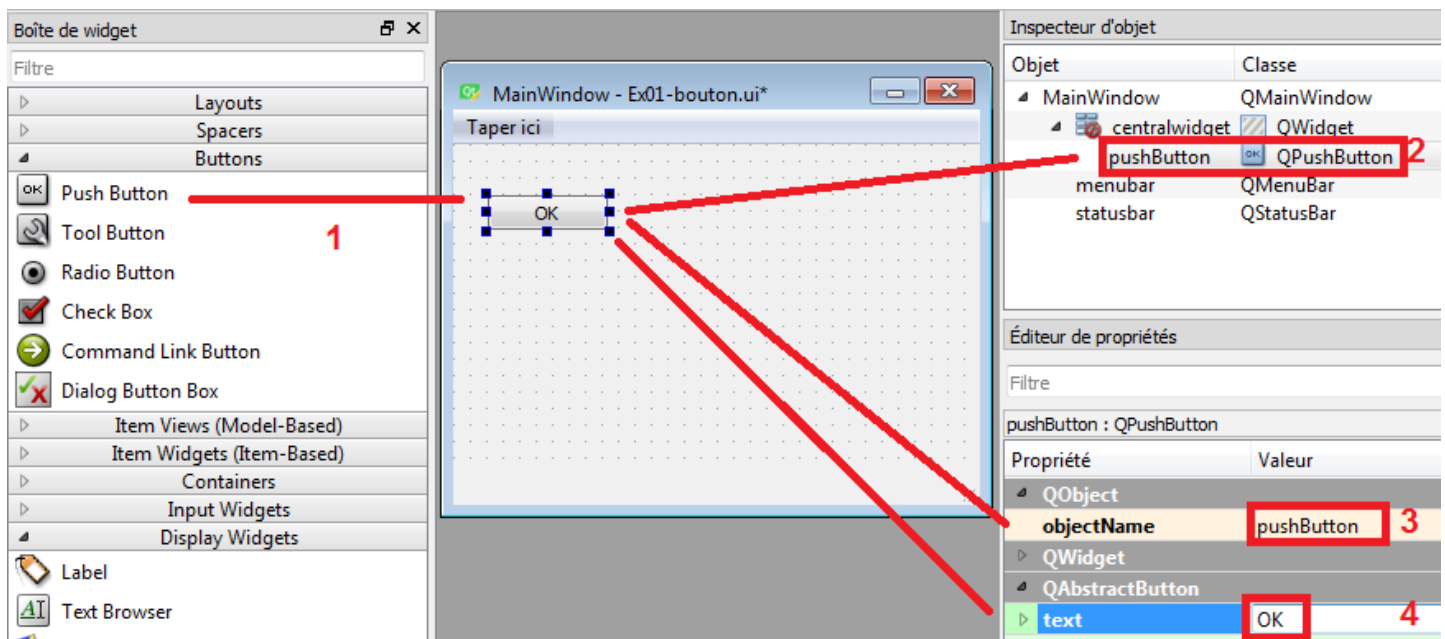


Bien repérer le nom de la fenêtre principale :

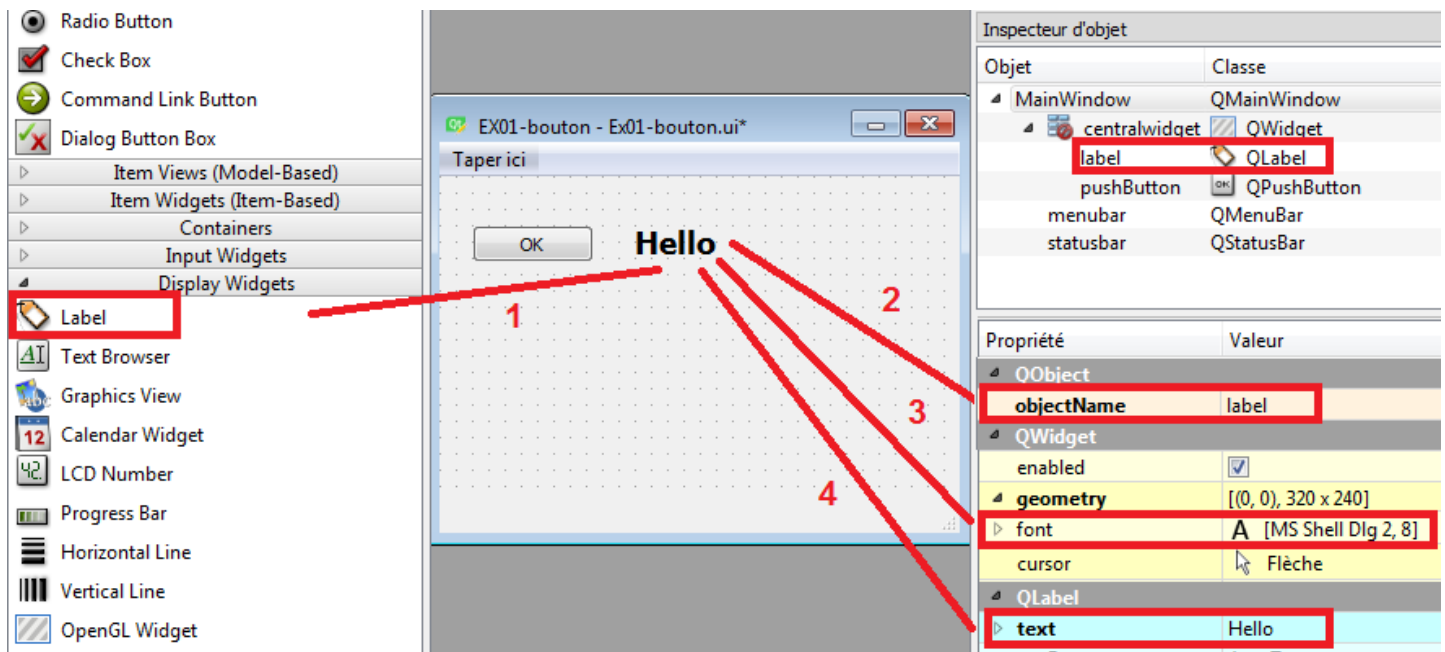
MainWindows (1)

Il est possible de renommer le titre de la fenêtre en changeant l'attribut de **windowTitle (2)**

- Ajouter un bouton (1)
- Repérer le nom de l'objet **pushButton** (3)
- Changer le texte du bouton (4)



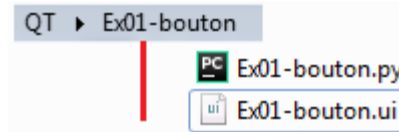
- Ajouter un label (1)
- Repérer le nom de l'objet : **label** (2)
- Changer la taille de la police de caractère : font (3)
- Changer le texte du label (4)



Enregistrer le formulaire

1.2 Code en python

Créer un fichier **Ex01-bouton.py** dans le même répertoire que le fichier ui



Coller le code ci-dessous :

```
#Ex01-bouton
#Quand on clique sur le bouton, on change le nom du label par Bonjour

from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtWidgets, QtMultimedia, QtGui, uic
import sys

class MainWindows(QMainWindow):
    def __init__(self):
        super(MainWindows, self).__init__()
        uic.loadUi('Ex01-bouton.ui', self) #chargement du formulaire XML
        self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable
        #Quand on clique sur le pushButton, on appelle la méthode boutonOK
        self.pushButton.clicked.connect(self.boutonOk)
        self.show() #affiche la fenêtre MainWindows

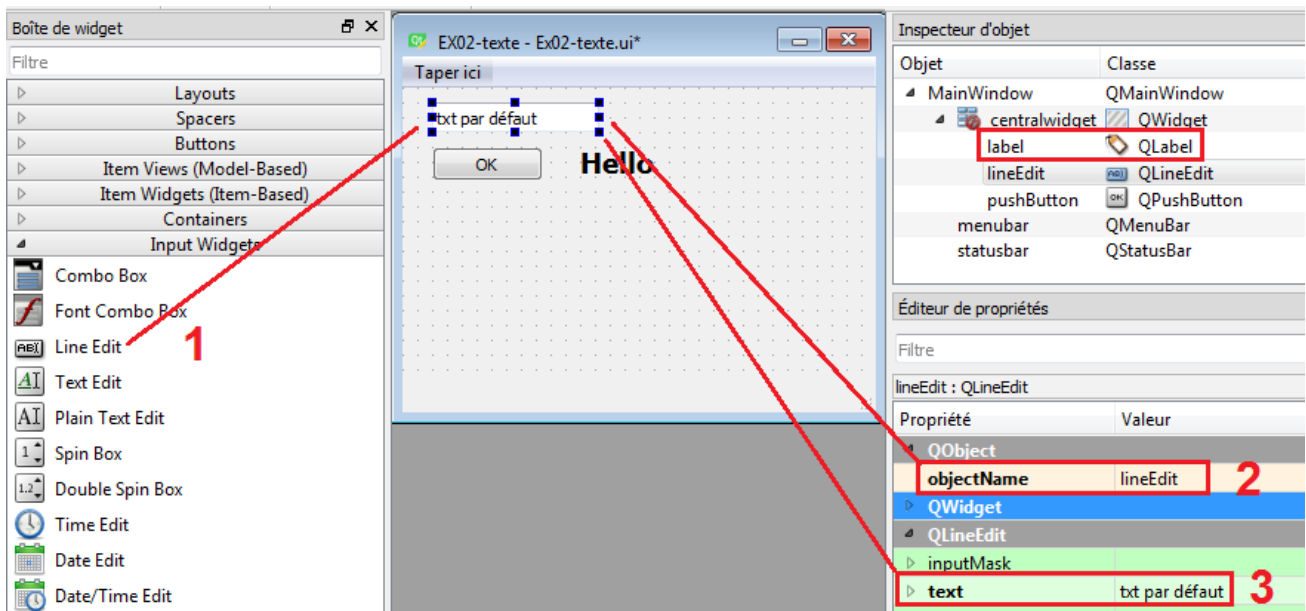
    def boutonOk(self):
        print('bouton cliqué')
        self.label.setText("Bonjour") #on change le nom du label

app = QApplication(sys.argv)
window = MainWindows()
app.exec_()
```

2 Edition de texte (nom des fichiers créés : EX02-texte.ui, EX02-texte.py)

2.1 Qt Designer

- Ajouter une ligne d'édition (1)
- Repérer le nom de l'objet : **lineEdit** (2)
- Changer le texte affiché lors de l'exécution de la fenêtre (3)



2.2 Code en python

Le code est identique au précédent. Seul le nom du formulaire change ainsi que le code de la méthode boutonOK.

```
uic.loadUi('Ex02-texte.ui', self) #chargement du formulaire XML

def boutonOk(self):
    print('bouton cliqué')
    txt=self.lineEdit.text() #récupère le texte de lineEdit
    print('texte saisi:',txt,type(txt)) #affiche le texte et le type
    if txt.isnumeric(): #le texte est-il un chiffre ?
        a=int(txt) #conversion en entier
        a=a+20 #ajoute 20
        self.label.setText(str(a)) #affiche le résultat en chaîne de caractères
    else:
        self.label.setText("nombre svp")
```

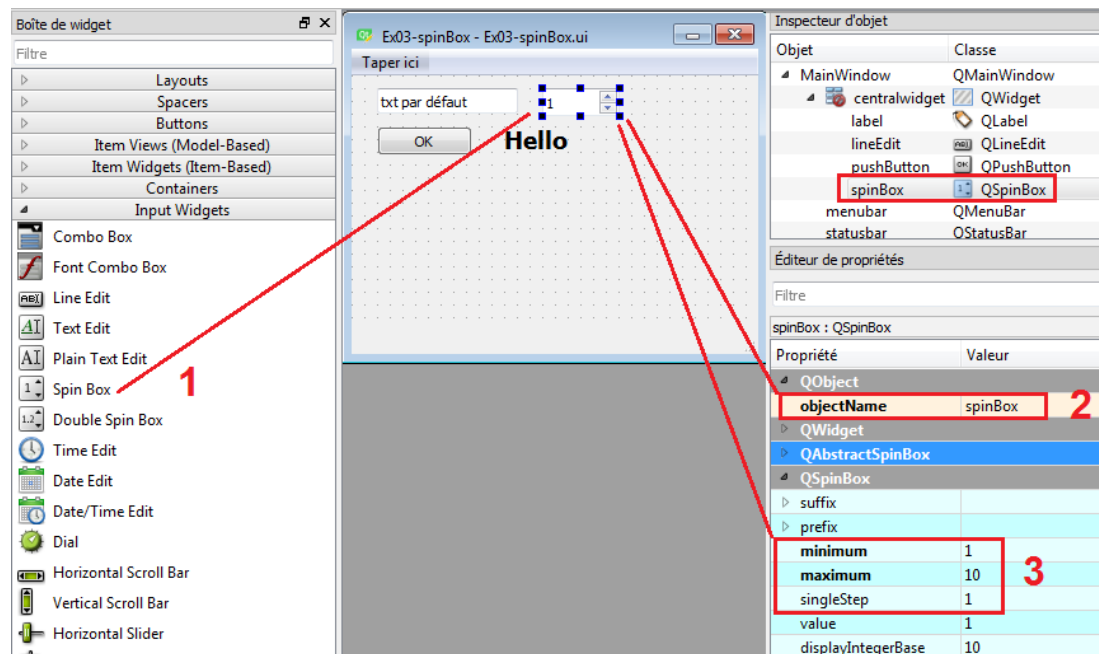
Attention :

Le type contenu dans lineEdit est obligatoirement str. Si l'on veut faire un calcul, il faudra convertir le str en int. Pour afficher le résultat du calcul dans le label il faut faire l'opération inverse, c'est-à-dire convertir le nombre entier en chaîne de caractères str.

3 Spin box (nom des fichiers créés : Ex03-spinBox.ui, Ex03-spinBox.py)

3.1 Qt Designer

- Ajouter une spin box (1)
- Repérer le nom de l'objet : **spinBox** (2)
- Changer les paramètres min, max (3)



3.2 Code en python

Le code est identique au précédent. Seul le nom du formulaire change ainsi que le code de la méthode boutonOK.

```
uic.loadUi('Ex03-spinBox.ui', self) #chargement du formulaire XML

def boutonOk(self):
    print('bouton cliqué')
    txt=self.lineEdit.text() #récupère le texte de lineEdit
    print('texte saisi:',txt,type(txt)) #affiche dans la console le texte et le type
    b=self.spinBox.value() #récupère le nombre de la spinbox
    print('valeur de la spin box',b) #affiche cette valeur dans la console
    if txt.isnumeric(): #le texte est-il un chiffre ?
        a=int(txt) #conversion en entier
        a=a+b #ajoute la valeur b
        self.label.setText(str(a)) #affiche le résultat
    else:
        self.label.setText("nombre svp")
```

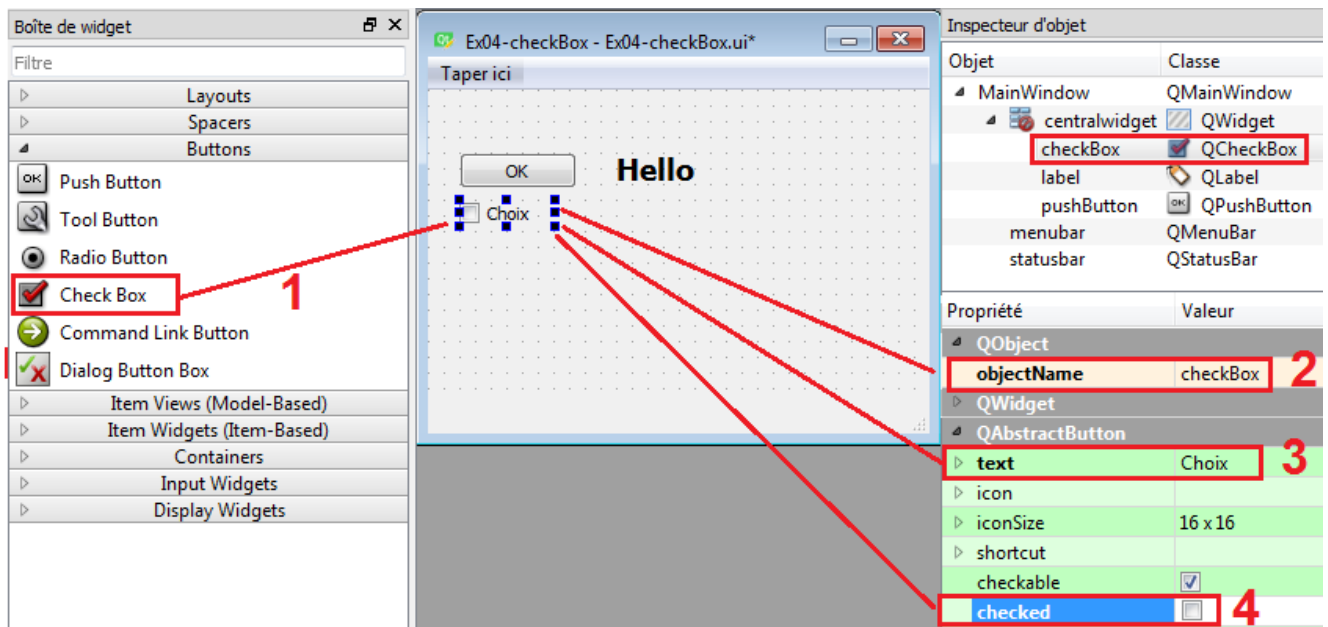
Attention :

Le type de la spinbox est int (entier).

4 Check Box (nom des fichiers créés : Ex04-checkBox.ui, Ex04-checkBox.py)

4.1 Qt Designer

- Ajouter une check Box (1)
- Repérer le nom de l'objet : **checkBox** (2)
- Changer le texte (3)
- Checked : validé par défaut ou pas (4)



4.2 Code en python

Le code est identique au précédent. Seul le nom du formulaire change ainsi que le code de la méthode boutonOK.

```

uic.loadUi('Ex04-checkBox.ui', self) #chargement du formulaire XML
self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable
#Quand on clique sur le pushButton, on appelle la méthode boutonOK
self.pushButton.clicked.connect(self.boutonOk)
#Quand on clique sur la checkbox, on appelle la méthode chkBox
self.checkBox.stateChanged.connect(self.chkBox)
self.show() #affiche la fenêtre MainWindows

def boutonOk(self):
    print('bouton cliqué')
    chk=self.checkBox.isChecked() #récupère l'état du checkBox
    print('chk',chk,type(chk)) #affiche dans la console l'état du check box et le type

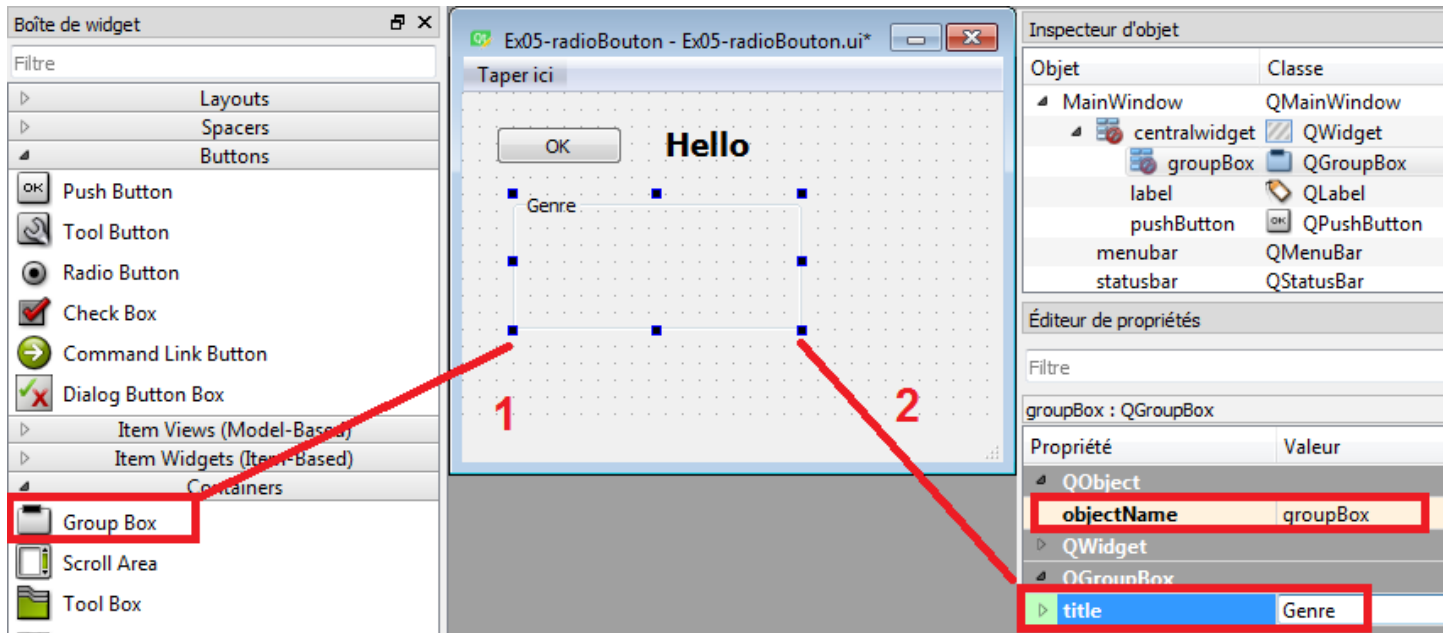
def chkBox(self):
    chk=self.checkBox.isChecked() #récupère l'état du checkBox
    if chk==True:
        self.label.setText("checkBox True") #affiche le résultat dans le label
    else:
        self.label.setText("checkBox False") #affiche le résultat dans le label

```

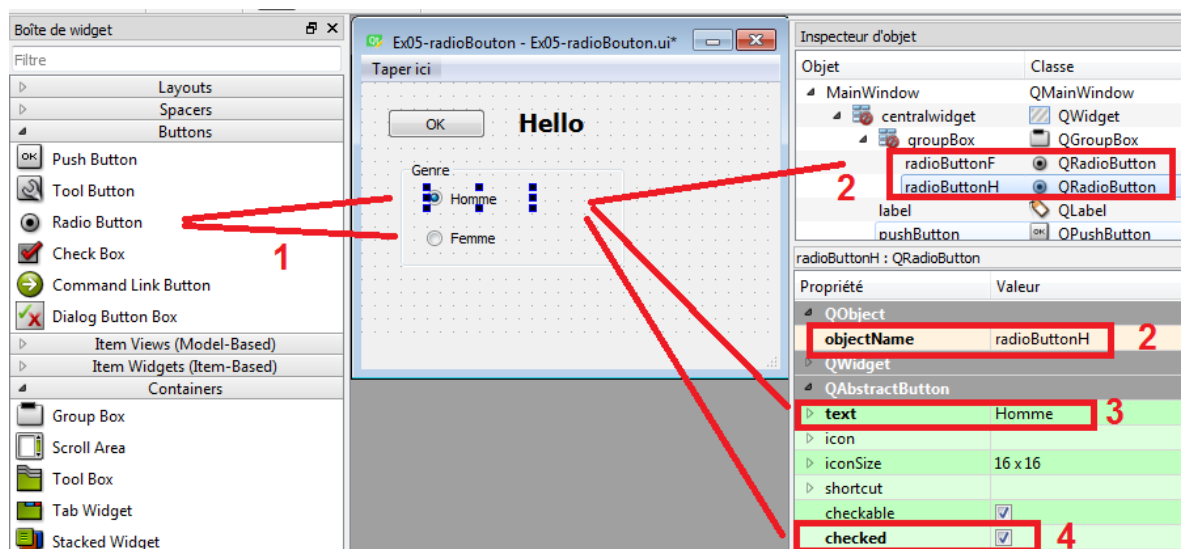
5 radio bouton (nom des fichiers créés : Ex05-radioBouton.ui, Ex05-radioBouton.py)

5.1 Qt Designer

- Ajouter un group Box (1)
- Changer le texte (2)



- Ajouter 2 radioButton (1)
- Repérer le nom des 2 objets : radioButtonF, radioButtonH (2)
- Changer le texte (3)
- Checked validé pour le 1^{er} radioButton (4)



5.2 Code en python

```
#Ex05-radioBouton
#Quand on clique sur le bouton, on affiche l'état des radio boutons
#Dès qu'un radio bouton est cliqué , on change le label

from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtWidgets, QtMultimedia, QtGui, uic
import sys

class MainWindows(QMainWindow):
    def __init__(self):
        super(MainWindows, self).__init__()
        uic.loadUi('Ex05-radioBouton.ui', self) #chargement du formulaire XML
        self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable
        #Quand on clique sur le pushButton, on appelle la méthode boutonOk
        self.pushButton.clicked.connect(self.boutonOk)
        #Quand on clique sur les radiobuttons, on appelle la méthode verifieRadios
        #L'utilisation de lambda permet de transmettre la source du signal à la methode comme argument.
        self.radioButtonH.toggled.connect(lambda: self.verifieRadios(self.radioButtonH))
        self.radioButtonF.toggled.connect(lambda: self.verifieRadios(self.radioButtonF))
        self.show() #affiche la fenêtre MainWindows

    def boutonOk(self):
        print('bouton cliqué')
        h=self.radioButtonH.isChecked() #récupère l'état du checkBox
        f=self.radioButtonF.isChecked() #récupère l'état du checkBox
        print('radio h',h,'radio f',f,type(h)) #affiche dans la console l'état du check box et le type

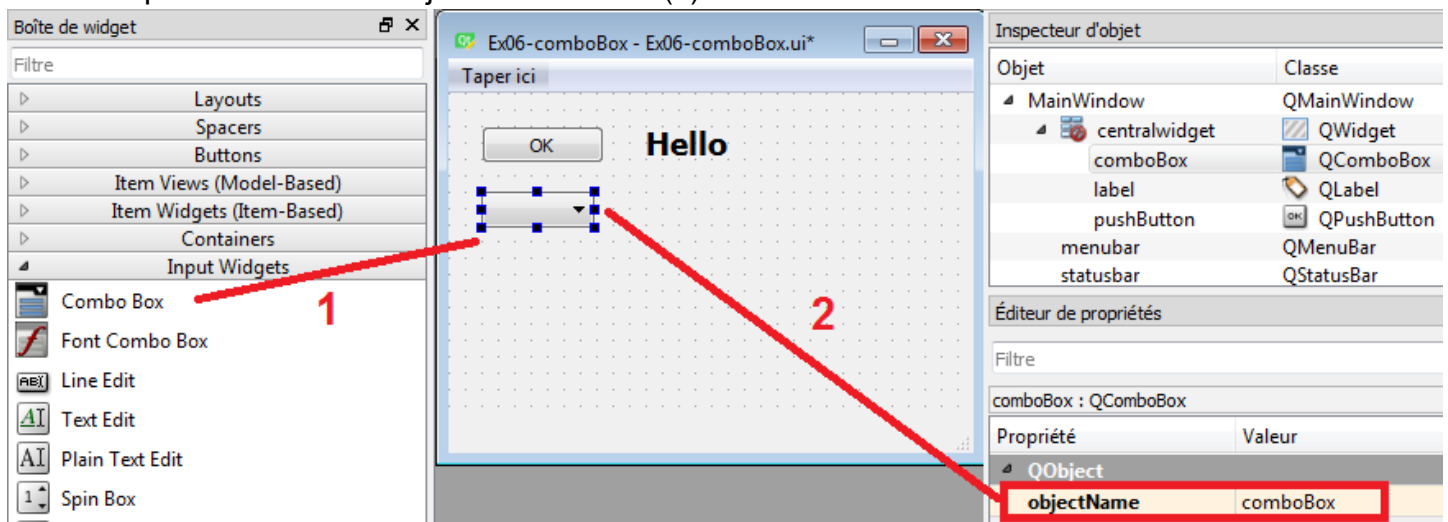
        #l'argument genre correspond à la source du signal
    def verifieRadios(self,genre):
        if genre.text() == "Homme": #le radiobutton est-il un homme ?
            if genre.isChecked() == True:
                self.label.setText("Homme") #affiche le résultat dans le label
        if genre.text() == "Femme":
            if genre.isChecked() == True:
                self.label.setText("Femme") #affiche le résultat dans le label

app = QApplication(sys.argv)
window = MainWindows()
app.exec_()
```

6 combo box (nom des fichiers créés : Ex06-comboBox.ui, Ex06-comboBox.py)

6.1 Qt Designer

- Ajouter une comboBox (1)
- Repérer le nom de l'objet : **comboBox** (2)



6.2 Code en python

```
#Ex06-comboBox
#Quand on clique sur le bouton, on affiche la ligne du comboBox choisi
#Dès que le combo box est changé, le label est rafraîchit

from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtWidgets, QtMultimedia, QtGui, uic
import sys

class MainWindows(QMainWindow):
    def __init__(self):
        super(MainWindows, self).__init__()
        uic.loadUi('Ex06-comboBox.ui', self) #chargement du formulaire XML
        self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable
        #Quand on clique sur le pushButton, on appelle la méthode boutonOK
        self.pushButton.clicked.connect(self.boutonOk)
        #ajoute des items dans le comboBox
        self.comboBox.addItem(["C", "C++", "Java", "Python"])
        #Quand on clique sur le comboBox, on appelle la méthode selectionchange
        self.comboBox.currentIndexChanged.connect(self.selectionchange)
        self.show() #affiche la fenêtre MainWindows

    def boutonOk(self):
        print('bouton cliqué')
        choix=self.comboBox.currentText()
        print("Choix de l'utilisateur",choix)

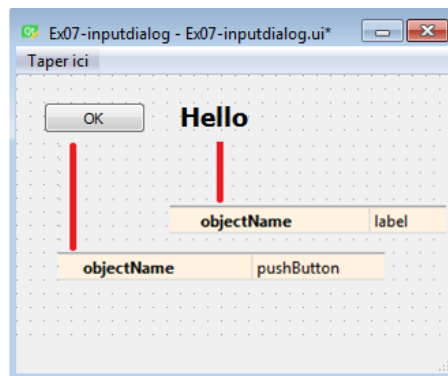
    def selectionchange(self):
        choix=self.comboBox.currentText()
        self.label.setText(choix) #affiche le résultat dans le label

app = QApplication(sys.argv)
window = MainWindows()
app.exec_()
```

7 [input dialog](#) (nom des fichiers créés : Ex07-inputdialog.ui, Ex07-inputdialog.py)

7.1 Qt Designer

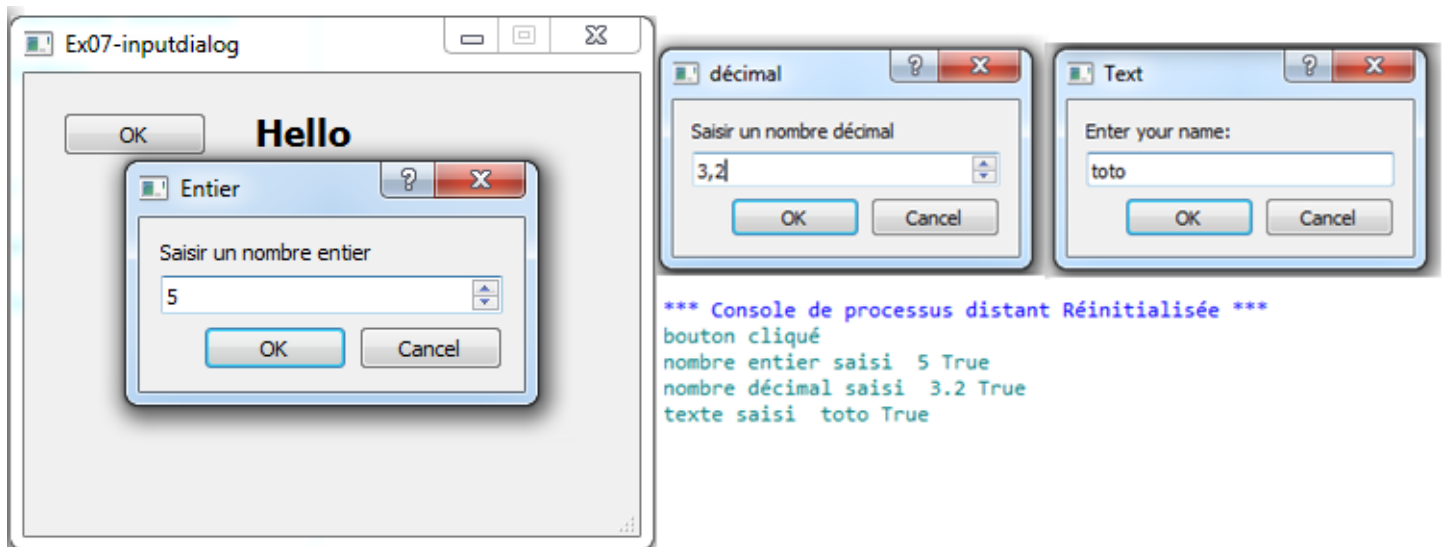
- Ajouter un bouton et un label



7.2 Code en python

```
def boutonOk(self):
    print('bouton cliqué')
    num, ok = QInputDialog.getInt(self, "Entier", "Saisir un nombre entier")
    print("nombre entier saisi ", num, ok)
    dec, ok = QInputDialog.getDouble(self, "décimal", "Saisir un nombre décimal")
    print("nombre décimal saisi ", dec, ok)
    text, ok = QInputDialog.getText(self, 'Text', 'Enter your name:')
    print("texte saisi ", text, ok)
```

Résultat :



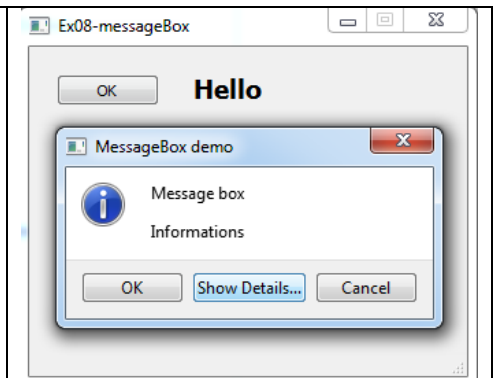
8 [message Box](#) (nom des fichiers créés : Ex08-messageBox.ui, Ex08-messageBox.py)

8.1 Qt Designer

- Ajouter un bouton et un label comme précédemment

8.2 Code en python

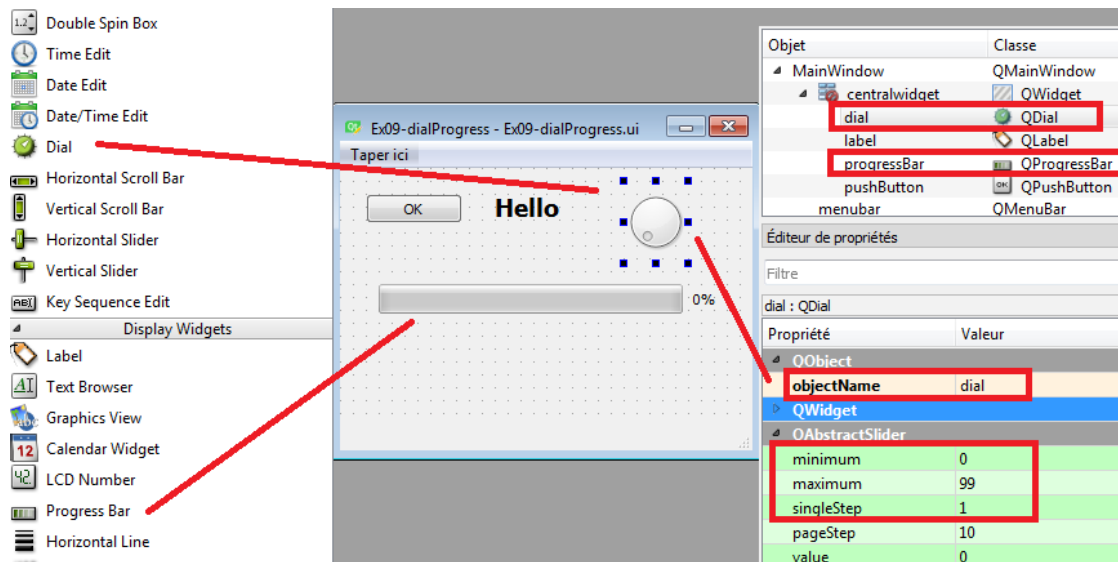
```
def boutonOk(self):
    print('bouton cliqué')
    msg = QMessageBox()
    msg.setIcon(QMessageBox.Information)
    msg.setText("Message box")
    msg.setInformativeText("Informations")
    msg.setWindowTitle("MessageBox demo")
    msg.setDetailedText("Alors c'est compris ?")
    msg.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
    retval = msg.exec_()
    print("valeur de retour:", retval)
```



9 Dial et Progress bar (nom des fichiers créés : Ex09-dialProgress.ui, Ex09-dialProgress.py)

8.1 Qt Designer

- Ajouter un bouton dial et une barre de progression
- Repérer le nom des objets : **dial** et **progressBar**
- Adapter les valeurs min et max (0 à 99 par défaut)



9.2 Code en python

```
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtWidgets, QtMultimedia, QtGui, uic
import sys

class MainWindows(QMainWindow):
    def __init__(self):
        super(MainWindows, self).__init__()
        uic.loadUi('Ex09-dialProgress.ui', self) #chargement du formulaire XML
        self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable
        #Quand on clique sur le pushButton, on appelle la méthode boutonOk
        self.pushButton.clicked.connect(self.boutonOk)
        self.dial.valueChanged.connect(self.potentiometre)
        self.pushButton.setStyleSheet("color: red")
        self.show() #affiche la fenêtre MainWindows

    def boutonOk(self):
        print('bouton cliqué')
        self.dial.setValue(50) #remets dial à 50

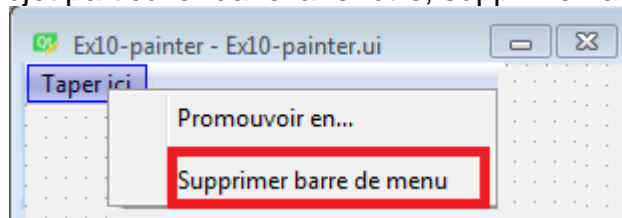
    def potentiometre(self):
        valeur=self.dial.value() #récupère la valeur du dial
        print("Valeur =",valeur)
        self.progressBar.setValue(valeur) #ajuste la barre de progression
        #changement de couleur progressive
        self.label.setStyleSheet('color: rgb({},0,0)'.format(valeur*255/100))
        #seuillage à 50 pour changer la couleur (valeur de la barre de progression)
        if valeur>50:
            self.progressBar.setStyleSheet("color: blue")
        else:
            self.progressBar.setStyleSheet("color: #00ff00")

app = QApplication(sys.argv)
window = MainWindows()
app.exec_()
```

10 Painter (nom des fichiers créés : Ex10-graphics.ui, Ex10-graphics.py)

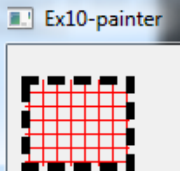
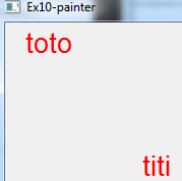
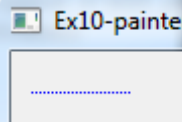
10.1 Qt Designer

- Sans aucun objet particulier dans la fenêtre, supprimer la barre de menu



10.2 Code en python

<pre>#tous les exemples suivants seront mis dans la méthode paintEvent def paintEvent(self, event): painter = QPainter(self)</pre>	<pre>class MainWindows(QMainWindow): def __init__(self): super(MainWindows, self).__init__() uic.loadUi('Ex10-painter.ui', self) #chargement du formulaire XML self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable self.show() #affiche la fenêtre MainWindows def paintEvent(self, event): painter = QPainter(self) #Affichage d'une image pic = QPixmap("logo.png") #chargement painter.drawPixmap(10,10, pic) #affichage aux coordonnées</pre>
<pre>#Affichage d'une image pic = QPixmap("logo.png") #chargement painter.drawPixmap(10,10, pic) #affichage aux coordonnées</pre>	
<pre>#tracé d'un rectangle painter.setPen(QPen(Qt.red, 5)) #definir un crayon painter.setBrush(Qt.green) # definir un remplissage painter.drawRect(0, 0, 100, 100) #tracé</pre>	
<pre>#tracé d'une ellipse painter.setPen(QPen(Qt.blue, 3)) #definir un crayon painter.setBrush(QColor(255, 255, 0)) #couleur de remplissage RGB painter.drawEllipse(50,30,80,40) #tracé</pre>	
<pre>#tracé de polygone points = [#definir les 4 points du polygone QPoint(10,10), QPoint(10,100), QPoint(100,10), QPoint(100,100)] painter.setPen(QPen(Qt.red, 5)) #crayon rouge largeur 5 painter.setBrush(Qt.green) # couleur du remplissage poly = QPolygon(points) #instancier un polygone a partir de points painter.drawPolygon(poly) #tracé</pre>	
<pre>#tracé de rectangle avec gradient painter.setPen(QPen(Qt.black, 5, Qt.SolidLine)) grad = QLinearGradient(10, 10, 50, 60) painter.setBrush(QBrush(grad)) painter.drawRect(10, 10, 50, 60)</pre>	

<pre>#tracé de rectangle contour en pointillés painter.setPen(QPen(Qt.black, 5, Qt.DotLine)) #crayon noir largeur 5 en pointillés painter.setBrush(QBrush(Qt.red, Qt.CrossPattern)) #remplissage rouge painter.drawRect(10, 20, 60, 50) #tracé</pre>	
<pre>#Affichage de texte painter.setPen(QColor(255, 0, 0)) #couleur du crayon painter.setFont(QFont('Arial', 20)) #police painter.drawText(self.rect(), Qt.AlignCenter, "titi") #affichage au centre de la fenetre painter.drawText(20,30, "toto") #affichage aux coordonnées</pre>	
<pre>#tracé de points painter.setPen(QColor(0, 0, 255)) #couleur du crayon for x in range(0,50,2): painter.drawPoint(x+10, 20)</pre>	

11 Détecter la souris et le clavier (nom des fichiers créés : Ex11-dessin.ui, Ex11-dessin.py)

11.1 Qt Designer

- Sans aucun objet particulier dans la fenêtre, supprimer la barre de menu (identique à l'exemple 10)

11.2 Code en python

```
#Ex11-dessin
#tracer un point avec la souris
#détecter une touche au clavier

from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtWidgets, QtMultimedia, QtGui, uic
import sys

class MainWindows(QMainWindow):
    def __init__(self):
        super(MainWindows, self).init()
        uic.loadUi('Ex11-dessin.ui', self) #chargement du formulaire XML
        self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable
        self.show() #affiche la fenêtre MainWindows

        self.position = QPoint(0,0)

    def keyPressEvent(self, event): #une touche a été pressée
        key = event.key()
        if key == Qt.Key_M: #touche m ou M
            print("touche 'm' pressée")
        elif key == Qt.Key_Right: #touche fleche droite
            print("touche droite pressée")

    def mousePressEvent(self, event): #la souris est pressée
        self.position = QPoint(event.x(), event.y()) #récupère la position de la souris
        print(event.x(), event.y()) #affiche les coordonnées
        self.update() #déclenche paintEvent

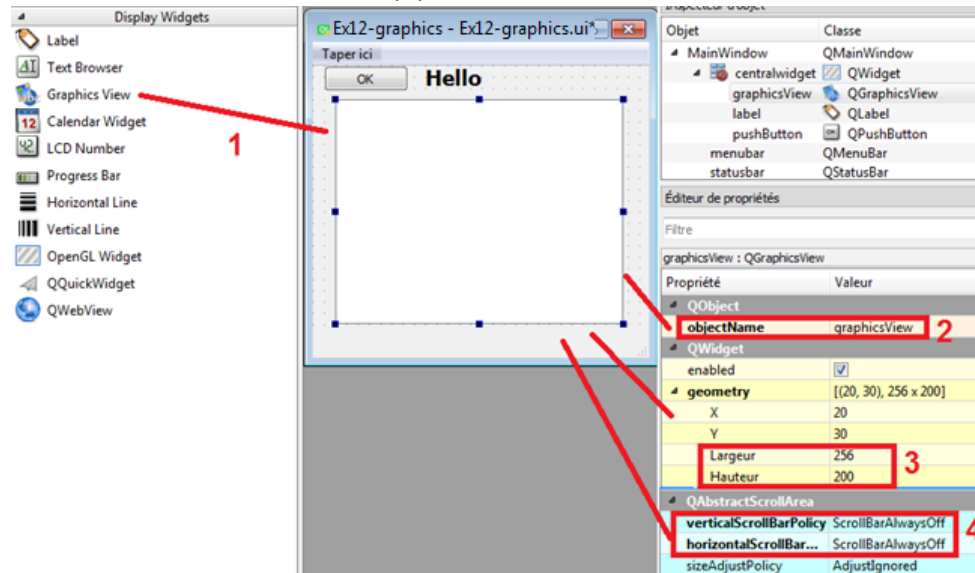
    def paintEvent(self, event):
        painter = QPainter(self)
        painter.setPen(QPen(Qt.blue, 5)) #definir un crayon
        painter.drawPoint(self.position) #afficher un point aux coordonnées position

app = QApplication(sys.argv)
window = MainWindows()
app.exec_()
```

12 Graphisme (nom des fichiers créés : Ex12-graphics.ui, Ex12-graphics.py)

12.1 Qt Designer

- Ajouter une vue graphique (1)
- Repérer le nom de l'objet : **graphicsView** (2)
- Configurer une dimension de 256 x 200 (3)
- Désactiver les barres de défilement (4)



12.2 Code en python

```
#Ex12-graphics
#tracé de formes géométriques

from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtWidgets, QtMultimedia, QtGui, uic
import sys

class MainWindows(QMainWindow):
    def __init__(self):
        super(MainWindows, self).__init__()
        uic.loadUi('Ex12-graphics.ui', self) #chargement du formulaire XML
        self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable
        self.scene = QGraphicsScene(self) #instancie un scene
        self.brush= QBrush(Qt.lightGray) #couleur de remplissage
        self.pen = QPen() #crayon
        #implante la scene dans la vue graphique
        self.scene.setSceneRect(0, 0, self.graphicsView.width(), self.graphicsView.height())
        #couleur de fond
        self.scene.setBackgroundBrush(self.brush)
        self.graphicsView.setScene(self.scene)
        #Quand on clique sur le pushButton, on appelle la méthode boutonOk
        self.pushButton.clicked.connect(self.boutonOk)
        self.show() #affiche la fenêtre MainWindows

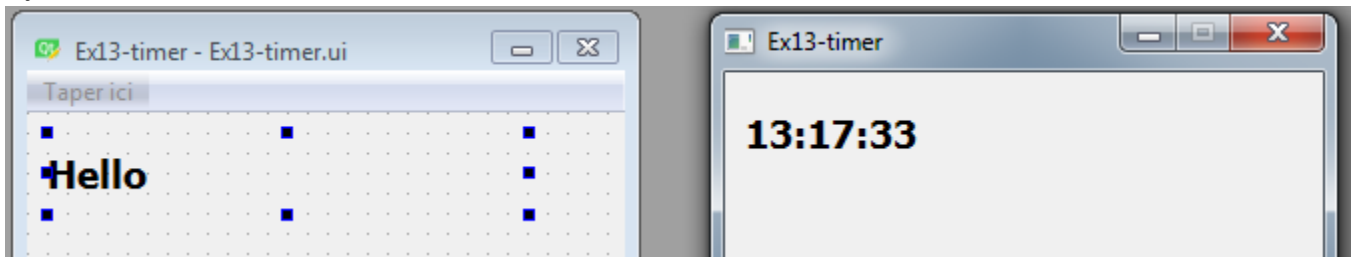
    def boutonOk(self):
        print('bouton cliqué')
        self.pen.setWidth(2)
        self.brush.setColor(Qt.red)
        self.scene.addLine(0,0,50,200,self.pen)
        self.scene.addRect(30,30, 50,50,self.pen,self.brush)
        #lst=self.scene.items()
        #self.scene.removeItem(lst[1])
        self.font=QFont("Fixed",20)
        self.texte = self.scene.addText("Mon texte",self.font)
        self.texte.setDefaultTextColor(Qt.blue)
        self.texte.setPos(50,100)

app = QApplication(sys.argv)
window = MainWindows()
app.exec_()
```

13 Timer (nom des fichiers créés : Ex13-timer.ui, Ex13-timer.py)

13.1 Qt Designer

- Ajouter un label



13.2 Code en python

```
#Ex13-timer
#Affiche l'heure dans un label toutes les secondes

from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtWidgets, QtMultimedia, QtGui, uic
import sys
from datetime import datetime #import de la bibliothèque datetime

class MainWindows(QMainWindow):
    def __init__(self):
        super(MainWindows, self).__init__()
        uic.loadUi('Ex13-timer.ui', self) #chargement du formulaire XML
        self.setFixedSize(self.size()) #la fenêtre principale n'est pas modifiable
        self.show() #affiche la fenêtre MainWindows

        self.cpt=0 #initialise un compteur
        self.timer = QTimer(self) #déclaration du timer
        self.timer.timeout.connect(self.miseAJour) #déclenche la méthode mise à jour
        self.timer.start(1000) #toutes les secondes (1000ms)

    def miseAJour(self):
        self.cpt+=1 #incrémente le compteur
        print(self.cpt) #affiche le compteur dans la console
        now = datetime.now() #récupère la date et l'heure du PC
        heure = now.strftime("%H:%M:%S") #formatage de l'heure
        self.label.setText(heure) #on change le nom du label avec l'heure

app = QApplication(sys.argv)
window = MainWindows()
app.exec_()
```