12/09/2018

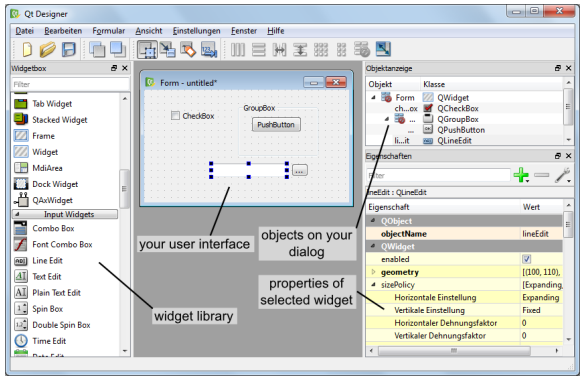
Python pour l'ingénieur

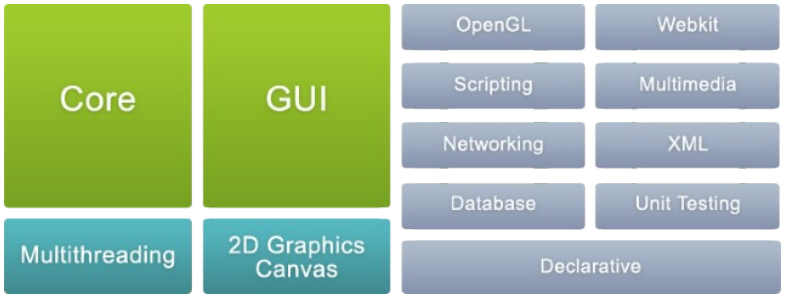
IHM avec PyQt

Module de formation

Introduction

- **Qt** est une bibliothèque de classes offrant entre autres des composants d'interface graphique appelés **widgets**.
- Qt est **multi-plateformes** (portable) et **open-source** (licence GNU LGPL permettant son utilisation légale et gratuite par des logiciels propriétaires).
- Qt est initialement écrit en langage C++.
- **PyQt** est un *binding* de Qt pour le langage Python.
  - PyQt n'est pas gratuit pour une utilisation commerciale
  - Autre binding Python de Qt : PySide
- Alternatives à Qt :
  - Python : Tk (intégré au langage), wxWidgets (wxPython)
  - C++ : Gtk, wxWidgets, MFC (Microsoft), etc.



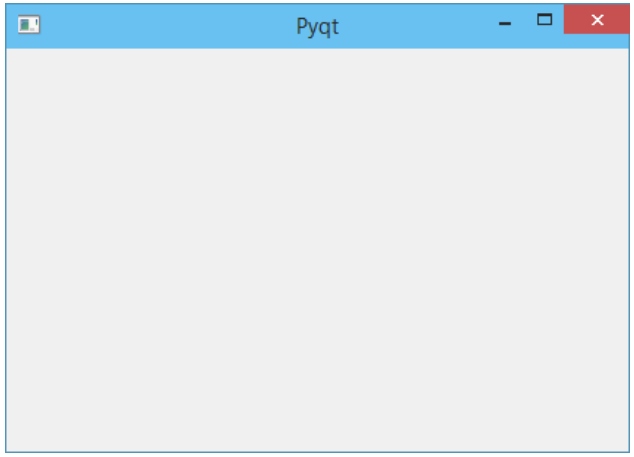


Ressources

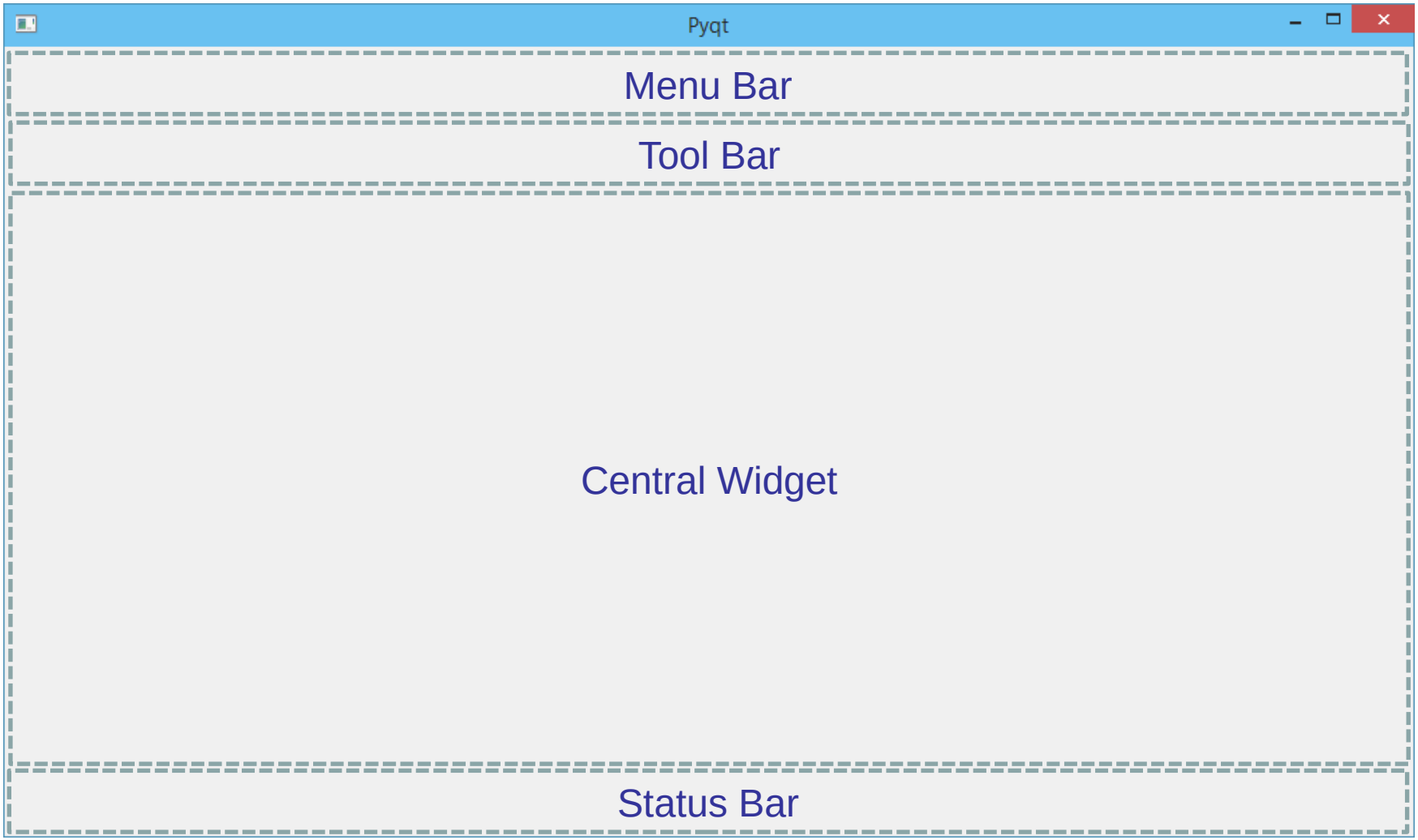
- Version de Qt utilisée pour le cours : **4.8**
- Site internet de Pyqt :
  - http://www.riverbankcomputing.com/software/pyqt/intro
  - Documentation : http://pyqt.sourceforge.net/Docs/PyQt4/classes.html
- Site internet de Qt :
  - http://qt-project.org/
  - Documentation : http://qt-project.org/doc/qt-4.8/
- *La documentation de PyQt n'est pas aussi complète que celle de Qt → il faut parfois se référer à celle de Qt (en C++, mais la transcription en Python est assez facile).*
- *Qt et PyQt existent aussi en version 5. Le code de ce cours fonctionne avec PyQt4 ou PyQt5.*

Application minimale

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')

def main():
    app = QApplication(sys.argv)
    fenetre = MaFenetre()
    fenetre.show()
    app.exec()

if __name__ == '__main__':
    main()
```



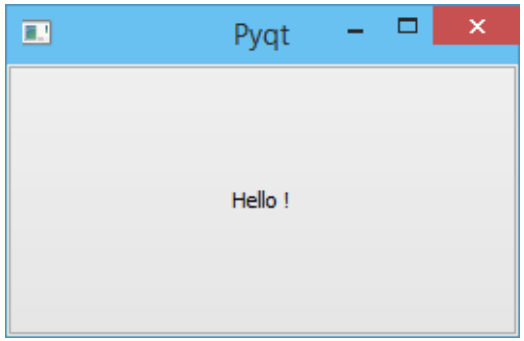
Layout de QMainWindow



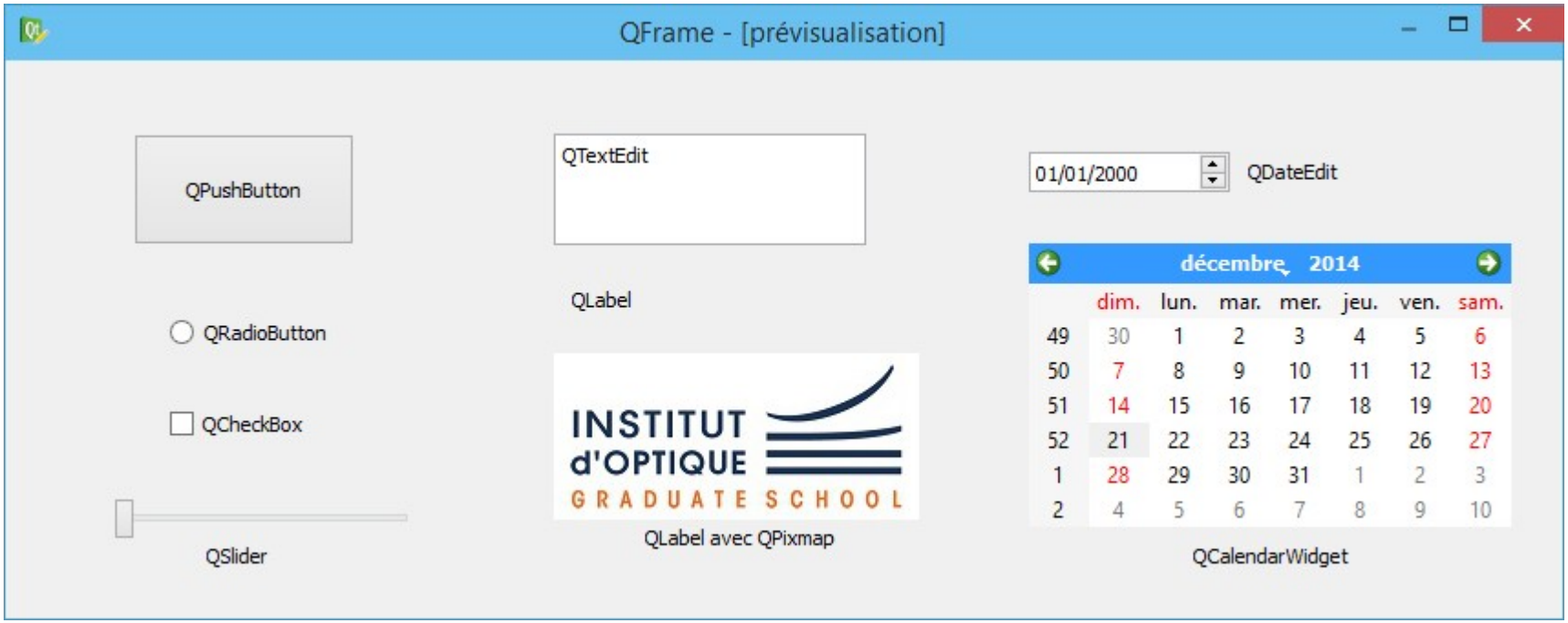
Central Widget

- Central Widget = un objet dérivant de QWidget
  - Soit un widget prédéfini
  - Soit un widget personnalisé, défini par une classe dérivant de QWidget

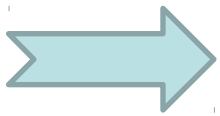
```
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')

    button = QPushButton('Hello !')

    self.setCentralWidget(button)
```

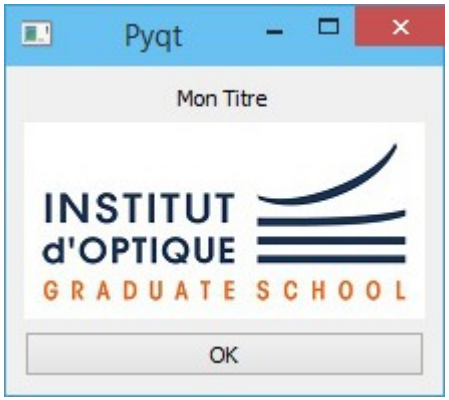


Widgets prédéfinis



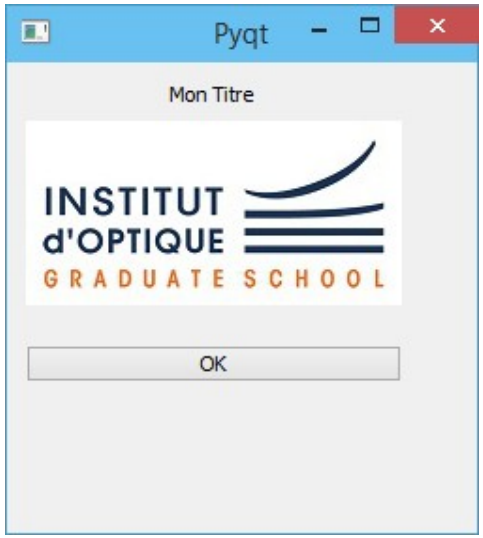

```
label = QLabel('Mon Titre', self)
label.setAlignment(Qt.AlignHCenter)
image = QLabel(self)
image.setPixmap(QPixmap('logo.jpg'))
bouton = QPushButton('OK', self)
```





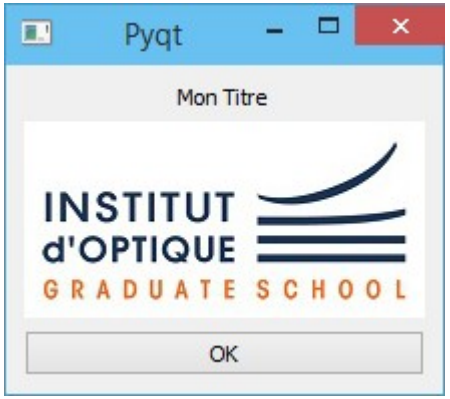
Widget personnalisé

```
class MonWidget(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        label = QLabel('Mon Titre', self)
        label.setAlignment(Qt.AlignHCenter)
        label.setGeometry(10, 10, 200, 20)
        image = QLabel(self)
        image.setPixmap(QPixmap('logo.jpg'))
        image.setGeometry(10, 30, 200, 100)
        bouton = QPushButton('OK', self)
        bouton.setGeometry(10, 150, 200, 20)

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')
        central_widget = MonWidget(self)
        self.setCentralWidget(central_widget)
        self.resize(250, 250)
```

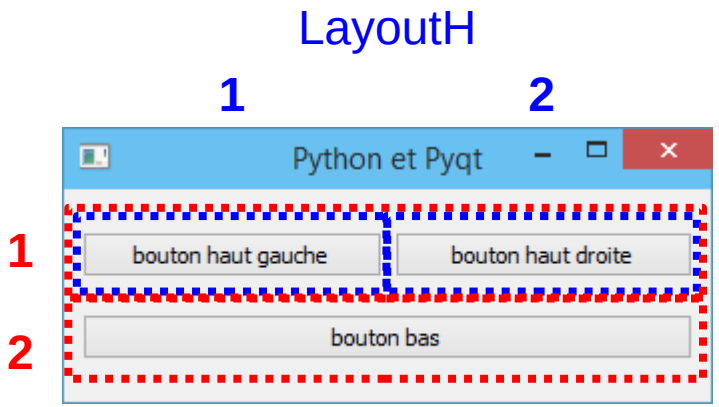


Placement avec Layout

```
class MonWidget(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        layout = QVBoxLayout()
        label = QLabel('Mon Titre')
        label.setAlignment(Qt.AlignHCenter)
        image = QLabel()
        image.setPixmap(QPixmap('logo.jpg'))
        bouton = QPushButton('OK')
        layout.addWidget(label)
        layout.addWidget(image)
        layout.addWidget(bouton)
        self.setLayout(layout)

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')
        central_widget = MonWidget(self)
        self.setCentralWidget(central_widget)
```



Imbrication des Layouts

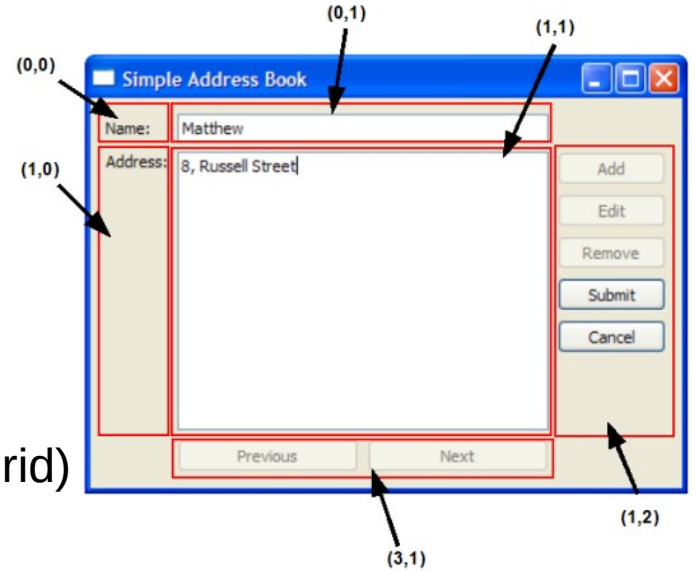



```
class MonWidget(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        bouton1 = QPushButton('bouton haut gauche')
        bouton2 = QPushButton('bouton haut droite')
        layoutH = QHBoxLayout()
        layoutH.addWidget(bouton1)
        layoutH.addWidget(bouton2)
        bouton3 = QPushButton('bouton bas')
        layoutV = QVBoxLayout()
        layoutV.addLayout(layoutH)
        layoutV.addWidget(bouton3)
        self.setLayout(layoutV)
```

Types de Layout

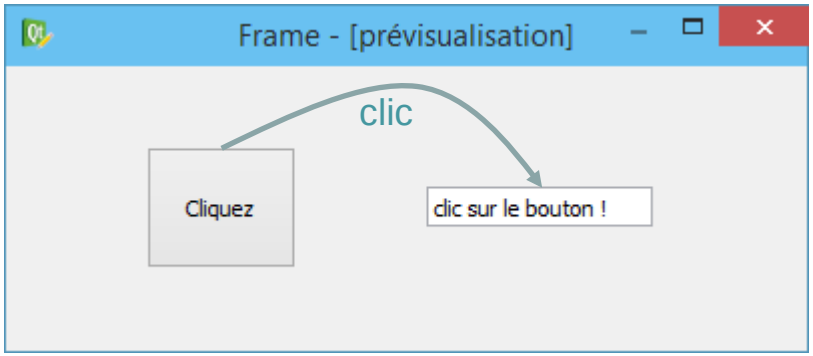
- QHBoxLayout
- QVBoxLayout
- QGridLayout
- QFormLayout

(grid)



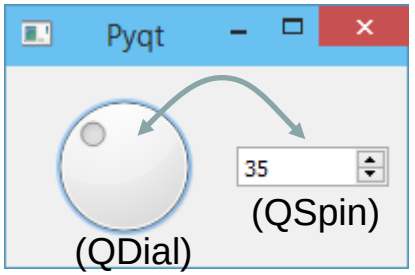
(form)

Slot et Signal




```
class MonWidget(QWidget):

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

        self.bouton = QPushButton('Cliquez', self)
        self.texte = QLineEdit(self)

        self.bouton.clicked.connect(self.on_bouton)
    …

    def on_bouton(self):
        self.texte.setText('clic sur le bouton !')
```

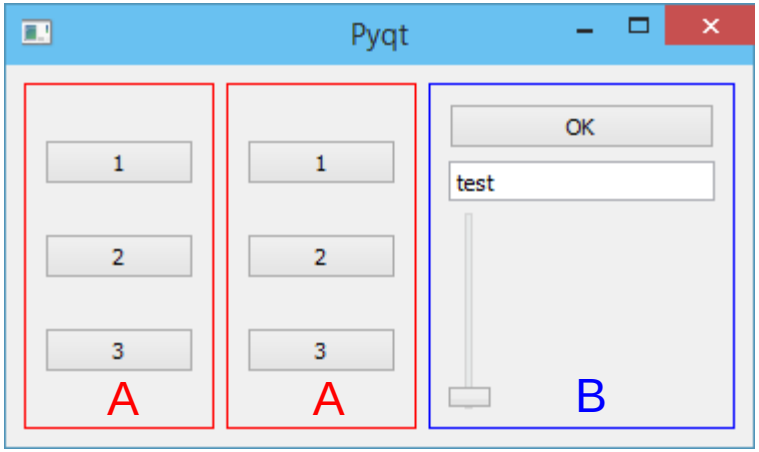
Slot et Signal




```
dial = QDial(self)
spin = QSpinBox(self)

spin.valueChanged.connect(dial.setValue)
dial.valueChanged.connect(spin.setValue)
```

Fenêtre plus complexe

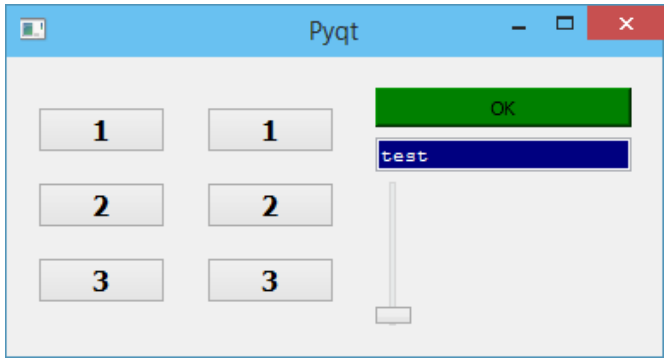
```
class MonWidgetA(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        layout = QVBoxLayout()
        layout.addWidget(QPushButton('1'))
        layout.addWidget(QPushButton('2'))
        layout.addWidget(QPushButton('3'))
        self.setLayout(layout)

class MonWidgetB(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        layout = QVBoxLayout()
        layout.addWidget(QPushButton('OK'))
        layout.addWidget(QLineEdit('test'))
        layout.addWidget(QSlider())
        self.setLayout(layout)
```




```
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        widget1 = MonWidgetA(self)
        widget2 = MonWidgetA(self)
        widget3 = MonWidgetB(self)
        layout = QHBoxLayout()
        layout.addWidget(widget1)
        layout.addWidget(widget2)
        layout.addWidget(widget3)
        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
```

Feuilles de style

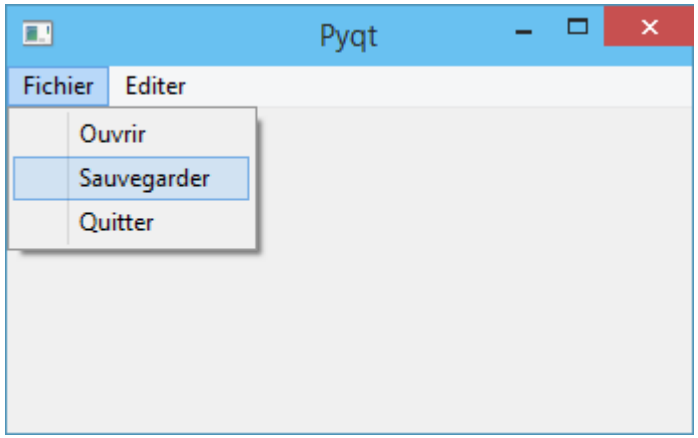
```
class MonWidgetA(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        self.setStyleSheet('QPushButton { font-weight: bold; font-size: 16px; }')
        layout = QVBoxLayout()
        layout.addWidget(QPushButton('1'))
        layout.addWidget(QPushButton('2'))
        layout.addWidget(QPushButton('3'))
        self.setLayout(layout)
```




```
class MonWidgetB(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        self.setStyleSheet(' \
            QLineEdit { background-color: rgb(0,0,128); color: white; font-family: courier; } \
            QPushButton { background-color: rgb(0,128,0); }')
        layout = QVBoxLayout()
        layout.addWidget(QPushButton('OK'))
        layout.addWidget(QLineEdit('test'))
        layout.addWidget(QSlider())
        self.setLayout(layout)
```

 CSS3

Menu déroulant



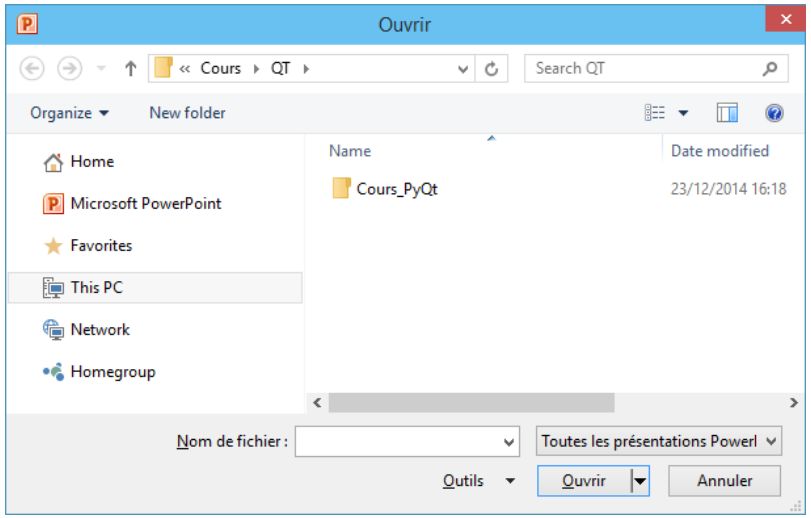

```
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()

        menu1 = self.menuBar().addMenu('Fichier')

        action1 = QAction('Ouvrir', self)
        action1.triggered.connect(self.on_ouvrir)
        menu1.addAction(action1)

        action2 = QAction('Sauvegarder', self)
        action2.triggered.connect(self.on_sauver)
        menu1.addAction(action2)

        action3 = QAction('Quitter', self)
        action3.triggered.connect(self.on_quitter)
        menu1.addAction(action3)

        menu2 = self.menuBar().addMenu('Editer')
```

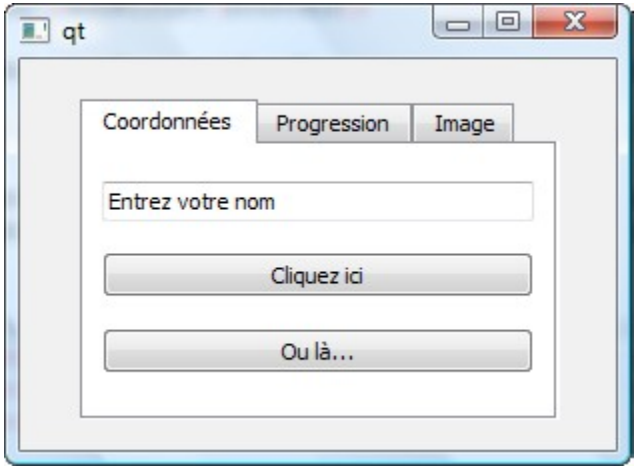
méthode de classe à définir

Voir aussi : QAction.setShorcut(), QAction.setIcon(), QAction.setCheckable(), sous-menus

Quelques autres widgets de Qt

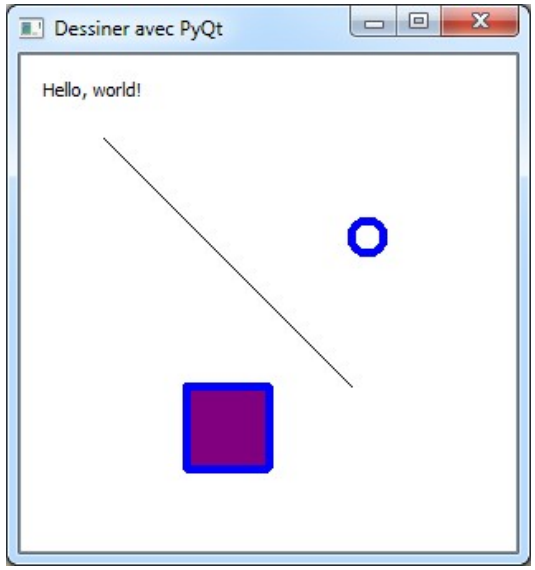
- Boites de dialogues usuelles
  - QMessageBox
  - QInputDialog
  - QFileDialog
- Widgets conteneurs
  - QGroupBox
  - QTabWidget
  - QStackedLayout
- Toolbar et Statusbar
  - QToolBar
  - QStatusBar
- Widgets complexes
  - QListView
  - QTreeView
  - QTableView





Dessiner

```
class MaScene(QGraphicsScene):
    """cette classe décrit la scène"""
    def __init__(self, parent):
        super().__init__(parent)
        self.setSceneRect(0, 0, 300, 300)
        texte = self.addText("Hello, world!")
        texte.setPos(10, 10)
        self.addLine(50, 50, 200, 200)
        stylo = QPen(Qt.blue, 5, Qt.SolidLine)
        self.addEllipse(200, 100, 20, 20, stylo)
        brosse = QBrush(QColor(128, 0, 128), Qt.SolidPattern)
        self.addRect(100, 200, 50, 50, stylo, brosse)

class MaVueGraphique(QGraphicsView):
    """cette classe fait le rendu (= dessin) de la scène"""
    def __init__(self, parent):
        super().__init__(parent)
        scene = MaScene(self)
        self.setScene(scene)

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Dessiner avec PyQt")
        vue = MaVueGraphique(self)
        self.setCentralWidget(vue)
```



Clavier et Souris

- Une classe dérivée de QWidget peut redéfinir des fonctions héritées pour réagir aux événements clavier et souris, par exemple :

- keyPressEvent()
  - appelée lorsque qu'une touche du clavier est pressée.

```
def keyPressEvent(self, keyevent):
    if keyevent.key() == Qt.Key_Q:
        ...
```

- mousePressEvent()
  - appelée lorsqu'un bouton de la souris est cliqué.

```
def mousePressEvent(self, mouseevent):
    self.x = mouseevent.scenePos().x()
    self.y = mouseevent.scenePos().y()
```

Timer

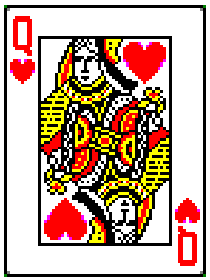
- Un *timer* permet de déclencher l'appel d'une fonction à intervalles réguliers.
- La classe Qt pour créer un timer est Qtimer

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

def afficher():
    print("bonjour")

app = QApplication(sys.argv)
timer = QTimer()
timer.timeout.connect(afficher)
# répétition toutes les 1000 millisecondes
timer.start(1000)
app.exec()
```

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

class MonTimer(QTimer):
    def __init__(self):
        super().__init__()
        self.timeout.connect(self.ontimer)

    def ontimer(self):
        print("bonjour")

app = QApplication(sys.argv)
timer = MonTimer()
# répétition toutes les 1000 millisecondes
timer.start(1000)
app.exec()
```

`timer.stop()` permet d'arrêter le timer

Exercice



- 1re partie
  - Créer une classe dessinant 2 cartes.

```
addPixmap(QPixmap("carte.png"))
```

  - La classe contient une fonction permettant de repositionner ces 2 cartes à l'endroit où on clique.
  - La classe contient une fonction permettant de créer une 3e carte quand on appuie sur la touche 'c'.
  - La classe contient une fonction permettant de déplacer la 3e carte quand on appuie sur les flèches du clavier.
- 2e partie
  - La classe contient un timer permettant d'animer les 2 premières cartes, en modifiant leurs positions de quelques pixels à intervalles réguliers.
  - La classe contient une fonction permettant de mettre l'animation en pause quand on appuie sur la touche 'p'.