

# Des interfaces graphiques en Python à l'aide du PyQt5

Réaliser par:

Abderrahim AMANAR

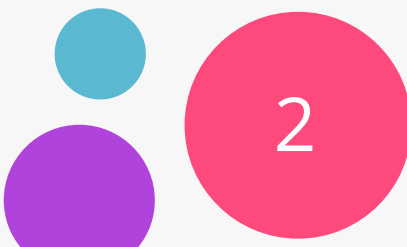
Abdessamad AHADAD



Encadre par: Mme. GUEROUATE

# Table des matières

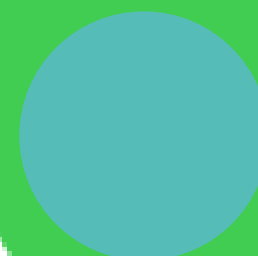
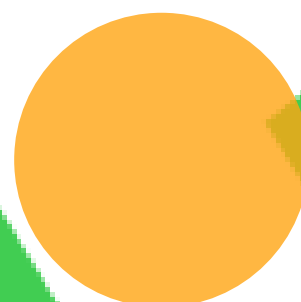
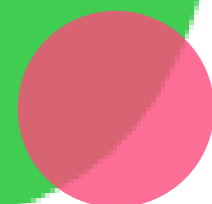
- 1 Introduction
- 2 Hello World
- 3 Classes majeures
- 4 Signals and Slots
- 5 Mise en page (Layout), Widget
- 6 Qt Designer



1



# INTRODUCTION





# Bienvenue à notre **présentation**




## Commençons maintenant!

**PyQt** est une boîte à outils de widgets GUI. Il s'agit d'une interface Python pour **Qt**, l'une des bibliothèques d'interface graphique les plus puissantes et les plus populaires. PyQt est un mélange de langage de programmation Python et de la bibliothèque **Qt**.

**PyQt** API est un ensemble de modules contenant un grand nombre de classes et de fonctions. Alors que le module **QtCore** contient des fonctionnalités non-GUI pour travailler avec le fichier et le répertoire etc., le module **QtGui** contient tous les contrôles graphiques. En outre, il existe des modules pour travailler avec XML (**QtXml**), SVG (**QtSvg**) et SQL (**QtSql**), etc.

# Environnements pris en charge



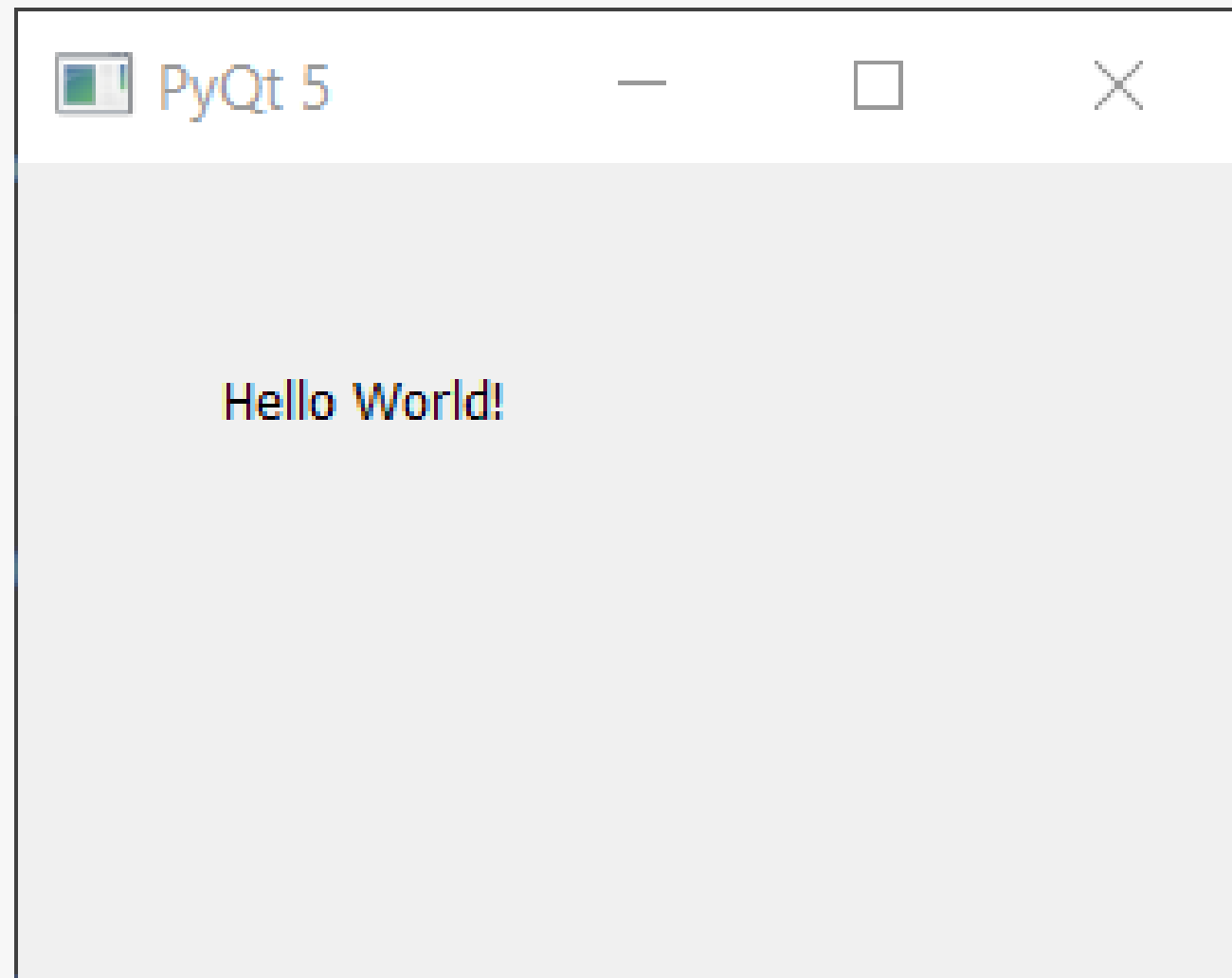
**PyQt** est compatible avec tous les systèmes d'exploitation populaires, y compris Windows, Linux et Mac OS.

## **Installation:**

- > pip3 install SIP
- > pip3 install PyQt5
- > pip3 install pyqt5-tools



# Hello World

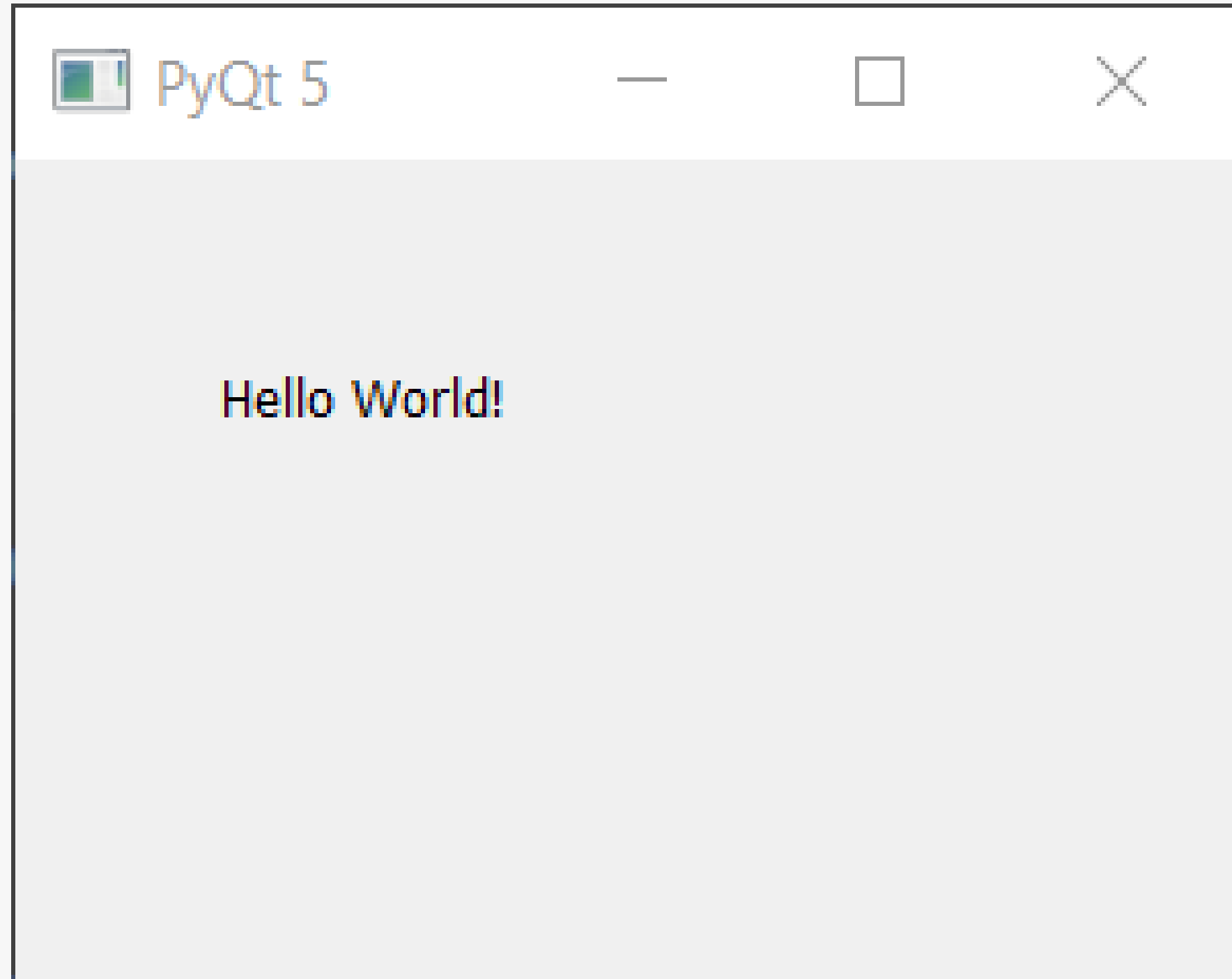


```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel

def window():
    app = QApplication(sys.argv)
    w = QWidget()
    b = QLabel(w)
    b.setText("Hello World!")
    w.setGeometry(100, 100, 300, 400)
    b.move(50, 50)
    w.setWindowTitle("PyQt 5")
    w.show()
    sys.exit(app.exec_())

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

# Exécution de code






# Classes majeures



## QObject

L'API PyQt contient plus de 400 classes. La classe QObject est en haut de la hiérarchie de classes. C'est la classe de base de tous les objets Qt. De plus, la classe QPaintDevice est la classe de base pour tous les objets pouvant être peints.



## QApplication

La classe QApplication gère les principaux paramètres et le flux de contrôle d'une application graphique. Il contient une boucle d'événement principal à l'intérieur de laquelle les événements générés par les éléments de fenêtre.

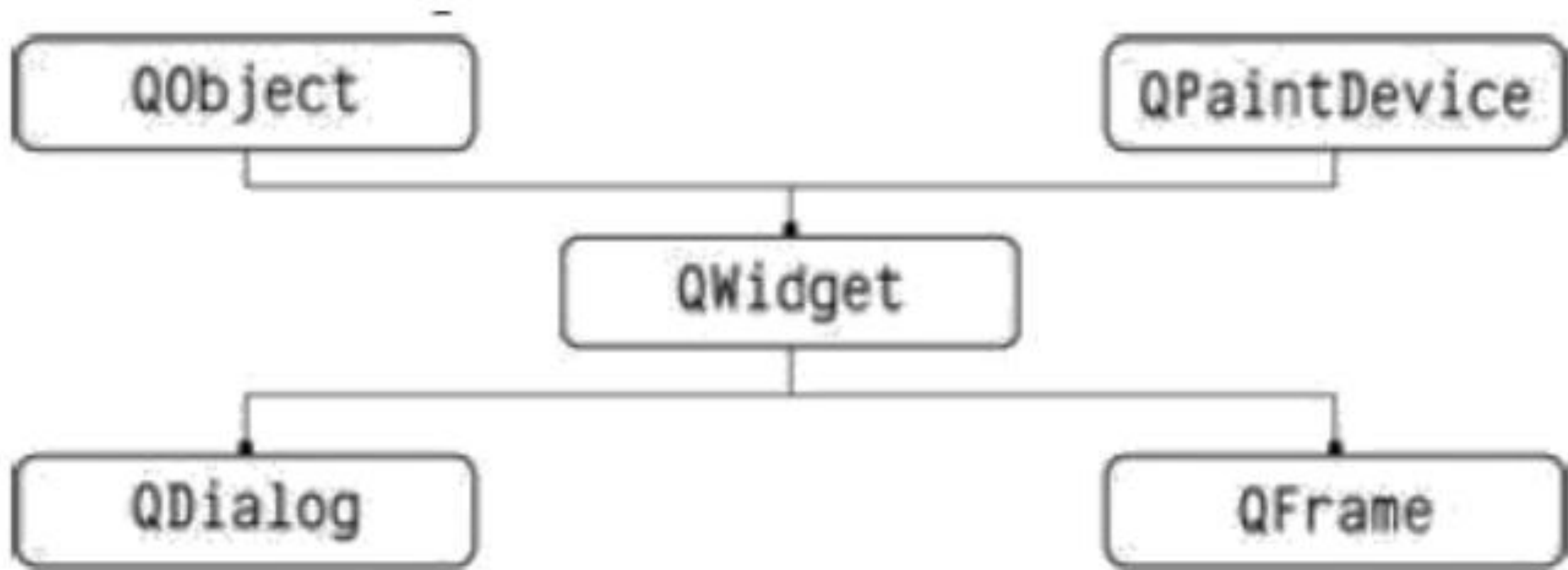


## QWidget

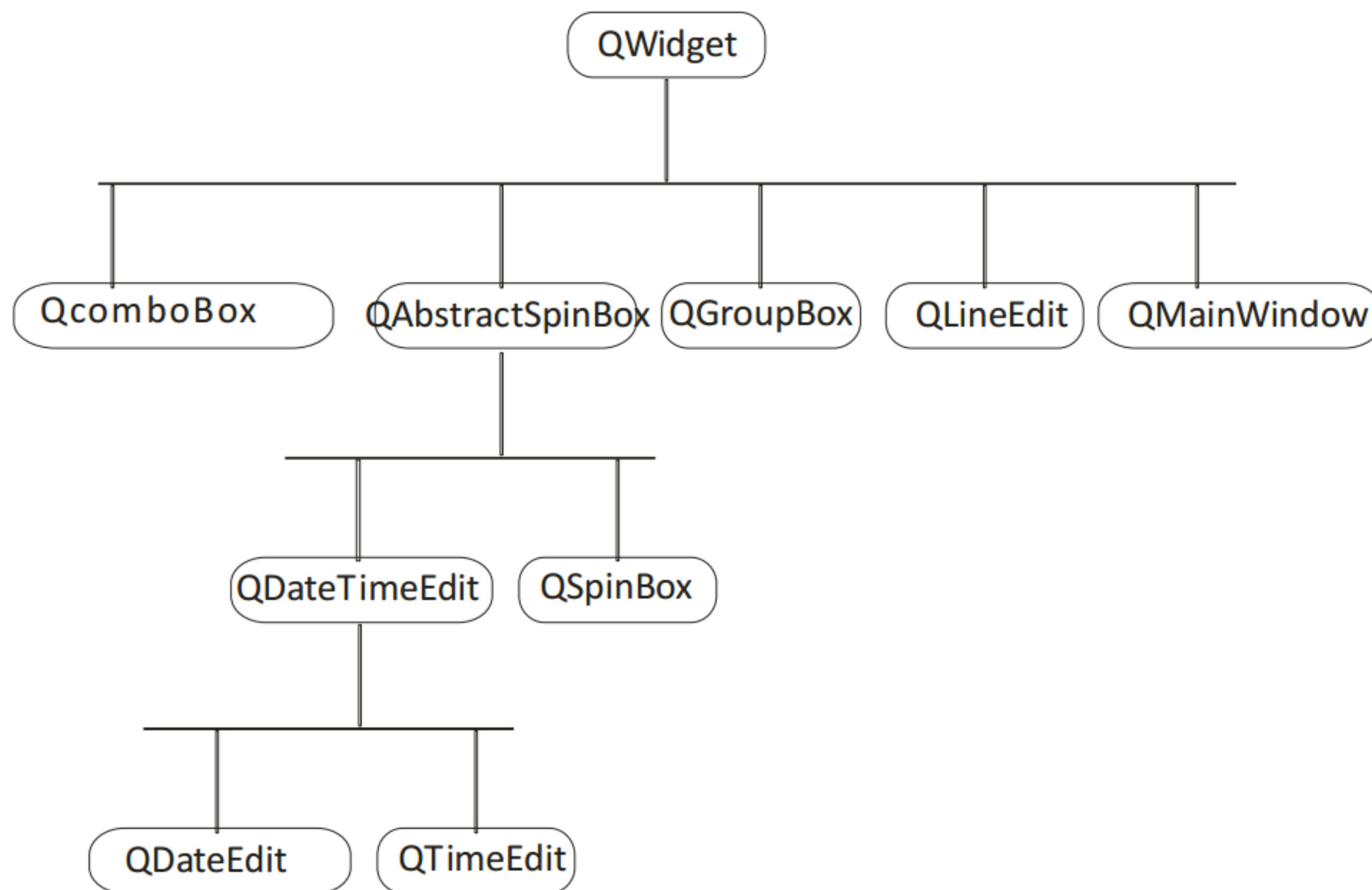
La classe QWidget, dérivée des classes QObject et QPaintDevice est la classe de base pour tous les objets de l'interface utilisateur. Les classes QDialog et QFrame sont également dérivées de la classe QWidget.



Les diagrammes suivants représentent certaines classes importantes dans leur hiérarchie.



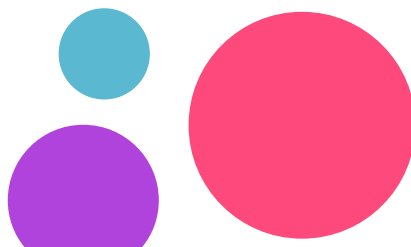
Les diagrammes suivants représentent certaines classes importantes dans leur hiérarchie.





# Widgets fréquemment utilisés:

Widgets	Description
QLabel	Utilisé pour afficher du texte ou une image
QLineEdit	Permet à l'utilisateur d'entrer une ligne de texte
QTextEdit	Permet à l'utilisateur d'entrer du texte multiligne
QPushButton	Un bouton de commande pour appeler l'action
QRadioButton	Permet d'en choisir un parmi plusieurs options
QCheckBox	Permet le choix de plus d'une option
QComboBox	Fournit une liste déroulante d'éléments à sélectionner

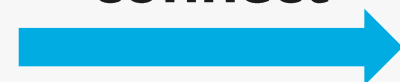


# Signals and Slots

## Signals

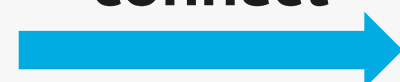
addButton : clicked()

connect



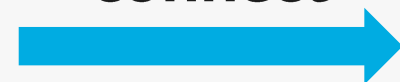
submitButton : clicked()

connect



cancelButton : clicked()

connect



## Slots

AddressBook : addContact()

AddressBook : submitContact()

AddressBook : cancel()

```
QtCore.QObject.connect(button, QtCore.SIGNAL("clicked()"), slot_function)  
btn.clicked.connect(slot_function)
```

# Gestionnaires de mise en page

**PyQt** API fournit des classes de disposition pour une gestion plus élégante du positionnement des widgets à l'intérieur du conteneur.

Les avantages des gestionnaires de mise en page par rapport au positionnement absolu sont:

- Les widgets à l'intérieur de la fenêtre sont automatiquement redimensionnés.
- Assure une apparence uniforme sur les périphériques d'affichage avec différentes résolutions
- L'ajout ou la suppression dynamique d'un widget est possible sans devoir procéder à une nouvelle conception.

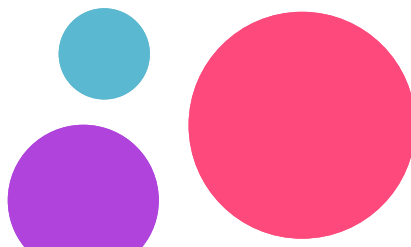
La classe **QLayout** est la classe de base à partir de laquelle les classes **QBoxLayout**, **QGridLayout** et **QFormLayout** sont dérivées.



# QBoxLayout Class

- La classe **QBoxLayout** aligne les widgets verticalement ou horizontalement. Ses classes dérivées sont **QVBoxLayout** (pour agencer les widgets verticalement) et **QHBoxLayout** (pour agencer les widgets horizontalement). Le tableau suivant montre les méthodes importantes de la classe **QBoxLayout**:

Widgets	Description
<code>addWidget()</code>	Ajouter un widget à la BoxLayout
<code>addStretch()</code>	Crée une boîte extensible vide
<code>addLayout()</code>	Ajouter une autre mise en page imbriquée

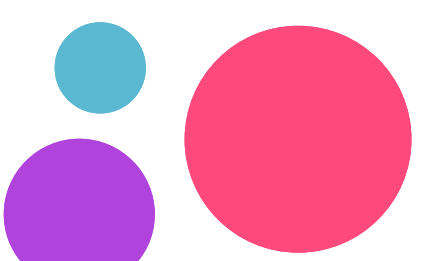




# QGridLayout Class

- Un objet de classe **GridLayout** présente une grille de cellules disposées en lignes et en colonnes. La classe contient la méthode **addWidget ()**. Tout widget peut être ajouté en spécifiant le nombre de lignes et de colonnes de la cellule. En option, un facteur d'étendue pour la ligne et la colonne, si spécifié, rend le widget plus large ou plus grand qu'une cellule. Deux surcharges de la méthode **addWidget ()** sont les suivantes:

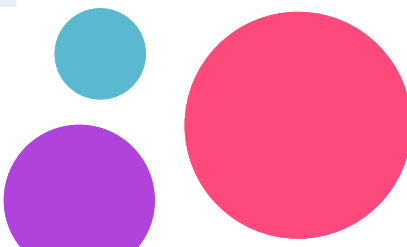
Widgets	Description
<code>addWidget(QWidget, int r, int c)</code>	Ajoute un widget à la ligne et à la colonne spécifiées
<code>addWidget(QWidget, int r, int c, int rowspan, int colspan)</code>	Ajoute un widget à la ligne et à la colonne spécifiées et ayant une largeur et / ou une hauteur spécifiée
<code>addLayout(QLayout, int r, int c)</code>	Ajoute un objet physique à la ligne et à la colonne spécifiées









# QPushButton Widget

Widgets	Description
<code>setCheckable()</code>	Reconnaît les états pressés et relâchés du bouton si défini sur true
<code>toggle()</code>	Bascule entre les états vérifiables
<code>setIcon()</code>	Affiche une icône formée de pixmap d'un fichier image
<code>setEnabled()</code>	Lorsque ce paramètre est défini sur false, le bouton est désactivé, ce qui signifie qu'il n'est pas associé à un signal.
<code>setText()</code>	Définit par programme la légende des boutons
<code>text()</code>	Récupère la légende des boutons





# QMessageBox

Widgets	Description
setIcon()	Affiche l'icône prédéfinie correspondant à la gravité du message. <div><div> Question</div><div> Warning</div><div> Information</div><div> Critical</div></div>
setText()	Définit le texte du message principal à afficher
setIcon()	Affiche une icône formée de pixmap d'un fichier image
setStandardButtons()	Liste des boutons standard à afficher. Chaque bouton est associé à. <div><div>QMessageBox.Ok0x00000400</div><div>QMessageBox.Open0x00002000</div><div>QMessageBox.Save0x00000800</div><div>QMessageBox.Cancel0x00400000</div><div>QMessageBox.Close0x00200000</div><div>QMessageBox.Yes0x00004000</div><div>QMessageBox.No0x00010000</div><div>QMessageBox.Abort0x00040000</div></div>

# Utilisation de Qt Designer

---

Grâce à sa simple interface glisser-déposer, une interface graphique peut être rapidement créée sans avoir à écrire le code. Ce n'est cependant pas un IDE tel que Visual Studio. Par conséquent, Qt Designer n'a pas la possibilité de déboguer et de créer l'application.

```
pyuic5 ex.ui > ex.py
```

```
pyuic5 -x ex.ui -o ex.py
```

Qt Designer

# MERCI POUR VOTRE ATTENTION

Any Questions?

