

LES BASES ESSENTIELLES DE PyQt6

Objectif :

À la fin de ce module, tu seras capable de :

- Créer une application PyQt6 stable et professionnelle
- Comprendre le rôle de `QApplication` et `QWidget`
- Contrôler la taille, le style, la position, et le comportement d'une fenêtre
- Gérer les événements et les propriétés principales

PARTIE 1 : La classe `QApplication`

■ Description

`QApplication` est le cœur de toute application PyQt6.

Elle :

- Initialise le système graphique,
- Gère les événements clavier/souris,
- Maintient la boucle d'exécution principale.

Sans elle, **aucun widget ne peut fonctionner.**

Structure générale d'une application PyQt6 :

```
from PyQt6.QtWidgets import QApplication, QWidget
import sys

# 1. Créer l'application
app = QApplication(sys.argv)

# 2. Créer une fenêtre (QWidget)
fenetre = QWidget()
fenetre.show()

# 3. Démarrer la boucle d'événements
app.exec()
```

🔍 Méthodes importantes de `QApplication`

Méthode	Description	Exemple
<code>exec()</code>	Démarre la boucle principale	<code>app.exec()</code>
<code>quit()</code>	Ferme l'application	<code>QApplication.quit()</code>
<code>setStyle("Fusion")</code>	Change le style visuel	<code>app.setStyle("Fusion")</code>
<code>palette()</code>	Retourne la palette de couleurs courante	<code>palette = app.palette()</code>

arguments()	Liste des arguments de ligne de commande	<code>print(app.arguments())</code>
clipboard()	Accès au presse-papiers du système	<code>app.clipboard().setText("Texte copié")</code>

Exemple complet

```
from PyQt6.QtWidgets import QApplication, QWidget, QLabel
from PyQt6.QtGui import QFont
import sys

app = QApplication(sys.argv)
app.setStyle("Fusion") # Style moderne

fenetre = QWidget()
fenetre.setWindowTitle("Exemple QApplication")
fenetre.resize(400, 200)

texte = QLabel("Bienvenue dans PyQt6 !", fenetre)
texte.setFont(QFont("Arial", 14))
texte.move(80, 80)

fenetre.show()
app.exec()
```

PARTIE 2 : La classe `QWidget`

Description

`QWidget` est la **classe mère de tous les éléments graphiques** :

- Fenêtres, boutons, labels, champs de texte, etc.
- Tout widget dérive de `QWidget`.

Propriétés essentielles

Méthode / propriété	Description	Exemple
setWindowTitle(str)	Titre de la fenêtre	<code>fen.setWindowTitle("Ma fenêtre")</code>
resize(w, h)	Taille	<code>fen.resize(400, 300)</code>
move(x, y)	Position	<code>fen.move(200, 100)</code>
setGeometry(x, y, w, h)	Position + taille	<code>fen.setGeometry(100, 100, 400, 200)</code>
setStyleSheet(css)	Style visuel	<code>fen.setStyleSheet("background-color: lightblue;")</code>
show()	Affiche la fenêtre	<code>fen.show()</code>
hide()	Cache la fenêtre	<code>fen.hide()</code>

close()	Ferme la fenêtre	fen.close()
setToolTip(str)	Message au survol	fen.setToolTip("Ceci est une fenêtre")

🔍 Méthodes utiles héritées de `QObject`

Méthode	Description
setObjectName("nom")	Définit un identifiant pour le widget
objectName()	Retourne cet identifiant
setEnabled(True/False)	Active/désactive le widget
setVisible(True/False)	Montre/cache le widget
setFocus()	Donne le focus au widget
hasFocus()	Vérifie si le widget a le focus

Exemple complet de `QWidget`

```
from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QPushButton
from PyQt6.QtCore import Qt
import sys

class MaFenetre(QWidget):
    def __init__(self):
        super().__init__()

        # Titre et apparence
        self.setWindowTitle("Fenêtre avancée PyQt6")
        self.setGeometry(100, 100, 400, 250)
        self.setStyleSheet("background-color: #ddeeff; border: 2px solid #557799;")

        # Label
        self.label = QLabel("Bonjour à tous 🖐️", self)
        self.label.move(130, 80)
        self.label.setStyleSheet("font-size: 18px; color: #223366;")

        # Bouton
        self.btn = QPushButton("Changer le texte", self)
        self.btn.move(130, 130)
        self.btn.clicked.connect(self.change_texte)

    def change_texte(self):
        self.label.setText("Texte modifié ✔️")
        self.label.adjustSize()  # Ajuste la taille du label

app = QApplication(sys.argv)
fen = MaFenetre()
fen.show()
app.exec()
```

PARTIE 3 : Gestion des événements (QEvent)

QWidget peut réagir à divers événements du système :

- clics, touches clavier, mouvement de la souris, redimensionnement, fermeture, etc.

Événements courants

Méthode à surcharger	Type d'événement	Exemple
<code>mousePressEvent(event)</code>	Clic de souris	<code>print("Clic détecté")</code>
<code>keyPressEvent(event)</code>	Touche du clavier	<code>event.key()</code>
<code>resizeEvent(event)</code>	Fenêtre redimensionnée	<code>print("Nouvelle taille")</code>
<code>closeEvent(event)</code>	Fermeture de la fenêtre	<code>event.accept()</code> ou <code>event.ignore()</code>

Exemple pratique – Gestion de clics et clavier

```
from PyQt6.QtWidgets import QApplication, QWidget, QLabel
from PyQt6.QtCore import Qt
import sys

class FenetreEvenement(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Événements PyQt6")
        self.setGeometry(200, 100, 400, 250)
        self.label = QLabel("Cliquez ou tapez une touche", self)
        self.label.move(80, 100)
        self.label.setStyleSheet("font-size: 16px; color: darkblue;")

    def mousePressEvent(self, event):
        if event.button() == Qt.MouseButton.LeftButton:
            self.label.setText("Clic gauche 🖱️")
        elif event.button() == Qt.MouseButton.RightButton:
            self.label.setText("Clic droit 🖱️")
        self.label.adjustSize()

    def keyPressEvent(self, event):
        self.label.setText(f"Touche pressée : {event.text()}")
        self.label.adjustSize()

app = QApplication(sys.argv)
fen = FenetreEvenement()
fen.show()
app.exec()
```

PARTIE 4 : Organisation professionnelle du code

Pour des projets plus grands, on utilise la **programmation orientée objet (POO)** :

```
import sys
from PyQt6.QtWidgets import QApplication, QWidget

class App(QApplication):
    def __init__(self, argv):
        super().__init__(argv)
        self.fen = FenetrePrincipale()
        self.fen.show()

class FenetrePrincipale(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Structure pro PyQt6")
        self.resize(400, 200)

if __name__ == "__main__":
    app = App(sys.argv)
    sys.exit(app.exec())
```

RÉSUMÉ DU MODULE 1

Élément	Classe	Rôle
Application	<code>QApplication</code>	Gère le cycle de vie de l'app
Fenêtre	<code>QWidget</code>	Élément graphique de base
Gestion d'événements	<code>QEvent</code>	Permet d'interagir avec l'utilisateur
Programmation orientée objet	<code>class ...</code>	Structure professionnelle du code

Souhaites-tu maintenant que je passe au **MODULE 2 : Les Widgets principaux**

(`QPushButton`, `QLabel`, `QLineEdit`, `QTextEdit`, etc.)

🔗 Avec **toutes les méthodes, propriétés, et exercices avancés** ?

Chapitre 2 — LES WIDGETS PRINCIPAUX

OBJECTIFS :

À la fin de ce module, tu sauras :

- Manipuler les widgets les plus utilisés (`QPushButton`, `QLabel`, `QLineEdit`, `QTextEdit`, `QComboBox`, `QCheckBox`, `QRadioButton`)
- Connecter des **signaux** et **slots**
- Créer des interfaces dynamiques et réactives
- Personnaliser les widgets avec **QStyle**, **QIcon** et **CSS**

PARTIE 1 : `QPushButton`

Description

Un **bouton cliquable**, utilisé pour exécuter une action.

Méthodes et propriétés utiles

Méthode / Propriété	Description	Exemple
<code>setText("Texte")</code>	Change le texte du bouton	<code>btn.setText("Envoyer")</code>
<code>text()</code>	Retourne le texte	<code>print(btn.text())</code>
<code>clicked.connect(fonction)</code>	Connecte un clic à une fonction	<code>btn.clicked.connect(self.action)</code>
<code>setIcon(QIcon("img.png"))</code>	Ajoute une icône	<code>btn.setIcon(QIcon("logo.png"))</code>
<code>setEnabled(False)</code>	Désactive le bouton	<code>btn.setEnabled(False)</code>
<code>setCheckable(True)</code>	Rend le bouton "toggle"	<code>btn.setCheckable(True)</code>
<code>isChecked()</code>	Vérifie l'état toggle	<code>if btn.isChecked(): ...</code>

Exemple complet

```
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, QLabel,
QVBoxLayout
from PyQt6.QtGui import QIcon
import sys

class FenetreBouton(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QPushButton avancé")
        self.resize(300, 200)

        self.layout = QVBoxLayout()

        self.label = QLabel("État : Aucun clic")
        self.bouton = QPushButton("Clique-moi")
        self.bouton.setIcon(QIcon("icon.png"))
        self.bouton.setCheckable(True)
        self.bouton.clicked.connect(self.on_click)

        self.layout.addWidget(self.label)
        self.layout.addWidget(self.bouton)
        self.setLayout(self.layout)

    def on_click(self):
        if self.bouton.isChecked():
            self.label.setText("État : Activé ✓")
        else:
            self.label.setText("État : Désactivé ✗")

app = QApplication(sys.argv)
fen = FenetreBouton()
fen.show()
app.exec()
```

PARTIE 2 : QLabel

Description

Affiche du texte ou une image.

C'est un widget **non interactif**, mais très personnalisable.

Méthodes principales

Méthode	Description	Exemple
<code>setText("Texte")</code>	Définit le texte	<code>label.setText("Bonjour")</code>
<code>setPixmap(QPixmap("photo.png"))</code>	Affiche une image	<code>label.setPixmap(QPixmap("photo.png"))</code>
<code>setAlignment(Qt.AlignmentFlag.AlignCenter)</code>	Aligne le contenu	<code>label.setAlignment(Qt.AlignmentFlag.AlignCenter)</code>
<code>setWordWrap(True)</code>	Active le retour à la ligne	<code>label.setWordWrap(True)</code>
<code>setStyleSheet(css)</code>	Applique du style	<code>label.setStyleSheet("color: blue; font-size: 16px;")</code>

Exemple complet

```
from PyQt6.QtWidgets import QApplication, QLabel, QWidget, QVBoxLayout
from PyQt6.QtGui import QPixmap
from PyQt6.QtCore import Qt
import sys

class FenetreLabel(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QLabel avancé")
        self.resize(400, 300)

        layout = QVBoxLayout()
        label_img = QLabel()
        label_img.setPixmap(QPixmap("image.png"))
        label_img.setAlignment(Qt.AlignmentFlag.AlignCenter)

        label_txt = QLabel("Bienvenue sur PyQt6 🐍")
        label_txt.setAlignment(Qt.AlignmentFlag.AlignCenter)
        label_txt.setStyleSheet("font-size: 18px; color: #0055aa;")

        layout.addWidget(label_img)
        layout.addWidget(label_txt)
        self.setLayout(layout)

app = QApplication(sys.argv)
```

```
fen = FenetreLabel()
fen.show()
app.exec()
```

PARTIE 3 : QLineEdit

Description

Zone de saisie à **une seule ligne** (nom, email, mot de passe, etc.)

Méthodes importantes

Méthode	Description	Exemple
<code>text()</code>	Récupère le texte saisi	<code>nom = saisie.text()</code>
<code>setText("Valeur")</code>	Définit le texte	<code>saisie.setText("Bonjour")</code>
<code>setPlaceholderText("Entrer...")</code>	Texte indicatif	<code>saisie.setPlaceholderText("Entrez votre nom")</code>
<code>setEchoMode(QLineEdit.EchoMode.Password)</code>	Masque le texte	<code>saisie.setEchoMode(QLineEdit.EchoMode.Password)</code>
<code>textChanged.connect()</code>	Détecte une saisie	<code>saisie.textChanged.connect(self.changement)</code>

Exemple

```
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLineEdit, QLabel
from PyQt6.QtCore import Qt
import sys

class FenetreLineEdit(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QLineEdit avancé")
        self.resize(350, 200)

        self.layout = QVBoxLayout()

        self.label = QLabel("Tapez votre nom :")
        self.saisie = QLineEdit()
        self.saisie.setPlaceholderText("Entrez ici...")
        self.saisie.textChanged.connect(self.mise_a_jour)

        self.affiche = QLabel("")
        self.affiche.setAlignment(Qt.AlignmentFlag.AlignCenter)

        self.layout.addWidget(self.label)
        self.layout.addWidget(self.saisie)
```

```

        self.layout.addWidget(self.affiche)
        self.setLayout(self.layout)

    def mise_a_jour(self):
        self.affiche.setText(f"Bonjour, {self.saisie.text()} 🖱️")

app = QApplication(sys.argv)
fen = FenetreLineEdit()
fen.show()
app.exec()

```

PARTIE 4 : QTextEdit

Description

Zone de texte **multiligne**, idéale pour les notes ou commentaires.

⚙️ Méthodes utiles

Méthode	Description
<code>setPlainText(str)</code>	Définit du texte brut
<code>setHtml("Texte")</code>	Accepte du HTML
<code>toPlainText()</code>	Retourne le texte
<code>toHtml()</code>	Retourne le HTML
<code>setReadOnly(True)</code>	Empêche la modification

Exemple

```

from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QTextEdit,
QPushButton
import sys

class FenetreTextEdit(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QTextEdit avancé")
        self.resize(400, 300)

        layout = QVBoxLayout()
        self.texte = QTextEdit()
        self.bouton = QPushButton("Afficher le contenu")
        self.bouton.clicked.connect(self.afficher)

        layout.addWidget(self.texte)
        layout.addWidget(self.bouton)
        self.setLayout(layout)

    def afficher(self):
        print(self.texte.toPlainText())

```

```
app = QApplication(sys.argv)
fen = FenetreTextEdit()
fen.show()
app.exec()
```

PARTIE 5 : QComboBox, QCheckBox, QRadioButton

QComboBox (liste déroulante)

```
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QComboBox,
QLabel
import sys

class FenetreCombo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QComboBox")
        self.resize(300, 150)

        layout = QVBoxLayout()
        self.combo = QComboBox()
        self.combo.addItem("Python", "C++", "Java", "Kotlin"])
        self.combo.currentTextChanged.connect(self.affiche_langue)

        self.label = QLabel("Choisissez un langage")

        layout.addWidget(self.combo)
        layout.addWidget(self.label)
        self.setLayout(layout)

    def affiche_langue(self, texte):
        self.label.setText(f"Langage choisi : {texte}")

app = QApplication(sys.argv)
fen = FenetreCombo()
fen.show()
app.exec()
```

QCheckBox (cases à cocher)

```
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QCheckBox,
QLabel
import sys

class FenetreCheckBox(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QCheckBox")
        self.resize(250, 150)

        layout = QVBoxLayout()
        self.box1 = QCheckBox("Activer le son")
        self.box2 = QCheckBox("Activer les notifications")
        self.label = QLabel("Aucune option sélectionnée")

        self.box1.stateChanged.connect(self.mise_a_jour)
```

```

        self.box2.stateChanged.connect(self.mise_a_jour)

        layout.addWidget(self.box1)
        layout.addWidget(self.box2)
        layout.addWidget(self.label)
        self.setLayout(layout)

    def mise_a_jour(self):
        etat = []
        if self.box1.isChecked(): etat.append("Son")
        if self.box2.isChecked(): etat.append("Notifications")
        self.label.setText("Activé : " + ", ".join(etat) if etat else
"Aucun")

app = QApplication(sys.argv)
fen = FenetreCheckBox()
fen.show()
app.exec()

```

QRadioButton (choix exclusif)

```

from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout,
QRadioButton, QLabel, QPushButton
import sys

```

```

class FenetreRadio(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QRadioButton")
        self.resize(250, 150)

        layout = QVBoxLayout()
        self.label = QLabel("Choisissez votre sexe :")

        self.homme = QRadioButton("Homme")
        self.femme = QRadioButton("Femme")

        groupe = QPushButton()
        groupe.addButton(self.homme)
        groupe.addButton(self.femme)

        self.homme.toggled.connect(self.selection)
        self.femme.toggled.connect(self.selection)

        layout.addWidget(self.label)
        layout.addWidget(self.homme)
        layout.addWidget(self.femme)
        self.setLayout(layout)

    def selection(self):
        if self.homme.isChecked():
            self.label.setText("Sexe choisi : Homme ♂")
        elif self.femme.isChecked():
            self.label.setText("Sexe choisi : Femme ♀")

app = QApplication(sys.argv)
fen = FenetreRadio()
fen.show()
app.exec()

```

Widget	Classe	Utilisation principale
Bouton	<code>QPushButton</code>	Action au clic
Étiquette	<code>QLabel</code>	Afficher texte/image
Champ de texte	<code>QLineEdit</code>	Saisie simple
Zone de texte	<code>QTextEdit</code>	Saisie multiligne
Liste déroulante	<code>QComboBox</code>	Choix parmi plusieurs
Case à cocher	<code>QCheckBox</code>	Choix multiples
Boutons radio	<code>QRadioButton</code>	Choix exclusif

Chapitre 3 – Les Layouts et l’organisation de l’interface

Comprendre et maîtriser les *layouts* est fondamental pour construire des interfaces propres, adaptables et réactives.

Objectifs :

À la fin de ce module, tu seras capable de :

Organiser tes widgets automatiquement sans utiliser `move(x, y)`

Créer des interfaces dynamiques qui s’adaptent à la taille de la fenêtre

Combiner plusieurs layouts pour structurer des interfaces complexes

Créer des formulaires professionnels avec `QFormLayout`

1. Qu’est-ce qu’un Layout ?

Un **layout** (disposition) est un objet qui **gère automatiquement la position et la taille** des widgets dans une fenêtre.

Au lieu de placer manuellement avec `.move()`, tu “ajoutes” les widgets dans un layout, et PyQt6 s’occupe du reste.

Tous les layouts héritent de `QLayout`.

2. Les principaux Layouts

Classe	Type de disposition	Exemple d’usage
<code>QVBoxLayout</code>	Verticale (haut → bas)	Pile de boutons
<code>QHBoxLayout</code>	Horizontale (gauche → droite)	Barre d’outils
<code>QGridLayout</code>	Grille (lignes/colonnes)	Tableaux, formulaires
<code>QFormLayout</code>	Étiquettes + champs	Formulaire d’inscription

3. QVBoxLayout (Disposition verticale)

Description :

Les widgets sont empilés du **haut vers le bas**.

Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel,
QPushButton
import sys

class FenetreVerticale(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QVBoxLayout")
        self.resize(300, 200)

        layout = QVBoxLayout()

        layout.addWidget(QLabel("En haut"))
        layout.addWidget(QPushButton("Milieu"))
        layout.addWidget(QPushButton("En bas"))

        self.setLayout(layout)

app = QApplication(sys.argv)
fen = FenetreVerticale()
fen.show()
app.exec()
```

4. QHBoxLayout (Disposition horizontale)

Description :

Les widgets sont placés **de gauche à droite**.

Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QHBoxLayout, QPushButton
import sys

class FenetreHorizontale(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QHBoxLayout")
        self.resize(300, 100)

        layout = QHBoxLayout()
        layout.addWidget(QPushButton("← Précédent"))
        layout.addWidget(QPushButton("Suivant →"))

        self.setLayout(layout)

app = QApplication(sys.argv)
fen = FenetreHorizontale()
fen.show()
app.exec()
```

5. Combinaison de Layouts (imbriqués)

Tu peux **combiner plusieurs layouts** dans une même fenêtre (par exemple, un `QVBoxLayout` contenant des `QHBoxLayout`).

❑ Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout,
QHBoxLayout, QPushButton, QLabel
import sys

class FenetreImbriquee(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Combinaison de layouts")
        self.resize(400, 250)

        principal = QVBoxLayout()
        label = QLabel("Barre de navigation :")

        nav = QHBoxLayout()
        nav.addWidget(QPushButton("Accueil"))
        nav.addWidget(QPushButton("Profil"))
        nav.addWidget(QPushButton("Paramètres"))

        principal.addWidget(label)
        principal.addLayout(nav)
        principal.addWidget(QPushButton("Déconnexion"))

        self.setLayout(principal)

app = QApplication(sys.argv)
fen = FenetreImbriquee()
fen.show()
app.exec()
```

Ici :

`principal` est un layout vertical, et `nav` est un layout horizontal intégré à l'intérieur.

6. QGridLayout (Disposition en grille)

Description :

Place les widgets selon un **système de lignes et colonnes**.
Très utile pour les formulaires ou panneaux de configuration.

Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QGridLayout, QLabel,
QLineEdit, QPushButton
import sys

class FenetreGrille(QWidget):
    def __init__(self):
        super().__init__()
```

```

self.setWindowTitle("QGridLayout")
self.resize(300, 200)

layout = QGridLayout()

layout.addWidget(QLabel("Nom :"), 0, 0)
layout.addWidget(QLineEdit(), 0, 1)

layout.addWidget(QLabel("Email :"), 1, 0)
layout.addWidget(QLineEdit(), 1, 1)

layout.addWidget(QPushButton("Envoyer"), 2, 0, 1, 2) # (row, col,
rowspan, colspan)

self.setLayout(layout)

app = QApplication(sys.argv)
fen = FenetreGrille()
fen.show()
app.exec()

```

Note :

- `addWidget(widget, ligne, colonne, hauteur, largeur)`
- Très utile pour **centrer** ou **fusionner des cellules**.

7. QFormLayout (Formulaires professionnels)

Description :

Dispose des **paires (Label / Champ)** de manière élégante.

Exemple :

```

from PyQt6.QtWidgets import QApplication, QWidget, QFormLayout, QLabel,
QLineEdit, QSpinBox, QPushButton
import sys

class FenetreForm(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QFormLayout avancé")
        self.resize(350, 200)

        form = QFormLayout()
        form.addRow("Nom :", QLineEdit())
        form.addRow("Âge :", QSpinBox())
        form.addRow("Email :", QLineEdit())
        form.addRow("", QPushButton("Soumettre"))

        self.setLayout(form)

app = QApplication(sys.argv)
fen = FenetreForm()
fen.show()
app.exec()

```

8. Personnalisation avancée des Layouts

Tu peux ajuster :

- **Les marges internes** (`setContentsMargins`)
- **Les espacements entre widgets** (`setSpacing`)
- **L'alignement des widgets** (`alignment` dans `addWidget`)

Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton
from PyQt6.QtCore import Qt
import sys

class FenetreAvancee(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Personnalisation Layouts")

        layout = QVBoxLayout()
        layout.setSpacing(20)
        layout.setContentsMargins(40, 20, 40, 20)

        layout.addWidget(QPushButton("Haut"),
            alignment=Qt.AlignmentFlag.AlignLeft)
        layout.addWidget(QPushButton("Milieu"),
            alignment=Qt.AlignmentFlag.AlignCenter)
        layout.addWidget(QPushButton("Bas"),
            alignment=Qt.AlignmentFlag.AlignRight)

        self.setLayout(layout)

app = QApplication(sys.argv)
fen = FenetreAvancee()
fen.show()
app.exec()
```

9. Layouts imbriqués dans QMainWindow

`QMainWindow` n'accepte **qu'un seul layout principal** via `setCentralWidget`.
Donc tu dois créer un `QWidget` qui contient le layout.

Exemple pro :

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget,
    QVBoxLayout, QLabel, QPushButton
import sys

class FenetrePrincipale(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Layouts dans QMainWindow")
        self.resize(400, 250)
```

```

    conteneur = QWidget()
    layout = QVBoxLayout()

    layout.addWidget(QLabel("Bienvenue"))
    layout.addWidget(QPushButton("Continuer"))
    conteneur.setLayout(layout)

    self.setCentralWidget(conteneur)

app = QApplication(sys.argv)
win = FenetrePrincipale()
win.show()
app.exec()

```

10. Résumé du module

Classe	Disposition	Usage typique
QVBoxLayout	Verticale	Pile de boutons, menu vertical
QHBoxLayout	Horizontale	Barre d'outils, navigation
QGridLayout	Grille	Tableaux, formulaires
QFormLayout	Aligné Label/Champ	Formulaire d'inscription
addLayout()	Imbrication	Layouts complexes
setContentsMargins()	Contrôle des marges	Design fin
setSpacing()	Espacement interne	Esthétique et clarté

EXERCICE

Crée une interface de **formulaire d'inscription** avec :

- **QVBoxLayout** principal
- **QFormLayout** pour les champs (Nom, Email, Mot de passe, Genre)
- Un **QHBoxLayout** pour les boutons (Envoyer, Annuler)
- Un message **QLabel** qui affiche les données envoyées quand on clique sur "Envoyer"

Chapitre 4 — Les Boîtes de Dialogue et Interactions

Objectifs du module

À la fin de ce module, tu sauras :

Afficher des messages d'information, d'erreur, d'avertissement ou de confirmation
 Créer des fenêtres de sélection de fichiers, de couleurs, et de polices
 Utiliser les dialogues standards de PyQt6 (**QMessageBox**, **QFileDialog**, **QColorDialog**, **QFontDialog**)

Créer tes **propres boîtes de dialogue personnalisées (QDialog)**
Faire communiquer ces dialogues avec ta fenêtre principale

1. Introduction aux dialogues

Un **dialogue** est une **fenêtre secondaire** (modale ou non-modale) utilisée pour interagir avec l'utilisateur.

- **Modale** : bloque la fenêtre principale tant qu'elle n'est pas fermée (ex: message d'erreur).
- **Non modale** : peut être ouverte tout en utilisant la fenêtre principale.

Tous les dialogues héritent de `QDialog`.

2. QMessageBox — Boîtes de message standard

Description :

Permet d'afficher des messages (information, avertissement, erreur, confirmation...).

Exemple de base :

```
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, QMessageBox
import sys




class Fenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QMessageBox")
        self.resize(300, 200)

        bouton = QPushButton("Afficher message", self)
        bouton.clicked.connect(self.afficher_message)

    def afficher_message(self):
        QMessageBox.information(self, "Information", "Bonjour, ceci est un
message d'information !")

app = QApplication(sys.argv)
fen = Fenetre()
fen.show()
app.exec()
```

Types de messages :

Type	Méthode	Couleur / Icône
Information	<code>QMessageBox.information()</code>	 Bleu
Avertissement	<code>QMessageBox.warning()</code>	 Jaune
Erreur / Critique	<code>QMessageBox.critical()</code>	 Rouge

Question	QMessageBox.question()	? Bleu
----------	------------------------	--------

Exemple avec question :

```
def afficher_question(self):
    reponse = QMessageBox.question(
        self, "Confirmation", "Voulez-vous vraiment quitter ?",
        QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
    )
    if reponse == QMessageBox.StandardButton.Yes:
        self.close()
```

💡 QMessageBox.question() retourne la réponse de l'utilisateur.

3. QFileDialog — Sélection de fichiers et dossiers

Description :

Permet à l'utilisateur de choisir un **fichier à ouvrir, enregistrer ou un dossier**.

Exemple (ouvrir un fichier) :

```
from PyQt6.QtWidgets import QFileDialog

def ouvrir_fichier(self):
    chemin, _ = QFileDialog.getOpenFileName(self, "Ouvrir un fichier", "",
    "Texte (*.txt);;Tous les fichiers (*)")
    if chemin:
        QMessageBox.information(self, "Fichier choisi", f"Chemin :
        {chemin}")
```

Exemple (enregistrer un fichier) :

```
def enregistrer_fichier(self):
    chemin, _ = QFileDialog.getSaveFileName(self, "Enregistrer sous", "",
    "Texte (*.txt)")
    if chemin:
        QMessageBox.information(self, "Sauvegarde", f"Fichier enregistré :
        {chemin}")
```

Exemple (choisir un dossier) :

```
def choisir_dossier(self):
    dossier = QFileDialog.getExistingDirectory(self, "Choisir un dossier")
    if dossier:
        QMessageBox.information(self, "Dossier choisi", f"Chemin :
        {dossier}")
```

4. QColorDialog — Sélection de couleur

Description :

Permet de choisir une couleur depuis une palette.

Exemple :

```
from PyQt6.QtWidgets import QColorDialog, QLabel
from PyQt6.QtGui import QColor

def choisir_couleur(self):
    couleur = QColorDialog.getColor(QColor("white"), self, "Choisir une couleur")
    if couleur.isValid():
        self.setStyleSheet(f"background-color: {couleur.name()}");
```

`couleur.name()` renvoie le code hexadécimal (ex: #ff0000).

5. QFontDialog — Choisir une police de texte

Exemple :

```
from PyQt6.QtWidgets import QFontDialog, QLabel

def choisir_police(self):
    police, ok = QFontDialog.getFont()
    if ok:
        self.label.setFont(police)
```

6. QInputDialog — Saisie rapide

■ Description :

Permet à l'utilisateur de saisir une **valeur simple** (texte, nombre, élément d'une liste...).

□ Exemple :

```
from PyQt6.QtWidgets import QInputDialog

def saisir_nom(self):
    nom, ok = QInputDialog.getText(self, "Nom", "Entrez votre nom :")
    if ok and nom:
        QMessageBox.information(self, "Bienvenue", f"Bonjour {nom} !")
```

7. QDialog personnalisé (avancé)

Tu peux créer **ta propre boîte de dialogue** avec des widgets personnalisés.

Exemple :

```
from PyQt6.QtWidgets import QDialog, QVBoxLayout, QLabel, QLineEdit,
QPushButton

class DialoguePerso(QDialog):
```

```
def __init__(self):
    super().__init__()
    self.setWindowTitle("Connexion")
    self.setModal(True)

    layout = QVBoxLayout()
    self.nom = QLineEdit()
    self.mdp = QLineEdit()
    self.mdp.setEchoMode(QLineEdit.EchoMode.Password)

    bouton = QPushButton("Se connecter")
    bouton.clicked.connect(self.accept)

    layout.addWidget(QLabel("Nom d'utilisateur :"))
    layout.addWidget(self.nom)
    layout.addWidget(QLabel("Mot de passe :"))
    layout.addWidget(self.mdp)
    layout.addWidget(bouton)
    self.setLayout(layout)
```

Et dans ta fenêtre principale :

```
def ouvrir_connexion(self):
    dlg = DialoguePerso()
    if dlg.exec():
        QMessageBox.information(self, "Connecté", f"Bonjour {dlg.nom.text()} !")
```

💡 `exec()` ouvre la boîte de dialogue **de manière modale** et retourne 1 (si acceptée) ou 0 (si annulée).

8. Différences entre Modale et Non-Modale

Type	Méthode	Comportement
Modale	<code>exec()</code>	Bloque la fenêtre principale jusqu'à fermeture
Non-Modale	<code>show()</code>	Permet de continuer à interagir avec la fenêtre principale

9. Styles et icônes dans QMessageBox

Tu peux personnaliser les boutons et icônes :

```
msg = QMessageBox()
msg.setWindowTitle("Attention")
msg.setText("Voulez-vous continuer ?")
msg.setIcon(QMessageBox.Icon.Warning)
msg.setStandardButtons(QMessageBox.StandardButton.Yes |
    QMessageBox.StandardButton.No)
msg.setDefaultButton(QMessageBox.StandardButton.No)
reponse = msg.exec()
```

Classe	Utilité	Exemple rapide
--------	---------	----------------

QMessageBox	Messages simples	information(), warning()
QFileDialog	Fichiers & dossiers	getOpenFileName()
QColorDialog	Choisir une couleur	getColor()
QFontDialog	Choisir police	getFont()
QInputDialog	Saisie rapide	getText()
QDialog	Boîte personnalisée	exec() / accept() / reject()

MODULE 5 — Signaux, Slots et Événements

Objectifs du module

À la fin de ce module, tu sauras :

Utiliser et connecter les **signaux (signals)** et **slots (slots)**

Créer **tes propres signaux personnalisés**

Gérer les **événements de la souris, du clavier et de la fenêtre**

Empêcher ou redéfinir certains comportements (ex: intercepter la fermeture d'une fenêtre)

Créer une **communication entre plusieurs widgets ou fenêtres**

1. Qu'est-ce qu'un Signal et un Slot ?

☞ **Signal** : émis lorsqu'un événement se produit (ex : clic sur un bouton).

☞ **Slot** : une fonction (ou méthode) appelée lorsqu'un signal est émis.

□ **Principe :**

```
signal.connect(slot)
```

2. Exemple de base

```
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, QLabel,
QVBoxLayout
import sys
```

```
class Fenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Signaux et Slots")
        self.resize(300, 200)

        self.label = QLabel("Clique sur le bouton")
        bouton = QPushButton("Clique-moi")

        bouton.clicked.connect(self.quand_clique)

        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(bouton)
        self.setLayout(layout)
```

```
def quand_clique(self):
    self.label.setText("Bouton cliqué !")

app = QApplication(sys.argv)
f = Fenetre()
f.show()
app.exec()
```

Ici, le signal `clicked` du bouton est **connecté** au slot `quand_clique`.

3. Connexion de plusieurs signaux à un même slot

```
btn1.clicked.connect(self.afficher_message)
btn2.clicked.connect(self.afficher_message)
```

Peu importe le bouton, la même fonction sera appelée.

4. Connexion d'un signal à plusieurs slots

```
btn.clicked.connect(self.afficher_message)
btn.clicked.connect(self.jouer_son)
```

Quand le bouton est cliqué, les deux fonctions s'exécutent.

5. Signaux avec arguments

Certains signaux envoient automatiquement des valeurs à leur slot.

Exemple avec `QLineEdit` :

```
self.lineEdit.textChanged.connect(self.afficher_texte)

def afficher_texte(self, texte):
    self.label.setText(f"Vous avez tapé : {texte}")
```

Exemple avec `QSlider` :

```
self.slider.valueChanged.connect(lambda v: self.label.setText(f"Valeur : {v}"))
```

6. Déconnecter un signal

```
btn.clicked.disconnect(self.afficher_message)
```

7. Créer ses propres signaux personnalisés

Pour cela, on utilise `pyqtSignal` du module `PyQt6.QtCore`.

Exemple :

```
from PyQt6.QtCore import pyqtSignal, QObject

class Compteur(QObject):
    changement = pyqtSignal(int)

    def __init__(self):
        super().__init__()
        self.valeur = 0

    def incrementer(self):
        self.valeur += 1
        self.changement.emit(self.valeur)
```

Et dans ta fenêtre :

```
self.compteur = Compteur()
self.compteur.changement.connect(self.mettre_a_jour)

def mettre_a_jour(self, val):
    self.label.setText(f"Compteur : {val}")
```

8. Les Événements en PyQt6

Chaque widget peut **réagir à des événements** comme :

- clic souris
- frappe clavier
- redimensionnement
- fermeture fenêtre, etc.

9. Les événements de souris

Tu peux **redéfinir les méthodes événementielles** dans ta classe :

Méthode	Description
<code>mousePressEvent(event)</code>	Quand on clique
<code>mouseReleaseEvent(event)</code>	Quand on relâche
<code>mouseMoveEvent(event)</code>	Quand on déplace la souris

Exemple :

```
from PyQt6.QtGui import QMouseEvent

def mousePressEvent(self, event: QMouseEvent):
    self.label.setText(f"Clic en position ({event.x()}, {event.y()})")
```

10. Les événements du clavier

Méthode	Description
keyPressEvent (event)	Lorsqu'une touche est enfoncée
keyReleaseEvent (event)	Lorsqu'elle est relâchée

Exemple :

```
from PyQt6.QtGui import QKeyEvent, Qt

def keyPressEvent(self, event: QKeyEvent):
    if event.key() == Qt.Key.Key_Escape:
        self.close()
    elif event.key() == Qt.Key.Key_Space:
        self.label.setText("Espace appuyé !")
```

11. Intercepter la fermeture de fenêtre

Tu peux **annuler ou confirmer la fermeture** avec `closeEvent`.

Exemple :

```
from PyQt6.QtWidgets import QMessageBox

def closeEvent(self, event):
    reponse = QMessageBox.question(self, "Quitter ?", "Voulez-vous vraiment fermer ?",
                                   QMessageBox.StandardButton.Yes |
                                   QMessageBox.StandardButton.No)
    if reponse == QMessageBox.StandardButton.Yes:
        event.accept()
    else:
        event.ignore()
```

12. Les événements de fenêtre

Méthode	Action
resizeEvent (event)	Quand la fenêtre est redimensionnée
moveEvent (event)	Quand elle est déplacée
enterEvent (event)	Quand la souris entre dans le widget
leaveEvent (event)	Quand la souris sort du widget

Exemple :

```
def resizeEvent(self, event):
    self.label.setText(f"Taille : {self.width()} x {self.height()}")
```

13. Événements personnalisés (avancé)

Tu peux aussi créer tes **propres événements** avec `QEvent.registerEventType()`, puis les gérer dans `event()`.

Exemple :

```
from PyQt6.QtCore import QEvent, QApplication

MonEvenement = QEvent.registerEventType()

def event(self, event):
    if event.type() == MonEvenement:
        self.label.setText("Événement personnalisé déclenché !")
        return True
    return super().event(event)

def declencher_evenement(self):
    QApplication.postEvent(self, QEvent(MonEvenement))
```

14. Signaux entre fenêtres

Tu peux utiliser des signaux pour **faire communiquer deux fenêtres**.

Exemple :

```
class FenetreFille(QDialog):
    envoyer_texte = pyqtSignal(str)

    def __init__(self):
        super().__init__()
        self.setWindowTitle("Enfant")
        bouton = QPushButton("Envoyer texte")
        bouton.clicked.connect(lambda: self.envoyer_texte.emit("Salut
depuis la fenêtre fille"))
        layout = QVBoxLayout()
        layout.addWidget(bouton)
        self.setLayout(layout)

class FenetrePrincipale(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Parent")

        self.label = QLabel("Rien reçu")
        bouton = QPushButton("Ouvrir Fille")
        bouton.clicked.connect(self.ouvrir_fille)
```

```

        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(bouton)
        self.setLayout(layout)

    def ouvrir_fille(self):
        dlg = FenetreFille()
        dlg.envoyer_texte.connect(self.mettre_texte)
        dlg.exec()

    def mettre_texte(self, txt):
        self.label.setText(txt)

```

15. Résumé du module

Élément	Description	Exemple
<code>signal.connect(slot)</code>	Connecter événement à fonction	<code>btn.clicked.connect(ma_fonction)</code>
<code>pyqtSignal</code>	Créer un signal personnalisé	<code>mon_signal = pyqtSignal(str)</code>
<code>mousePressEvent</code>	Clic souris	<code>event.x(), event.y()</code>
<code>keyPressEvent</code>	Touche clavier	<code>event.key()</code>
<code>closeEvent</code>	Interceptor fermeture	<code>event.ignore()</code>
<code>resizeEvent</code>	Fenêtre redimensionnée	<code>self.width(), self.height()</code>
<code>event()</code>	Gérer tout événement	<code>QEvent.type()</code>

Chapitre 6 — Widgets Avancés et Interfaces Complexes

Objectifs du module

À la fin de ce module, tu sauras :

Utiliser les **widgets avancés** (`QTableWidget`, `QTreeWidget`, `QTabWidget`, `QStackedWidget`, `QGroupBox`, `QSplitter`)

Créer des **interfaces multi-pages et organisées**

Gérer les **tableaux et arborescences de données**

Construire des **interfaces modulaires et dynamiques**

Grouper, séparer et organiser les zones de ton application

1. QTableWidget — Tableau interactif

Description :

`QTableWidget` affiche et modifie des **données en tableau** (comme Excel).

Exemple simple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QTableWidgetItem,
QTableWidget, QVBoxLayout
import sys

class Tableau(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QTableWidget")
        self.resize(400, 300)

        table = QTableWidget(3, 3) # 3 lignes, 3 colonnes
        table.setHorizontalHeaderLabels(["Nom", "Âge", "Ville"])

        table.setItem(0, 0, QTableWidgetItem("Yasine"))
        table.setItem(0, 1, QTableWidgetItem("25"))
        table.setItem(0, 2, QTableWidgetItem("Kinshasa"))

        table.setItem(1, 0, QTableWidgetItem("Sarah"))
        table.setItem(1, 1, QTableWidgetItem("21"))
        table.setItem(1, 2, QTableWidgetItem("Lubumbashi"))

        layout = QVBoxLayout()
        layout.addWidget(table)
        self.setLayout(layout)

app = QApplication(sys.argv)
fen = Tableau()
fen.show()
app.exec()
```

Propriétés et méthodes utiles :

Méthode	Rôle
<code>setRowCount(n)</code>	Définit le nombre de lignes
<code>setColumnCount(n)</code>	Définit le nombre de colonnes
<code>setItem(row, col, QTableWidgetItem)</code>	Ajoute une cellule
<code>item(row, col).text()</code>	Récupère la valeur
<code>insertRow(index) / removeRow(index)</code>	Ajouter / supprimer ligne
<code>setEditTriggers()</code>	Contrôle l'édition (ex: double clic)
<code>setSelectionBehavior()</code>	Sélection ligne entière

Exemple avancé : lecture des données du tableau

```
def lire_table(self, table):
    for i in range(table.rowCount()):
        nom = table.item(i, 0).text()
        age = table.item(i, 1).text()
        ville = table.item(i, 2).text()
        print(f"{nom}, {age} ans, {ville}")
```

2. QTreeWidget — Arborescence de données

Description :

Permet d'afficher des **données hiérarchiques** (ex : dossiers/fichiers, catégories/sous-catégories).

Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QTreeWidget,
QTreeWidgetItem, QVBoxLayout
import sys

class Arbre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QTreeWidget")

        arbre = QTreeWidget()
        arbre.setHeaderLabels(["Catégorie", "Description"])

        parent = QTreeWidgetItem(["Langages", "Programmation"])
        QTreeWidgetItem(parent, ["Python", "Langage simple"])
        QTreeWidgetItem(parent, ["C++", "Langage rapide"])
        arbre.addTopLevelItem(parent)

        layout = QVBoxLayout()
        layout.addWidget(arbre)
        self.setLayout(layout)

app = QApplication(sys.argv)
fen = Arbre()
fen.show()
app.exec()
```

Méthodes importantes :

Méthode	Description
<code>addTopLevelItem(item)</code>	Ajoute un élément racine
<code>QTreeWidgetItem(parent, [col1, col2])</code>	Ajoute un sous-élément
<code>itemClicked.connect()</code>	Signal déclenché quand on clique sur un item
<code>currentItem()</code>	Retourne l'élément sélectionné
<code>item.text(col)</code>	Récupère le texte d'une colonne

Exemple — Afficher l'élément cliqué :

```
def __init__(self):
    ...
    arbre.itemClicked.connect(self.afficher_item)

def afficher_item(self, item, column):
    print(f"Tu as cliqué sur : {item.text(0)}")
```

3. QTabWidget — Interface à onglets

■ Description :

Permet de créer des **onglets** comme dans un navigateur web.Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QTabWidget, QLabel,
QVBoxLayout
import sys

class Onglets(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QTabWidget")
        self.resize(400, 200)

        tabs = QTabWidget()

        tab1 = QWidget()
        tab2 = QWidget()

        tab1.setLayout(QVBoxLayout())
        tab1.layout().addWidget(QLabel("Contenu de l'onglet 1"))

        tab2.setLayout(QVBoxLayout())
        tab2.layout().addWidget(QLabel("Contenu de l'onglet 2"))

        tabs.addTab(tab1, "Accueil")
        tabs.addTab(tab2, "Paramètres")

        layout = QVBoxLayout()
        layout.addWidget(tabs)
        self.setLayout(layout)

app = QApplication(sys.argv)
f = Onglets()
f.show()
app.exec()
```

Méthodes utiles :

Méthode	Description
<code>addTab(widget, titre)</code>	Ajoute un onglet
<code>removeTab(index)</code>	Supprime un onglet
<code>currentIndex()</code>	Récupère l'onglet actif
<code>setTabPosition()</code>	Position (haut, bas, gauche, droite)
<code>tabBarClicked.connect()</code>	Signal de clic sur onglet

4. QStackedWidget — Interface multi-pages

Description :

Permet d'empiler plusieurs widgets et d'en afficher **un seul à la fois** (comme un menu avec plusieurs pages).

Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton,
QStackedWidget, QVBoxLayout, QLabel, QHBoxLayout
import sys

class MultiPages(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QStackedWidget")

        self.stack = QStackedWidget()

        page1 = QLabel("Page 1 : Accueil")
        page2 = QLabel("Page 2 : Paramètres")
        self.stack.addWidget(page1)
        self.stack.addWidget(page2)

        bouton1 = QPushButton("Accueil")
        bouton2 = QPushButton("Paramètres")

        bouton1.clicked.connect(lambda: self.stack.setCurrentIndex(0))
        bouton2.clicked.connect(lambda: self.stack.setCurrentIndex(1))

        boutons = QHBoxLayout()
        boutons.addWidget(bouton1)
        boutons.addWidget(bouton2)

        layout = QVBoxLayout()
        layout.addLayout(boutons)
        layout.addWidget(self.stack)
        self.setLayout(layout)

app = QApplication(sys.argv)
fen = MultiPages()
fen.show()
app.exec()
```

Idéal pour créer une **interface à plusieurs écrans** sans changer de fenêtre.

5. QGroupBox — Regrouper visuellement des widgets

Description :

Permet de **grouper des widgets similaires** dans un cadre avec un titre.

Exemple :

```
from PyQt6.QtWidgets import QGroupBox, QCheckBox

groupe = QGroupBox("Options")
layout = QVBoxLayout()
layout.addWidget(QCheckBox("Activer le son"))
layout.addWidget(QCheckBox("Afficher les notifications"))
```

```
groupe.setLayout(layout)
```

6. QSplitter — Séparer les zones redimensionnables

Description :

Permet de créer des **zones ajustables** que l'utilisateur peut redimensionner avec la souris.

Exemple :

```
from PyQt6.QtWidgets import QSplitter, QTextEdit
from PyQt6.QtCore import Qt

split = QSplitter(Qt.Orientation.Horizontal)
split.addWidget(QTextEdit("Zone gauche"))
split.addWidget(QTextEdit("Zone droite"))
```

7. Combinaison avancée : Interface complète

Exemple d'interface combinant QTabWidget, QTableWidget et QStackedWidget :

```
from PyQt6.QtWidgets import *
import sys

class InterfacePro(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Interface professionnelle PyQt6")
        self.resize(600, 400)

        # Création des onglets
        tabs = QTabWidget()
        self.page_accueil = QLabel("Bienvenue dans ton application PyQt6")
        self.page_table = QTableWidget(3, 3)
        self.page_table.setHorizontalHeaderLabels(["Nom", "Âge", "Ville"])

        tabs.addTab(self.page_accueil, "Accueil")
        tabs.addTab(self.page_table, "Tableau")

        # Splitter horizontal
        splitter = QSplitter(Qt.Orientation.Horizontal)
        splitter.addWidget(tabs)
        splitter.addWidget(QTextEdit("Notes"))

        layout = QVBoxLayout()
        layout.addWidget(splitter)
        self.setLayout(layout)

app = QApplication(sys.argv)
fen = InterfacePro()
fen.show()
app.exec()
```

Widget	Utilité	Exemple
--------	---------	---------

QTableWidget	Tableau de données	Grilles interactives
QTreeWidget	Structure hiérarchique	Fichiers, catégories
QTabWidget	Onglets multiples	Interface multi-sections
QStackedWidget	Pages empilées	Navigation entre écrans
QGroupBox	Groupement logique	Paramètres, options
QSplitter	Séparation redimensionnable	Interface ajustable

Chapitre 7 : QMainWindow, Menus, Outils et Barres d'état

Objectifs du module

À la fin de ce module, tu sauras :

Créer une fenêtre principale (QMainWindow)

Ajouter des **menus dynamiques** avec des actions et raccourcis

Créer des **barres d'outils personnalisées**

Afficher une **barre d'état** (status bar)

Gérer les **actions avec signaux**

Intégrer des **boîtes de dialogue** (fichiers, messages, couleur, police)

Structurer ton application comme un **vrai logiciel professionnel**

1. Comprendre QMainWindow

QMainWindow est la **classe de base** pour toutes les applications principales sous PyQt6.

Elle fournit :

- Une **barre de menus**
- Une **barre d'outils**
- Une **barre d'état**
- Une **zone centrale**
- Des **dock widgets** (volets détachables)

Exemple de base :

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QLabel
import sys

class FenetrePrincipale(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Fenêtre principale - PyQt6")
        self.resize(600, 400)
```

```

        label = QLabel("Bienvenue dans PyQt6")
        self.setCentralWidget(label)

app = QApplication(sys.argv)
fen = FenetrePrincipale()
fen.show()
app.exec()

```

2. Ajouter une barre de menus (QMenuBar)

Structure :

```

menuBar = self.menuBar()
menuFichier = menuBar.addMenu("Fichier")
menuEdition = menuBar.addMenu("Édition")
menuAide = menuBar.addMenu("Aide")

```

Ajouter des actions :

```

nouveau = menuFichier.addAction("Nouveau")
ouvrir = menuFichier.addAction("Ouvrir...")
enregistrer = menuFichier.addAction("Enregistrer")
quitter = menuFichier.addAction("Quitter")

quitter.triggered.connect(self.close)

```

Exemple complet :

```

from PyQt6.QtWidgets import QApplication, QMainWindow, QTextEdit
import sys

class Application(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Menu complet PyQt6")
        self.resize(700, 500)

        self.editor = QTextEdit()
        self.setCentralWidget(self.editor)

        # --- Barre de menu ---
        menu = self.menuBar()
        fichier = menu.addMenu("Fichier")
        edition = menu.addMenu("Édition")
        aide = menu.addMenu("Aide")

        # Actions
        nouveau = fichier.addAction("Nouveau")
        ouvrir = fichier.addAction("Ouvrir...")
        enregistrer = fichier.addAction("Enregistrer sous...")
        quitter = fichier.addAction("Quitter")

        quitter.triggered.connect(self.close)
        nouveau.triggered.connect(lambda: self.editor.clear())

```

3. Ajouter des raccourcis clavier

Chaque `QAction` peut avoir un **raccourci** :

```
nouveau.setShortcut("Ctrl+N")
ouvrir.setShortcut("Ctrl+O")
enregistrer.setShortcut("Ctrl+S")
quitter.setShortcut("Ctrl+Q")
```

4. Barre d'outils (QToolBar)

Description :

Une **barre d'outils** contient des boutons rapides pour exécuter des actions du menu.

Exemple :

```
from PyQt6.QtWidgets import QToolBar
from PyQt6.QtGui import QIcon

toolbar = QToolBar("Outils rapides")
self.addToolBar(toolbar)

action_nouveau = toolbar.addAction(QIcon(), "Nouveau")
action_enregistrer = toolbar.addAction(QIcon(), "Enregistrer")

action_nouveau.triggered.connect(lambda: self.editor.setText(""))
```

Personnalisation :

```
toolbar.setMovable(True)
toolbar.setFloatable(True)
toolbar.setToolButtonStyle(Qt.ToolButtonStyle.ToolButtonTextUnderIcon)
```

5. Barre d'état (QStatusBar)

Description :

Affiche des **informations temporaires** à l'utilisateur (état, message, progression...).

Exemple :

```
self.statusBar().showMessage("Prêt")
```

Et pour un message temporaire :

```
self.statusBar().showMessage("Fichier sauvegardé", 3000) # 3 secondes
```

6. Actions avancées (QAction)

Les actions sont le cœur des menus et barres d'outils.

Exemple :

```
from PyQt6.QtGui import QAction, QIcon

action_ouvrir = QAction(QIcon(), "Ouvrir", self)
action_ouvrir.setShortcut("Ctrl+O")
action_ouvrir.setStatusTip("Ouvrir un fichier texte")
action_ouvrir.triggered.connect(self.ouvrir_fichier)
```

7. Boîtes de dialogue standard (QFileDialog, QMessageBox, QColorDialog, QFontDialog)

Ouvrir / Enregistrer un fichier :

```
from PyQt6.QtWidgets import QFileDialog

def ouvrir_fichier(self):
    fichier, _ = QFileDialog.getOpenFileName(self, "Ouvrir un fichier", "",
    "Texte (*.txt)")
    if fichier:
        with open(fichier, "r", encoding="utf-8") as f:
            self.editor.setText(f.read())
```

Enregistrer :

```
def enregistrer(self):
    fichier, _ = QFileDialog.getSaveFileName(self, "Enregistrer sous", "",
    "Texte (*.txt)")
    if fichier:
        with open(fichier, "w", encoding="utf-8") as f:
            f.write(self.editor.toPlainText())
            self.statusBar().showMessage("Fichier enregistré", 3000)
```

Message d'avertissement :

```
from PyQt6.QtWidgets import QMessageBox

def confirmer_quitter(self):
    reponse = QMessageBox.question(self, "Quitter", "Voulez-vous vraiment
    quitter ?")
    if reponse == QMessageBox.StandardButton.Yes:
        self.close()
```

Choisir une couleur :

```
from PyQt6.QtWidgets import QColorDialog

couleur = QColorDialog.getColor()
if couleur.isValid():
    self.editor.setTextColor(couleur)
```

Choisir une police :

```
from PyQt6.QtWidgets import QFontDialog
```

```

police, ok = QFontDialog.getFont()
if ok:
    self.editor.setFont(police)

```

8. Exemple complet — Application de bureau fonctionnelle

```

from PyQt6.QtWidgets import *
from PyQt6.QtGui import QAction, QIcon
import sys

class EditeurTexte(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Editeur PyQt6")
        self.resize(700, 500)

        self.editor = QTextEdit()
        self.setCentralWidget(self.editor)
        self.statusBar().showMessage("Prêt")

        # Menu
        menu = self.menuBar()
        fichier = menu.addMenu("Fichier")

        # Actions
        ouvrir = QAction("Ouvrir", self)
        ouvrir.setShortcut("Ctrl+O")
        ouvrir.triggered.connect(self.ouvrir_fichier)

        enregistrer = QAction("Enregistrer", self)
        enregistrer.setShortcut("Ctrl+S")
        enregistrer.triggered.connect(self.enregistrer_fichier)

        quitter = QAction("Quitter", self)
        quitter.setShortcut("Ctrl+Q")
        quitter.triggered.connect(self.close)

        fichier.addAction([ouvrir, enregistrer, quitter])

        # Barre d'outils
        toolbar = QToolBar("Outils")
        self.addToolBar(toolbar)
        toolbar.addAction(ouvrir)
        toolbar.addAction(enregistrer)

    def ouvrir_fichier(self):
        fichier, _ = QFileDialog.getOpenFileName(self, "Ouvrir", "", "Texte (*.txt)")
        if fichier:
            with open(fichier, "r", encoding="utf-8") as f:
                self.editor.setText(f.read())
            self.statusBar().showMessage(f"Ouvert : {fichier}", 3000)

    def enregistrer_fichier(self):
        fichier, _ = QFileDialog.getSaveFileName(self, "Enregistrer", "", "Texte (*.txt)")
        if fichier:
            with open(fichier, "w", encoding="utf-8") as f:
                f.write(self.editor.toPlainText())
            self.statusBar().showMessage(f"Enregistré : {fichier}", 3000)

```

```
app = QApplication(sys.argv)
fen = EditeurTexte()
fen.show()
app.exec()
```

🔥 Cette application est **un vrai éditeur de texte minimaliste**, fonctionnel et structuré professionnellement.

Élément	Classe	Description
Fenêtre principale	QMainWindow	Contient tout le reste
Menu	QMenuBar / QMenu	Actions du logiciel
Outils	QToolBar	Boutons rapides
Barre d'état	QStatusBar	Infos utilisateur
Action	QAction	Élément déclencheur
Boîtes de dialogue	QFileDialog, QMessageBox, QColorDialog, QFontDialog	Interaction utilisateur

Chapitre 8 — Formulaires, Widgets Interactifs et Validation des Données

Objectifs

À la fin de ce module, tu sauras :

Créer des **formulaires professionnels** avec `QFormLayout`

Utiliser les **widgets d'entrée de données** (`QLineEdit`, `QComboBox`, `QSpinBox`, `QDateEdit`, etc.)

Appliquer des **validateurs de données** (`QValidator`, `QIntValidator`, `QDoubleValidator`, `QRegExpValidator`)

Créer des **formulaires dynamiques** avec validation en temps réel

Gérer les événements utilisateur et la logique d'enregistrement

1. QFormLayout — Création de formulaires structurés

`QFormLayout` aligne les labels et les champs de saisie verticalement, très pratique pour les interfaces de gestion.

Exemple :

```
from PyQt6.QtWidgets import QApplication, QWidget, QFormLayout, QLineEdit,
QPushButton
import sys

class Formulaire(QWidget):
```

```

def __init__(self):
    super().__init__()
    self.setWindowTitle("Formulaire utilisateur")
    self.resize(350, 200)

    layout = QFormLayout()

    self.nom = QLineEdit()
    self.age = QLineEdit()
    self.email = QLineEdit()

    bouton = QPushButton("Enregistrer")

    layout.addRow("Nom :", self.nom)
    layout.addRow("Âge :", self.age)
    layout.addRow("Email :", self.email)
    layout.addRow(bouton)

    self.setLayout(layout)

app = QApplication(sys.argv)
f = Formulaire()
f.show()
app.exec()

```

2. Widgets d'entrée avancés

Widget	Description	Exemple
QLineEdit	Champ de texte	Nom, Email
QTextEdit	Texte long	Description
QComboBox	Liste déroulante	Sexe, Pays
QSpinBox	Nombre entier	Âge
QDoubleSpinBox	Nombre décimal	Montant
QDateEdit / QTimeEdit	Date et heure	Date de naissance
QCheckBox	Case à cocher	Accepter les conditions
QRadioButton	Boutons radio	Choix exclusif
QSlider	Curseur	Volume, niveau
QProgressBar	Barre de progression	Avancement

Exemple — Combinaison complète :

```

from PyQt6.QtWidgets import *

class FormulaireComplet(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Formulaire complet")
        self.resize(400, 300)

```

```

form = QFormLayout()

self.nom = QLineEdit()
self.age = QSpinBox()
self.age.setRange(0, 120)

self.sexe = QComboBox()
self.sexe.addItem("Homme", "Femme", "Autre"])

self.date_naissance = QDateEdit()
self.date_naissance.setCalendarPopup(True)

self.conditions = QCheckBox("J'accepte les conditions")
self.envoyer = QPushButton("Soumettre")

form.addRow("Nom :", self.nom)
form.addRow("Âge :", self.age)
form.addRow("Sexe :", self.sexe)
form.addRow("Date de naissance :", self.date_naissance)
form.addRow(self.conditions)
form.addRow(self.envoyer)

self.setLayout(form)

```

3. Validation des données (QValidator)

Types de validateurs :

Classe	Rôle	Exemple
QIntValidator(min, max)	Nombres entiers	âge, quantité
QDoubleValidator(min, max, décimales)	Nombres décimaux	montant
QRegularExpressionValidator	Modèle personnalisé (Regex)	email, téléphone

Exemple QIntValidator :

```

from PyQt6.QtGui import QIntValidator

self.age.setValidator(QIntValidator(1, 120))

```

Exemple QDoubleValidator :

```

from PyQt6.QtGui import QDoubleValidator

montant = QLineEdit()
montant.setValidator(QDoubleValidator(0.0, 9999.99, 2))

```

Exemple QRegularExpressionValidator (email) :

```

from PyQt6.QtGui import QRegularExpressionValidator
from PyQt6.QtCore import QRegularExpression

regex = QRegularExpression(r"^[\\w\\.\\-]+@[\\w\\.\\-]+\\.\\w+$")
email_validator = QRegularExpressionValidator(regex)
self.email.setValidator(email_validator)

```

4. Validation dynamique et signal textChanged

Permet de vérifier **en temps réel** si l'utilisateur saisit correctement les données.

Exemple :

```

self.email.textChanged.connect(self.verifier_email)

def verifier_email(self):
    texte = self.email.text()
    if "@" not in texte or "." not in texte:
        self.email.setStyleSheet("border: 2px solid red;")
    else:
        self.email.setStyleSheet("border: 2px solid green;")

```

5. Enregistrement et affichage des données

Une fois validées, les données peuvent être récupérées :

```

def enregistrer(self):
    nom = self.nom.text()
    age = self.age.value()
    sexe = self.sexe.currentText()
    date = self.date_naissance.date().toString("dd/MM/yyyy")

    if not self.conditions.isChecked():
        QMessageBox.warning(self, "Erreur", "Vous devez accepter les conditions")
        return

    QMessageBox.information(self, "Succès", f"Nom : {nom}\nÂge : {age}\nSexe : {sexe}\nNé(e) le : {date}")

```

6. Exemple complet Formulaire interactif validé

```

from PyQt6.QtWidgets import *
from PyQt6.QtGui import QRegularExpressionValidator
from PyQt6.QtCore import QRegularExpression
import sys

class FormulaireAvance(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Formulaire Avancé - PyQt6")
        self.resize(400, 300)

        form = QFormLayout()

```

```

self.nom = QLineEdit()
self.age = QSpinBox()
self.age.setRange(0, 120)

self.email = QLineEdit()
regex = QRegularExpression(r"^[\w\.-]+@[\w\.-]+\.\w+$")
self.email.setValidator(QRegularExpressionValidator(regex))

self.sexe = QComboBox()
self.sexe.addItem("Homme", "Femme", "Autre"])

self.date_naissance = QDateEdit()
self.date_naissance.setCalendarPopup(True)

self.conditions = QCheckBox("J'accepte les conditions
d'utilisation")
self.envoyer = QPushButton("Soumettre")
self.envoyer.clicked.connect(self.valider)

form.addRow("Nom :", self.nom)
form.addRow("Âge :", self.age)
form.addRow("Email :", self.email)
form.addRow("Sexe :", self.sexe)
form.addRow("Date de naissance :", self.date_naissance)
form.addRow(self.conditions)
form.addRow(self.envoyer)

self.setLayout(form)

def valider(self):
    if not self.nom.text():
        QMessageBox.warning(self, "Erreur", "Le nom est obligatoire")
        return

    if not self.email.hasAcceptableInput():
        QMessageBox.warning(self, "Erreur", "Adresse email invalide")
        return

    if not self.conditions.isChecked():
        QMessageBox.warning(self, "Erreur", "Veuillez accepter les
conditions")
        return

    QMessageBox.information(self, "Succès", f"Bienvenue
{self.nom.text()} !")

app = QApplication(sys.argv)
f = FormulaireAvance()
f.show()
app.exec()

```

7. Personnalisation visuelle avec CSS (styleSheet)

Tu peux styliser les widgets pour rendre ton formulaire **professionnel et moderne** :

```

self.setStyleSheet("""
    QLineEdit {
        border: 1px solid gray;
        border-radius: 6px;
        padding: 5px;
    }

```

```

    }
    QPushButton {
        background-color: #2c7be5;
        color: white;
        border-radius: 8px;
        padding: 6px 12px;
    }
    QPushButton:hover {
        background-color: #1a5fc2;
    }
    """
)

```

8. Validation avancée — dépendance entre champs

Tu peux faire dépendre un champ d’un autre, par exemple :

“Si le sexe est Femme, afficher le champ ‘Nom de jeune fille’.”

Exemple :

```

self.sexe.currentTextChanged.connect(self.afficher_nom_jeune_fille)

def afficher_nom_jeune_fille(self, texte):
    if texte == "Femme":
        self.nom_jeune_fille.show()
    else:
        self.nom_jeune_fille.hide()

```

Élément	Classe	Description
Formulaire	QFormLayout	Structure alignée label + champ
Champ texte	QLineEdit	Entrée utilisateur
Nombre	QSpinBox, QDoubleSpinBox	Entrée numérique
Liste déroulante	QComboBox	Choix d’options
Date / Heure	QDateEdit, QTimeEdit	Saisie de date
Validation	QValidator	Vérifie les entrées
Interactions	QCheckBox, QRadioButton	Choix utilisateur
Signal dynamique	textChanged, currentTextChanged	Contrôle en temps réel

Chapitre 9 : Gestion avancée des événements, signaux et slots

1 □ Concepts fondamentaux

Dans PyQt6, **les événements** sont des actions déclenchées (clic, saisie, survol, redimensionnement, etc.).

Ces événements peuvent être :

- **automatiques** (clavier, souris...)
- **personnalisés** (créés par toi-même)

Les **signaux et slots** permettent de **connecter un événement à une action** :

Quand un signal est émis un slot (fonction) est exécuté.

Les signaux et slots intégrés

Exemple simple :

```
from PyQt6.QtWidgets import QApplication, QPushButton, QWidget, QVBoxLayout
import sys

def afficher_message():
    print("Bouton cliqué !")

app = QApplication(sys.argv)
fen = QWidget()
fen.setWindowTitle("Signaux et Slots")

layout = QVBoxLayout()
bouton = QPushButton("Clique-moi")
bouton.clicked.connect(afficher_message)  # Signal → Slot

layout.addWidget(bouton)
fen.setLayout(layout)
fen.show()
app.exec()
```

Ici :

- `clicked` = signal du bouton
- `afficher_message` = slot (fonction appelée quand on clique)

Définir ses propres signaux et slots personnalisés

Pour créer un **signal personnalisé**, on hérite de `QObject` ou d'un widget, puis on utilise `pyqtSignal`.

Exemple :

```
from PyQt6.QtCore import QObject, pyqtSignal

class MonObjet(QObject):
    # Définition du signal
    mon_signal = pyqtSignal(str)

    def envoyer(self):
        self.mon_signal.emit("Signal personnalisé émis !")

def reception(message):
```

```

        print("Message reçu :", message)

obj = MonObjet()
obj.mon_signal.connect(reception)
obj.envoyer()

```

□ Explication :

- `pyqtSignal(str)` : signal transportant une chaîne.
- `.emit("...")` : envoie le signal.
- `.connect()` : lie le signal à une fonction.

Créer des signaux personnalisés dans un widget

```

from PyQt6.QtCore import pyqtSignal
from PyQt6.QtWidgets import QApplication, QPushButton, QWidget, QVBoxLayout
import sys

class MonBouton(QPushButton):
    double_clique = pyqtSignal()

    def mouseDoubleClickEvent(self, event):
        self.double_clique.emit() # émet un signal quand double clic

def reaction():
    print("Double clic détecté !")

app = QApplication(sys.argv)
fen = QWidget()
layout = QVBoxLayout()

btn = MonBouton("Double-clique-moi")
btn.double_clique.connect(reaction)

layout.addWidget(btn)
fen.setLayout(layout)
fen.show()
app.exec()

```

💡 Tu viens de créer un **signal personnalisé** sur un bouton standard !

Les événements clavier et souris

Chaque widget peut **intercepter** les événements avec des méthodes comme :

- `mousePressEvent()`
- `mouseReleaseEvent()`
- `mouseMoveEvent()`
- `keyPressEvent()`
- `keyReleaseEvent()`

Exemple :

```

from PyQt6.QtWidgets import QApplication, QWidget
from PyQt6.QtGui import QKeyEvent, QMouseEvent

```

```

import sys

class FenetreEvenement(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Événements clavier et souris")
        self.resize(400, 300)

    def keyPressEvent(self, event: QKeyEvent):
        print(f"Touche appuyée : {event.text()}")

    def mousePressEvent(self, event: QMouseEvent):
        print(f"Clic à la position : {event.position().x()},
{event.position().y()}")

app = QApplication(sys.argv)
fen = FenetreEvenement()
fen.show()
app.exec()

```

Gestion du focus et des raccourcis clavier

Exemple : raccourci clavier (Ctrl+Q pour quitter)

```

from PyQt6.QtWidgets import QApplication, QMainWindow, QAction
from PyQt6.QtGui import QKeySequence
import sys

app = QApplication(sys.argv)
win = QMainWindow()
win.setWindowTitle("Raccourci clavier")

action_quitter = QAction("Quitter", win)
action_quitter.setShortcut(QKeySequence("Ctrl+Q"))
action_quitter.triggered.connect(app.quit)

menu = win.menuBar().addMenu("Fichier")
menu.addAction(action_quitter)

win.show()
app.exec()

```

Utiliser QTimer pour des événements programmés

```

from PyQt6.QtCore import QTimer
from PyQt6.QtWidgets import QApplication, QLabel
import sys

def mise_a_jour():
    global compteur
    compteur += 1
    label.setText(f"Compteur : {compteur}")

app = QApplication(sys.argv)
label = QLabel("Compteur : 0")
label.show()

compteur = 0

```

```

timer = QTimer()
timer.timeout.connect(mise_a_jour)
timer.start(1000) # toutes les 1 seconde

app.exec()

```

Filtrage d'événements (`eventFilter`)

Tu peux **intercepter tous les événements** d'un widget avec `installEventFilter`.

```

from PyQt6.QtCore import QObject, QEvent
from PyQt6.QtWidgets import QApplication, QLabel, QWidget, QVBoxLayout
import sys

class Filtre(QObject):
    def eventFilter(self, obj, event):
        if event.type() == QEvent.Type.MouseButtonPress:
            print("Clic intercepté !")
            return True # bloque l'événement
        return False

app = QApplication(sys.argv)
fen = QWidget()
layout = QVBoxLayout()
label = QLabel("Clique-moi (mais filtré)")
layout.addWidget(label)
fen.setLayout(layout)

filtre = Filtre()
label.installEventFilter(filtre)

fen.show()
app.exec()

```

Créer ses propres événements personnalisés

Tu peux créer ton propre type d'événement en héritant de `QEvent`.

```

from PyQt6.QtCore import QEvent, QCoreApplication
from PyQt6.QtWidgets import QApplication, QWidget
import sys

class MonEvenement(QEvent):
    TypeMonEvenement = QEvent.Type(QEvent.registerEventType())

app = QApplication(sys.argv)

class Fenetre(QWidget):
    def event(self, event):
        if event.type() == MonEvenement.TypeMonEvenement:
            print("Événement personnalisé reçu !")
            return True
        return super().event(event)

fen = Fenetre()
fen.show()

# Émettre l'événement

```

```
QCoreApplication.postEvent(fen,  
MonEvenement(MonEvenement.TypeMonEvenement))
```

```
app.exec()
```

Récapitulatif

Élément	Classe / Fonction	Utilité
Signal standard	<code>.clicked.connect()</code>	Connecter un événement à une action
Signal personnalisé	<code>pyqtSignal / .emit()</code>	Créer des signaux faits maison
Slots personnalisés	Méthodes normales	Réagir à un signal
Événements clavier/souris	<code>keyPressEvent()</code> , <code>mousePressEvent()</code>	Interactivité
Timer	<code>QTimer</code>	Actions répétées
Filtrage global	<code>installEventFilter()</code>	Intercepter tous les événements
Événement custom	<code>QEvent.registerEventType()</code>	Créer tes propres types

Chapitre 10 : Animations, transitions et effets graphiques avancés

Introduction à l'animation dans PyQt6

Les animations dans PyQt6 reposent principalement sur le module `PyQt6.QtCore`, en particulier les classes :

Classe	Rôle
<code>QPropertyAnimation</code>	Anime une propriété d'un widget
<code>QSequentialAnimationGroup</code>	Enchaîne des animations
<code>QParallelAnimationGroup</code>	Exécute plusieurs animations simultanément
<code>QEasingCurve</code>	Définit la vitesse et la courbe d'accélération
<code>QGraphicsOpacityEffect</code>	Anime la transparence
<code>QGraphicsColorizeEffect</code>	Ajoute un effet de couleur

`QPropertyAnimation` (Animation de propriétés)

Tu peux animer **toutes les propriétés** d'un widget qui possèdent un *getter* et un *setter* (par exemple : `geometry`, `pos`, `size`, `windowOpacity`...).

Exemple : Animation de déplacement d'un bouton

```
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton
from PyQt6.QtCore import QPropertyAnimation, QRect
import sys

app = QApplication(sys.argv)
fen = QWidget()
fen.setWindowTitle("Animation de déplacement")
fen.resize(400, 300)

btn = QPushButton("Clique pour bouger", fen)
btn.setGeometry(50, 100, 150, 50)

anim = QPropertyAnimation(btn, b"geometry")
anim.setDuration(2000) # 2 secondes
anim.setStartValue(QRect(50, 100, 150, 50))
anim.setEndValue(QRect(200, 100, 150, 50))
anim.start()

fen.show()
app.exec()
```

b"geometry" indique que tu veux animer la position et la taille.

Courbes de vitesse (`QEasingCurve`)

Elles permettent de rendre le mouvement **fluide, naturel ou rebondissant**.

Exemple :

```
from PyQt6.QtCore import QPropertyAnimation, QEasingCurve, QRect
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton
import sys

app = QApplication(sys.argv)
fen = QWidget()
fen.resize(400, 300)

btn = QPushButton("Animation fluide", fen)
btn.setGeometry(50, 120, 150, 50)

anim = QPropertyAnimation(btn, b"geometry")
anim.setDuration(2000)
anim.setStartValue(QRect(50, 120, 150, 50))
anim.setEndValue(QRect(250, 120, 150, 50))
anim.setEasingCurve(QEasingCurve.Type.OutBounce) # effet rebondissant
anim.start()

fen.show()
app.exec()
```

Types possibles :

- InOutQuad, OutBounce, OutElastic, OutBack, InCubic, etc.

QSequentialAnimationGroup (Exécution en séquence)

Permet de **chaîner plusieurs animations** (une après l'autre).

Exemple :

```
from PyQt6.QtCore import QPropertyAnimation, QSequentialAnimationGroup,
QRect
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton
import sys

app = QApplication(sys.argv)
fen = QWidget()
fen.resize(400, 300)

btn = QPushButton("Séquence", fen)
btn.setGeometry(50, 100, 150, 50)

anim1 = QPropertyAnimation(btn, b"geometry")
anim1.setDuration(1000)
anim1.setStartValue(QRect(50, 100, 150, 50))
anim1.setEndValue(QRect(250, 100, 150, 50))

anim2 = QPropertyAnimation(btn, b"geometry")
anim2.setDuration(1000)
anim2.setStartValue(QRect(250, 100, 150, 50))
anim2.setEndValue(QRect(250, 200, 150, 50))

sequence = QSequentialAnimationGroup()
sequence.addAnimation(anim1)
sequence.addAnimation(anim2)
sequence.start()

fen.show()
app.exec()
```

QParallelAnimationGroup – Exécution simultanée

Permet de **faire bouger plusieurs widgets** ou **plusieurs propriétés d'un même widget** en même temps.

Exemple :

```
from PyQt6.QtCore import QPropertyAnimation, QParallelAnimationGroup, QRect
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton
import sys

app = QApplication(sys.argv)
fen = QWidget()
fen.resize(400, 300)

btn = QPushButton("Animation parallèle", fen)
btn.setGeometry(50, 100, 150, 50)
```

```

anim1 = QPropertyAnimation(btn, b"geometry")
anim1.setDuration(2000)
anim1.setEndValue(QRect(200, 100, 150, 50))

anim2 = QPropertyAnimation(btn, b"windowOpacity")
anim2.setDuration(2000)
anim2.setStartValue(1)
anim2.setEndValue(0.2)

group = QParallelAnimationGroup()
group.addAnimation(anim1)
group.addAnimation(anim2)
group.start()

fen.show()
app.exec()

```

QGraphicsOpacityEffect (Effet de transparence)

Tu peux rendre un widget **semi-transparent** ou **faire un fondu**.

Exemple :

```

from PyQt6.QtWidgets import QApplication, QWidget, QPushButton,
QGraphicsOpacityEffect
from PyQt6.QtCore import QPropertyAnimation
import sys

app = QApplication(sys.argv)
fen = QWidget()
fen.resize(300, 200)

btn = QPushButton("Fondu", fen)
btn.setGeometry(80, 80, 120, 40)

effet = QGraphicsOpacityEffect(btn)
btn.setGraphicsEffect(effet)

anim = QPropertyAnimation(effet, b"opacity")
anim.setDuration(2000)
anim.setStartValue(1.0)
anim.setEndValue(0.0)
anim.start()

fen.show()
app.exec()

```

QGraphicsColorizeEffect (Changer la couleur d'un widget)

Exemple :

```

from PyQt6.QtWidgets import QApplication, QWidget, QPushButton,
QGraphicsColorizeEffect
from PyQt6.QtCore import QPropertyAnimation
from PyQt6.QtGui import QColor
import sys

```

```

app = QApplication(sys.argv)
fen = QWidget()
fen.resize(300, 200)

btn = QPushButton("Colorisation", fen)
btn.setGeometry(80, 80, 120, 40)

effet = QGraphicsColorizeEffect()
btn.setGraphicsEffect(effet)

anim = QPropertyAnimation(effet, b"color")
anim.setDuration(2000)
anim.setStartValue(QColor("red"))
anim.setEndValue(QColor("blue"))
anim.start()

fen.show()
app.exec()

```

Animation répétée et infinie

Tu peux répéter une animation plusieurs fois ou la rendre infinie.

```

anim.setLoopCount(5)          # Répète 5 fois
# ou
anim.setLoopCount(-1)         # Boucle infinie

```

Combiner animation + logique Python

Tu peux lier une animation à un signal `finished` pour exécuter une action après.

```

def apres_anim():
    print("Animation terminée !")

anim.finished.connect(apres_anim)

```

Exercice pratique : Bouton rebondissant et clignotant

Voici un exemple complet qui combine plusieurs effets :

```

from PyQt6.QtWidgets import QApplication, QWidget, QPushButton,
    QGraphicsOpacityEffect
from PyQt6.QtCore import QPropertyAnimation, QEasingCurve,
    QParallelAnimationGroup, QRect
import sys

app = QApplication(sys.argv)
fen = QWidget()
fen.setWindowTitle("Bouton animé")
fen.resize(400, 300)

btn = QPushButton("PyQt6 Power", fen)
btn.setGeometry(120, 120, 160, 50)

```

```
# Effet de transparence
effet = QGraphicsOpacityEffect(btn)
btn.setGraphicsEffect(effet)

# Animation de rebond
anim_move = QPropertyAnimation(btn, b"geometry")
anim_move.setDuration(2000)
anim_move.setStartValue(QRect(120, 120, 160, 50))
anim_move.setEndValue(QRect(120, 60, 160, 50))
anim_move.setEasingCurve(QEasingCurve.Type.OutBounce)

# Animation d'opacité
anim_opacity = QPropertyAnimation(effet, b"opacity")
anim_opacity.setDuration(2000)
anim_opacity.setStartValue(1)
anim_opacity.setEndValue(0.3)
anim_opacity.setLoopCount(-1)

# Groupe parallèle
group = QParallelAnimationGroup()
group.addAnimation(anim_move)
group.addAnimation(anim_opacity)
group.start()

fen.show()
app.exec()
```

Classe	Rôle
QPropertyAnimation	Anime une propriété (taille, position, opacité...)
QSequentialAnimationGroup	Enchaîne des animations
QParallelAnimationGroup	Lance plusieurs animations ensemble
QEasingCurve	Donne un style de mouvement
QGraphicsOpacityEffect	Crée un effet de fondu
QGraphicsColorizeEffect	Modifie dynamiquement la couleur
QPropertyAnimation.setLoopCount()	Répète l'animation
.finished.connect()	Déclenche une action à la fin

🎵 Chapitre 11 : Multimédia, sons et vidéos dans PyQt6

Introduction au multimédia dans PyQt6

PyQt6 propose un module puissant :

PyQt6.QtMultimedia et **PyQt6.QtMultimediaWidgets**

Les classes les plus importantes :

Classe	Description

QMediaPlayer	Lit les fichiers audio et vidéo
QAudioOutput	Contrôle la sortie audio
QVideoWidget	Affiche la vidéo
QSoundEffect	Joue des sons courts (clic, notification, alerte, etc.)
QMediaPlaylist	Gère une liste de musiques/vidéos
QMediaDevices	Liste les périphériques audio/vidéo
QCamera, QCameraViewfinder	Capture vidéo avec webcam (avancé)

Lecture d'un son court avec `QSoundEffect`

Utilisé pour jouer de petits effets sonores.

Exemple :

```
from PyQt6.QtMultimedia import QSoundEffect
from PyQt6.QtCore import QUrl
import time

sound = QSoundEffect()
sound.setSource(QUrl.fromLocalFile("clic.wav"))
sound.setVolume(0.8)  # de 0.0 à 1.0
sound.play()

time.sleep(2)  # attendre la fin du son
```

Idéal pour les sons de clics, alertes, notifications, etc.

Lecture d'un fichier audio avec `QMediaPlayer`

Le couple `QMediaPlayer` + `QAudioOutput` est utilisé pour lire les musiques.

Exemple :

```
from PyQt6.QtWidgets import QApplication, QPushButton, QWidget, QVBoxLayout
from PyQt6.QtMultimedia import QMediaPlayer, QAudioOutput
from PyQt6.QtCore import QUrl
import sys

class LecteurAudio(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Lecteur audio PyQt6")
```

```

self.resize(300, 150)

self.audio_output = QAudioOutput()
self.player = QMediaPlayer()
self.player.setAudioOutput(self.audio_output)
self.player.setSource(QUrl.fromLocalFile("musique.mp3"))

layout = QVBoxLayout()
btn_play = QPushButton("▶ Lire")
btn_pause = QPushButton("⏸ Pause")
btn_stop = QPushButton("⏹ Stop")

btn_play.clicked.connect(self.player.play)
btn_pause.clicked.connect(self.player.pause)
btn_stop.clicked.connect(self.player.stop)

layout.addWidget(btn_play)
layout.addWidget(btn_pause)
layout.addWidget(btn_stop)
self.setLayout(layout)

app = QApplication(sys.argv)
fen = LecteurAudio()
fen.show()
app.exec()

```

□ Points clés :

- `setSource(QUrl.fromLocalFile(...))` : définit le fichier à lire
- `setAudioOutput()` : associe la sortie audio
- `play()`, `pause()`, `stop()` : contrôlent la lecture

Lecture vidéo avec `QVideoWidget` et `QMediaPlayer`

Pour les vidéos, on ajoute un **QVideoWidget** à la fenêtre.

Exemple :

```

from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton
from PyQt6.QtMultimedia import QMediaPlayer, QAudioOutput
from PyQt6.QtMultimediaWidgets import QVideoWidget
from PyQt6.QtCore import QUrl
import sys

class LecteurVideo(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Lecteur vidéo PyQt6")
        self.resize(600, 400)

        layout = QVBoxLayout()

        self.video_widget = QVideoWidget()
        self.audio_output = QAudioOutput()
        self.player = QMediaPlayer()
        self.player.setAudioOutput(self.audio_output)
        self.player.setVideoOutput(self.video_widget)

```

```

self.player.setSource(QUrl.fromLocalFile("video.mp4"))

btn_play = QPushButton("Lire la vidéo")
btn_play.clicked.connect(self.player.play)

layout.addWidget(self.video_widget)
layout.addWidget(btn_play)
self.setLayout(layout)

app = QApplication(sys.argv)
fen = LecteurVideo()
fen.show()
app.exec()

```

Tu peux utiliser `self.player.pause()` et `self.player.stop()` de la même façon que pour l'audio.

Contrôle du volume, durée, position

```

self.audio_output.setVolume(0.5)          # Volume (0.0 à 1.0)
self.player.setPosition(30000)            # Position en ms
self.player.durationChanged.connect(lambda d: print("Durée :", d))
self.player.positionChanged.connect(lambda p: print("Position :", p))

```

QMediaPlaylist (Jouer plusieurs fichiers audio/vidéo)

Tu peux créer un lecteur multimédia complet avec playlist.

Exemple :

```

from PyQt6.QtWidgets import QApplication, QPushButton, QWidget, QVBoxLayout
from PyQt6.QtMultimedia import QMediaPlayer, QAudioOutput, QMediaPlaylist
from PyQt6.QtCore import QUrl
import sys

class LecteurPlaylist(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Playlist PyQt6")

        self.audio_output = QAudioOutput()
        self.player = QMediaPlayer()
        self.player.setAudioOutput(self.audio_output)

        self.playlist = QMediaPlaylist()
        self.playlist.addMedia(QUrl.fromLocalFile("musique1.mp3"))
        self.playlist.addMedia(QUrl.fromLocalFile("musique2.mp3"))
        self.playlist.setCurrentIndex(0)
        self.playlist.setPlaybackMode(QMediaPlaylist.PlaybackMode.Loop)

        self.player.setPlaylist(self.playlist)

        layout = QVBoxLayout()
        btn_play = QPushButton("▶ Play")
        btn_next = QPushButton("▢ Suivant")
        btn_prev = QPushButton("▢ Précédent")

```

```

        btn_play.clicked.connect(self.player.play)
        btn_next.clicked.connect(self.playlist.next)
        btn_prev.clicked.connect(self.playlist.previous)

        layout.addWidget(btn_play)
        layout.addWidget(btn_next)
        layout.addWidget(btn_prev)
        self.setLayout(layout)

app = QApplication(sys.argv)
fen = LecteurPlaylist()
fen.show()
app.exec()

```

Lire un son depuis une URL ou un flux réseau

```

self.player.setSource(QUrl("https://www.example.com/stream.mp3"))
self.player.play()

```

Accéder à la webcam (QCamera + QCameraViewfinder)

Cette partie nécessite PyQt6 complet et support matériel vidéo.

Exemple :

```

from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout
from PyQt6.QtMultimedia import QCamera, QMediaDevices
from PyQt6.QtMultimediaWidgets import QVideoWidget
import sys

app = QApplication(sys.argv)
fen = QWidget()
fen.setWindowTitle("Caméra PyQt6")
fen.resize(640, 480)

layout = QVBoxLayout()
video = QVideoWidget()
layout.addWidget(video)

camera = QCamera(QMediaDevices.defaultVideoInput())
camera.setVideoOutput(video)
camera.start()

fen.setLayout(layout)
fen.show()
app.exec()

```

Tu peux ensuite capturer des images ou enregistrer la vidéo.

Contrôles avancés du `QMediaPlayer`

Méthode	Description
<code>play()</code>	Lance la lecture
<code>pause()</code>	Met en pause
<code>stop()</code>	Arrête
<code>setSource(QUrl)</code>	Définit la source média
<code>setPosition(ms)</code>	Va à une position précise
<code>duration()</code>	Retourne la durée totale
<code>mediaStatusChanged</code>	Signale changement d'état
<code>errorOccurred</code>	Gère les erreurs (fichier introuvable, codec manquant, etc.)

Gestion des erreurs (bonne pratique)

Toujours connecter le signal `errorOccurred` :

```
def erreur(player, erreur):
    print("Erreur :", player.errorString())

self.player.errorOccurred.connect(lambda e: print("Erreur détectée :", e))
```

Mini projet : Lecteur audio complet 🎵

Un petit lecteur minimaliste avec lecture/pause/volume :

```
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton,
QVBoxLayout, QSlider
from PyQt6.QtMultimedia import QMediaPlayer, QAudioOutput
from PyQt6.QtCore import QUrl, Qt
import sys

class LecteurSimple(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Lecteur Audio Avancé")
        self.resize(300, 200)

        layout = QVBoxLayout()

        self.audio_output = QAudioOutput()
        self.player = QMediaPlayer()
        self.player.setAudioOutput(self.audio_output)
        self.player.setSource(QUrl.fromLocalFile("musique.mp3"))

        self.slider = QSlider(Qt.Orientation.Horizontal)
        self.slider.setRange(0, 100)
        self.slider.setValue(50)
        self.slider.valueChanged.connect(lambda v:
self.audio_output.setVolume(v/100))

        btn_play = QPushButton("▶ Lire")
        btn_pause = QPushButton("⏸ Pause")
        btn_stop = QPushButton("⏹ Stop")
```

```

        btn_play.clicked.connect(self.player.play)
        btn_pause.clicked.connect(self.player.pause)
        btn_stop.clicked.connect(self.player.stop)

        layout.addWidget(btn_play)
        layout.addWidget(btn_pause)
        layout.addWidget(btn_stop)
        layout.addWidget(self.slider)
        self.setLayout(layout)

app = QApplication(sys.argv)
fen = LecteurSimple()
fen.show()
app.exec()

```

Classe	Rôle
QSoundEffect	Joue un son court
QMediaPlayer	Lit des fichiers audio/vidéo
QAudioOutput	Contrôle le volume et sortie audio
QVideoWidget	Affiche la vidéo
QMediaPlaylist	Gère plusieurs fichiers multimédias
QCamera / QCameraViewfinder	Capture et affiche le flux vidéo
QMediaDevices	Liste les périphériques multimédia
errorOccurred	Détecte les erreurs

Chpitre 12 — Multithreading et Processus Parallèles dans PyQt6

Comprendre le problème du “Freezing”

PyQt6 (et Qt en général) a **une seule boucle d’événements principale (GUI Thread)**. Si tu exécutes une tâche longue (ex. boucle, téléchargement, calcul lourd) dans cette boucle, ton interface **gèle** .

Solution : déplacer les tâches lourdes dans **des threads secondaires** avec `QThread` ou `QThreadPool`.

Introduction à `QThread`

La classe `QThread` permet d’exécuter une tâche dans un **fil d’exécution séparé**.

Exemple :

```

from PyQt6.QtCore import QThread, QObject, pyqtSignal
import time

class Travailleur(QObject):
    progression = pyqtSignal(int)
    fini = pyqtSignal()

```

```

    def faire_travail(self):
        for i in range(1, 6):
            time.sleep(1)
            self.progression.emit(i * 20)
        self.fini.emit()
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout,
QPushButton, QLabel
import sys

app = QApplication(sys.argv)

class Fenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QThread avancé")
        self.label = QLabel("Progression : 0%")
        self.bouton = QPushButton("Démarrer le travail")
        self.bouton.clicked.connect(self.demarrer)

        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(self.bouton)
        self.setLayout(layout)

    def demarrer(self):
        self.thread = QThread()
        self.travailleur = Travailleur()
        self.travailleur.moveToThread(self.thread)

        # Connexions signaux / slots
        self.thread.started.connect(self.travailleur.faire_travail)
        self.travailleur.progression.connect(self.mettre_a_jour)
        self.travailleur.fini.connect(self.travail_fini)

        self.thread.start()

    def mettre_a_jour(self, valeur):
        self.label.setText(f"Progression : {valeur}%")

    def travail_fini(self):
        self.label.setText("Travail terminé ✔")
        self.thread.quit()

fen = Fenetre()
fen.show()
app.exec()

```

Explication :

- `QThread` = un fil d'exécution séparé.
- `moveToThread()` = déplace un objet dans ce thread.
- `pyqtSignal` = permet de communiquer avec le thread principal sans bloquer l'UI.

Utiliser `QRunnable` et `QThreadPool`

QThread est puissant, mais parfois lourd à gérer.

Pour les tâches parallèles simples, Qt offre **QThreadPool + QRunnable** (gestion automatique du pool de threads).

Exemple :

```
from PyQt6.QtCore import QRunnable, QThreadPool
import time

class MaTache(QRunnable):
    def run(self):
        for i in range(5):
            print(f"Tâche en cours... {i+1}")
            time.sleep(1)
        print("Tâche terminée !")

pool = QThreadPool.globalInstance()
pool.start(MaTache())
```

Tu n'as pas à gérer manuellement les threads — Qt réutilise le pool.

Communication entre threads (signaux / slots)

Quand tu veux mettre à jour l'interface depuis un thread, **tu ne dois jamais modifier directement le widget depuis ce thread** ⚠.

Utilise **les signaux** pour envoyer les données vers le thread principal.

Exemple :

```
from PyQt6.QtCore import QObject, QThread, pyqtSignal
import time

class Calculeur(QObject):
    progression = pyqtSignal(int)

    def lancer(self):
        for i in range(1, 11):
            time.sleep(0.5)
            self.progression.emit(i * 10)
```

Dans la fenêtre :

```
self.travailleur.progression.connect(self.mettre_a_jour)
```

Ainsi, la mise à jour du label se fait **dans le thread principal** → aucun bug ni blocage.

Annuler une tâche proprement

Tu peux permettre à l'utilisateur d'interrompre un thread :

```
class Travailleur(QObject):
    progression = pyqtSignal(int)
    fini = pyqtSignal()
```

```

def __init__(self):
    super().__init__()
    self._stop = False

def arreter(self):
    self._stop = True

def faire_travail(self):
    for i in range(100):
        if self._stop:
            break
        time.sleep(0.1)
        self.progression.emit(i + 1)
    self.fini.emit()

```

Et dans ton interface :

```
self.travailleur.arreter()
```

Tâches parallèles multiples avec `QThreadPool`

Exemple :

```

from PyQt6.QtCore import QRunnable, QThreadPool
import time, random

class Tache(QRunnable):
    def __init__(self, nom):
        super().__init__()
        self.nom = nom

    def run(self):
        print(f"{self.nom} démarrée")
        time.sleep(random.randint(1, 3))
        print(f"{self.nom} terminée")

pool = QThreadPool.globalInstance()
for i in range(5):
    pool.start(Tache(f"Tâche {i+1}"))

```

Cela lance plusieurs threads **en parallèle** sans blocage de l'interface.

Signaux personnalisés dans `QRunnable` (via `QObject` mixin)

```

from PyQt6.QtCore import QObject, QRunnable, pyqtSignal, QThreadPool
import time

class Signaleur(QObject):
    fini = pyqtSignal(str)

class TacheAvecSignal(QRunnable):
    def __init__(self, nom):
        super().__init__()
        self.nom = nom

```

```

        self.signaux = Signaleur()

    def run(self):
        time.sleep(2)
        self.signaux.fini.emit(f"{self.nom} terminée !")

def afficher(msg):
    print(msg)

pool = QThreadPool.globalInstance()
tache = TacheAvecSignal("Calcul")
tache.signaux.fini.connect(afficher)
pool.start(tache)

```

Différences entre QThread et QRunnable

Critère	QThread	QRunnable + QThreadPool
Gestion	Manuelle	Automatique
Communication	Signaux/slots	Signaux via QObject mixin
Recommandé pour	Longs processus continus	Petites tâches courtes et multiples
Contrôle	Total (pause, arrêt, etc.)	Moins flexible mais plus léger

Combiner threads et GUI

Tu peux, par exemple, exécuter un téléchargement de fichier tout en affichant une barre de progression animée.

Qt propose aussi des classes prêtes à l'emploi comme :

- `QFuture`
- `QFutureWatcher`
- `QtConcurrent.run()`

Mais `QThread` et `QThreadPool` restent la base pour un contrôle complet.

Élément	Classe / Méthode	Utilité
QThread	Thread complet	Gestion manuelle
QRunnable	Classe exécutable	Tâche légère
QThreadPool	Pool de threads	Exécution parallèle automatique
pyqtSignal	Communication	Échange de données entre threads
moveToThread()	Transfert d'objet	Déplace un objet dans un thread secondaire
QFuture, QtConcurrent	Haute abstraction	Multithreading simplifié

Chapitre 13 : Gestion des Bases de Données (PyQt6 + SQL)

Introduction

PyQt6 offre un module puissant :

```
from PyQt6.QtSql import *
```

Ce module permet de :

- Se connecter à des bases de données (SQLite, MySQL, PostgreSQL, etc.)
- Lire, insérer, modifier et supprimer des données
- Relier directement les données à des widgets (QTableView, QComboBox, etc.)

Connexion à une base de données SQLite

Exemple :

```
from PyQt6.QtSql import QSqlDatabase

db = QSqlDatabase.addDatabase("SQLITE")
db.setDatabaseName("etudiants.db")

if not db.open():
    print("Erreur de connexion à la base de données.")
else:
    print("Connexion réussie !")
```

💡 SQLite ne nécessite aucun serveur — parfait pour des applications locales (ex. gestion scolaire, paiements, etc.)

Création d'une table via SQL

```
from PyQt6.QtSql import QSqlQuery

query = QSqlQuery()
query.exec("""
CREATE TABLE IF NOT EXISTS etudiants (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT,
    age INTEGER,
    filiere TEXT
)
""")
```

Insérer des données (CREATE)

```
query.prepare("INSERT INTO etudiants (nom, age, filiere) VALUES (?, ?, ?)")
query.addBindValue("Mabengo")
query.addBindValue(23)
query.addBindValue("Informatique")
```

```
query.exec()
```

Lire les données (READ)

```
query.exec("SELECT * FROM etudiants")
while query.next():
    print(query.value(0), query.value(1), query.value(2), query.value(3))
```

`query.value(i)` retourne la i-ème colonne de la ligne courante.

Mettre à jour (UPDATE)

```
query.prepare("UPDATE etudiants SET filiere=? WHERE id=?")
query.addBindValue("Réseaux")
query.addBindValue(1)
query.exec()
```

Supprimer (DELETE)

```
query.prepare("DELETE FROM etudiants WHERE id=?")
query.addBindValue(1)
query.exec()
```

Afficher les données dans un `QTableView`

Qt permet d'afficher directement une table SQL dans un widget !

```
from PyQt6.QtWidgets import QApplication, QTableView
from PyQt6.QtSql import QSqlTableModel
import sys

app = QApplication(sys.argv)

model = QSqlTableModel()
model.setTable("etudiants")
model.select()

table = QTableView()
table.setModel(model)
table.show()

app.exec()
```

Tu peux éditer les cellules directement depuis le tableau !

Configuration du modèle

Tu peux préciser :

```
model.setHeaderData(1, Qt.Orientation.Horizontal, "Nom")
model.setHeaderData(2, Qt.Orientation.Horizontal, "Âge")
model.setHeaderData(3, Qt.Orientation.Horizontal, "Filière")
model.setEditStrategy(QSqlTableModel.EditStrategy.OnFieldChange)
```

Stratégies d'édition :

Stratégie	Description
OnFieldChange	Enregistre dès qu'on change une cellule
OnRowChange	Enregistre quand on quitte la ligne
OnManualSubmit	Enregistre uniquement après <code>submitAll()</code>

Utiliser un `QSqlRelationalTableModel`

Permet de gérer des **relations entre tables** (clé étrangère).

Exemple :

```
from PyQt6.QtSql import QSqlRelationalTableModel, QSqlRelation
from PyQt6.QtCore import Qt

model = QSqlRelationalTableModel()
model.setTable("paiements")
model.setRelation(1, QSqlRelation("etudiants", "id", "nom"))
model.select()

table = QTableView()
table.setModel(model)
table.show()
```

💡 Qt remplace automatiquement l'ID de l'étudiant par son nom.

Sauvegarder ou annuler les modifications

```
model.submitAll() # Enregistrer toutes les modifications
model.revertAll() # Annuler
```

Filtrer et trier les données

```
model.setFilter("filiere = 'Réseaux'")
model.setSort(1, Qt.SortOrder.AscendingOrder)
model.select()
```

Lier une table SQL à un `QComboBox`

```
from PyQt6.QtWidgets import QComboBox
from PyQt6.QtSql import QSqlQueryModel

model = QSqlQueryModel()
model.setQuery("SELECT nom FROM etudiants")

combo = QComboBox()
combo.setModel(model)
```

```
combo.setModelColumn(0)
```

Gestion d'erreurs SQL

Toujours vérifier les erreurs après `exec()` :

```
if not query.exec():
    print("Erreur SQL :", query.lastError().text())
```

Connexion à MySQL (externe)

```
db = QSqlDatabase.addDatabase("QMYSQL")
db.setHostName("localhost")
db.setDatabaseName("ecole")
db.setUserName("root")
db.setPassword("motdepasse")

if not db.open():
    print("Erreur :", db.lastError().text())
```

Nécessite le driver MySQL pour Qt (`qsqlmysql.dll` sous Windows).

Exemple complet : Interface CRUD PyQt6 + SQLite

```
from PyQt6.QtWidgets import *
from PyQt6.QtSql import *
import sys

class FenetreBDD(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Gestion des Étudiants")
        self.resize(500, 400)

        # Connexion DB
        db = QSqlDatabase.addDatabase("QSQLITE")
        db.setDatabaseName("etudiants.db")
        db.open()

        # Modèle
        self.model = QSqlTableModel()
        self.model.setTable("etudiants")

self.model.setEditStrategy(QSqlTableModel.EditStrategy.OnManualSubmit)
self.model.select()

# Vue
self.table = QTableView()
self.table.setModel(self.model)

# Boutons
self.ajouter = QPushButton("Ajouter")
self.supprimer = QPushButton("Supprimer")
```

```

self.sauvegarder = QPushButton("Enregistrer")

layout = QVBoxLayout()
layout.addWidget(self.table)
layout.addWidget(self.ajouter)
layout.addWidget(self.supprimer)
layout.addWidget(self.sauvegarder)
self.setLayout(layout)

# Événements
self.ajouter.clicked.connect(self.ajout)
self.supprimer.clicked.connect(self.suppression)
self.sauvegarder.clicked.connect(self.model.submitAll)

def ajout(self):
    self.model.insertRow(self.model.rowCount())

def suppression(self):
    self.model.removeRow(self.table.currentIndex().row())

app = QApplication(sys.argv)
win = FenetreBDD()
win.show()
app.exec()

```

□ Application CRUD complète et fonctionnelle.

Élément	Classe	Description
Connexion DB	QSqlDatabase	Connecte à SQLite/MySQL/PostgreSQL
Requête SQL	QSqlQuery	Exécute des commandes SQL
Table modèle	QSqlTableModel	Représente une table dans un modèle Qt
Table relationnelle	QSqlRelationalTableModel	Gère les relations (jointures)
Vue	QTableView	Affiche les données d'un modèle
Requêtes personnalisées	QSqlQueryModel	Affiche les résultats d'un SELECT
ComboBox liée	setModel()	Affiche des données dynamiques
Stratégie d'édition	setEditStrategy()	Gère la manière d'enregistrer

Chapitre 14 : Déploiement et distribution professionnelle d'une application PyQt6

Objectif

À la fin de ce module, tu sauras :

Compiler ton application PyQt6 en `.exe` (Windows) ou `.app` (macOS)
 Inclure des fichiers (images, icônes, bases de données...) à l'intérieur du binaire
 Créer un fichier `.qrc` (Qt Resource Collection)

Ajouter une **icône personnalisée** à ta fenêtre et à ton exécutable
Gérer les dépendances et créer un **installeur** propre

Organisation d'un projet PyQt6 prêt pour le déploiement

Exemple d'arborescence :

```
MonApp/
├── main.py
├── interface.ui
├── ressources.qrc
├── images/
│   ├── logo.png
│   └── icone.ico
├── db/
│   └── etudiants.db
└── requirements.txt
```

Garde toujours une structure claire : le dossier `images` pour tes ressources, `db` pour les bases de données, etc.

❑ 3 ❑ Créer un fichier `.qrc` (Qt Resource Collection)

Un fichier `.qrc` est un XML qui regroupe les chemins des fichiers à intégrer dans l'exécutable.

Exemple :

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource prefix="/">
    <file>images/logo.png</file>
    <file>images/icone.ico</file>
</qresource>
</RCC>
```

Compiler le fichier `.qrc` en `.py`

Qt ne lit pas directement le `.qrc`.
Tu dois le convertir avec **pyrcc6** :

```
pyrcc6 ressources.qrc -o ressources_rc.py
```

💡 Cela crée un fichier Python que tu peux importer :

```
import ressources_rc
```

Utiliser une ressource dans ton application

Une fois ton `.qrc` compilé :

```
from PyQt6.QtWidgets import QApplication, QLabel
from PyQt6.QtGui import QPixmap
import ressources_rc

app = QApplication([])
label = QLabel()
label.setPixmap(QPixmap(":/images/logo.png"))
label.show()
app.exec()
```

Les fichiers sont intégrés à ton exécutable : pas besoin du dossier images.

Ajouter une icône à ta fenêtre PyQt6

```
from PyQt6.QtGui import QIcon
from PyQt6.QtWidgets import QApplication, QWidget
import ressources_rc

app = QApplication([])
fen = QWidget()
fen.setWindowTitle("Application Mabengo")
fen.setWindowIcon(QIcon(":/images/icône.ico"))
fen.show()
app.exec()
```

Intégrer une base de données locale (.db)

Place ta base SQLite dans un dossier db/, puis utilise un chemin relatif :

```
import os
from PyQt6.QtSql import QSqlDatabase

chemin = os.path.join(os.path.dirname(__file__), "db/etudiants.db")

db = QSqlDatabase.addDatabase("SQLITE")
db.setDatabaseName(chemin)
db.open()
```

Cela garantit que la base sera trouvée même après compilation.

Compiler ton application en .exe avec PyInstaller

Installe PyInstaller :

```
pip install pyinstaller
```

Puis exécute :

```
pyinstaller --onefile --windowed --icon=images/icône.ico main.py
```

Options principales :

Option	Description
--------	-------------

<code>--onefile</code>	Génère un seul fichier <code>.exe</code>
<code>--windowed</code>	Cache la console (GUI uniquement)
<code>--icon=...</code>	Ajoute une icône
<code>--add-data</code>	Ajoute des fichiers externes
<code>--noconfirm</code>	Évite les confirmations
<code>--name</code>	Donne un nom personnalisé à ton exécutable

Inclure des fichiers supplémentaires (base, images...)

Exemple :

```
pyinstaller --onefile --windowed main.py \
    --add-data "db/etudiants.db;db" \
    --add-data "images;/images" \
    --icon=images/icône.ico
```

Sous **Windows**, le séparateur est ;

Sous **Linux/macOS**, c'est :.

Accéder à des fichiers intégrés après compilation

Quand ton programme est packagé, les chemins changent.

Utilise ceci pour être compatible **avant et après compilation** :

```
import sys, os

def chemin_relatif(chemin):
    """Retourne le bon chemin selon le mode d'exécution"""
    if hasattr(sys, '_MEIPASS'):
        return os.path.join(sys._MEIPASS, chemin)
    return os.path.join(os.path.abspath("."), chemin)
```

Puis :

```
db_path = chemin_relatif("db/etudiants.db")
```

Ajouter un splash screen (écran de chargement)

```
from PyQt6.QtWidgets import QApplication, QSplashScreen, QMainWindow
from PyQt6.QtGui import QPixmap
import time, ressources_rc, sys

app = QApplication(sys.argv)
```

```
splash = QSplashScreen(QPixmap(":/images/logo.png"))
splash.showMessage("Chargement en cours...", alignment=1)
splash.show()
time.sleep(2)

fen = QMainWindow()
fen.setWindowTitle("Application chargée")
fen.show()

splash.finish(fen)
app.exec()
```

Créer un installeur Windows professionnel

Option 1 : Inno Setup

Gratuit et très populaire.

1. Installe Inno Setup
2. Crée un script .iss :

```
[Setup]
AppName=MonApplication
AppVersion=1.0
DefaultDirName={autopf}\MonApplication
OutputDir=output
OutputBaseFilename=Setup_MonApplication

[Files]
Source: "dist\main.exe"; DestDir: "{app}"; Flags: ignoreversion

[Icons]
Name: "{group}\MonApplication"; Filename: "{app}\main.exe"
```

3. Compile avec **Inno Setup Compiler** → tu obtiens ton Setup.exe.

Version alternative : fbs (Qt-friendly packager)

```
pip install fbs
fbs startproject
```

Puis :

```
fbs run
fbs freeze
fbs installer
```

fbs est conçu spécialement pour **PyQt5/6** et produit des installeurs professionnels avec mises à jour automatiques.

Personnalisation avancée de l'exécutable

- Nom de l'application :

- `pyinstaller --name "MabengoApp" main.py`
- **Icône du programme :**
- `pyinstaller --icon=images/icône.ico main.py`
- **Splash / logo de démarrage :** via `QSplashScreen`
- **Version / métadonnées :** ajoute un fichier `version.txt` et l'intègre à ton build

Sécurisation et obfuscation du code

Ton code Python reste visible dans l'exécutable.
Tu peux le **protéger** :

- **Chiffrement du bytecode :**
- `pyinstaller --key "secret" main.py`
- **Obfuscation** (via `pyarmor`) :
- `pip install pyarmor`
- `pyarmor gen main.py`

Optimisation finale

- Supprime le dossier `build/` après la compilation
- Vérifie ton `.exe` dans `dist/`
- Teste sur un autre PC sans Python installé
- Inclue un fichier `requirements.txt` si tu veux publier ton projet sur GitHub

Élément	Outil / Classe	Rôle
Fichier .qrc	Qt Resource	Liste des fichiers intégrés
pyrcc6	Qt Compiler	Convertit <code>.qrc</code> en <code>.py</code>
PyInstaller	Packager	Crée un <code>.exe</code> ou <code>.app</code>
--add-data	Option	Intègre fichiers externes
QIcon / QPixmap	QtGui	Gère les icônes et images
QSplashScreen	QtWidgets	Crée un écran de chargement
Inno Setup / fbs	Installeurs	Distribution professionnelle

Exercice pratique :

Crée une petite application PyQt6 :

- avec ton logo et icône personnalisée,
- une base SQLite intégrée,
- un splash screen,
- et compile-la en `.exe` autonome.

Chapitre 15 : Intégration Web et API REST dans PyQt6

Objectif

À la fin de ce module, tu sauras :

Faire des requêtes HTTP (GET, POST, etc.) depuis ton interface PyQt6
Récupérer et afficher des données JSON d'une API (comme météo, devises, etc.)
Intégrer un navigateur Web dans ton application avec **QWebEngineView**
Créer une application connectée (ex : convertisseur de devise, météo, actualités)
Gérer les erreurs réseau et les chargements

Comprendre les API et HTTP

Une **API REST** (comme *OpenWeather*, *CoinGecko*, *Google Maps*, etc.) permet de récupérer ou d'envoyer des données via Internet.

Exemple d'API :

`https://api.exchangerate-api.com/v4/latest/USD`

Retourne des données **JSON** :

```
{
  "base": "USD",
  "date": "2025-10-15",
  "rates": {
    "EUR": 0.92,
    "CDF": 2800.45,
    "XAF": 600.32
  }
}
```

Installer les modules nécessaires

```
pip install PyQt6 PyQt6-WebEngine requests
```

Exemple simple de requête API (sans interface)

```
import requests

url = "https://api.exchangerate-api.com/v4/latest/USD"
response = requests.get(url)
data = response.json()

print("1 USD =", data["rates"]["CDF"], "CDF")
```

Intégrer une requête API dans une interface PyQt6

Voici un exemple complet :

```

from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout,
QPushButton, QLabel
import requests, sys

class ApiApp(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Taux de change USD - CDF")
        self.setGeometry(200, 200, 300, 150)

        self.label = QLabel("Cliquez pour actualiser le taux")
        self.bouton = QPushButton("Actualiser")
        self.bouton.clicked.connect(self.get_taux)

        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(self.bouton)
        self.setLayout(layout)

    def get_taux(self):
        try:
            url = "https://api.exchangerate-api.com/v4/latest/USD"
            response = requests.get(url)
            data = response.json()
            taux = data["rates"]["CDF"]
            self.label.setText(f"1 USD = {taux:.2f} CDF")
        except Exception as e:
            self.label.setText(f"Erreur : {str(e)}")

app = QApplication(sys.argv)
fen = ApiApp()
fen.show()
app.exec()

```

Une fenêtre s’ouvre, et lorsque tu cliques sur “**Actualiser**”, le taux de change s’affiche.

Utiliser QNetworkAccessManager pour des requêtes asynchrones (sans bloquer l’UI)

```

from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout,
QPushButton
from PyQt6.QtNetwork import QNetworkAccessManager, QNetworkRequest
from PyQt6.QtCore import QUrl, QByteArray
import json, sys

class ApiQt(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Météo via API OpenWeather")
        self.resize(350, 200)

        self.label = QLabel("Cliquez pour charger la météo")
        self.btn = QPushButton("Charger météo")
        self.btn.clicked.connect(self.get_meteo)

        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(self.btn)
        self.setLayout(layout)

```

```

        self.manager = QNetworkAccessManager()
        self.manager.finished.connect(self.reponse_api)

    def get_meteo(self):
        ville = "Kinshasa"
        cle_api = "demo" # à remplacer par ta vraie clé API
        url =
f"https://api.openweathermap.org/data/2.5/weather?q={ville}&appid={cle_api}
&units=metric&lang=fr"
        requete = QNetworkRequest(QUrl(url))
        self.manager.get(requete)

    def reponse_api(self, reply):
        donnees = json.loads(reply.readAll().data().decode())
        if "main" in donnees:
            temp = donnees["main"]["temp"]
            desc = donnees["weather"][0]["description"]
            self.label.setText(f"Météo : {desc}, {temp}°C")
        else:
            self.label.setText("Erreur ou clé API invalide")

app = QApplication(sys.argv)
fen = ApiQt()
fen.show()
app.exec()

```

Ici, **QNetworkAccessManager** gère la requête sans bloquer ton interface.

Intégrer une page Web dans ton interface (navigateur intégré)

Tu peux intégrer une page Web ou une carte Google Maps avec **QWebEngineView**.

```

from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QWidget
from PyQt6.QtWebEngineWidgets import QWebEngineView
from PyQt6.QtCore import QUrl
import sys

class Navigateur(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Navigateur PyQt6")
        self.resize(900, 600)

        self.view = QWebEngineView()
        self.view.setUrl(QUrl("https://www.google.com"))

        conteneur = QWidget()
        layout = QVBoxLayout()
        layout.addWidget(self.view)
        conteneur.setLayout(layout)
        self.setCentralWidget(conteneur)

app = QApplication(sys.argv)
win = Navigateur()
win.show()
app.exec()

```

Tu obtiens un navigateur complet intégré dans ton application.

Charger une page HTML locale

```
html = """
<html>
  <head><title>Test HTML</title></head>
  <body>
    <h1 style='color:blue'>Bienvenue dans Mabengo Web!</h1>
  </body>
</html>
"""

self.view.setHtml(html)
```

Très utile pour générer des **rapports HTML dynamiques** à partir de ton application (ex : bulletins, factures...).

Exemple d'application complète : Convertisseur de devise en ligne

```
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel,
QComboBox, QPushButton
import requests, sys

class Convertisseur(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Convertisseur de devises en ligne")
        self.resize(350, 200)

        self.label = QLabel("1 USD = ? CDF")
        self.combo = QComboBox()
        self.combo.addItem("EUR")
        self.combo.addItem("CDF")
        self.combo.addItem("XAF")
        self.combo.addItem("GBP")
        self.btn = QPushButton("Convertir")

        layout = QVBoxLayout()
        layout.addWidget(self.combo)
        layout.addWidget(self.btn)
        layout.addWidget(self.label)
        self.setLayout(layout)

        self.btn.clicked.connect(self.convertir)

    def convertir(self):
        try:
            monnaie = self.combo.currentText()
            url = "https://api.exchangerate-api.com/v4/latest/USD"
            data = requests.get(url).json()
            taux = data["rates"][monnaie]
            self.label.setText(f"1 USD = {taux:.2f} {monnaie}")
        except Exception as e:
            self.label.setText(f"Erreur : {str(e)}")

app = QApplication(sys.argv)
fen = Convertisseur()
fen.show()
```

```
app.exec()
```

Gérer le chargement (progression)

Ajoute un indicateur de chargement pendant la requête :

```
self.label.setText("Chargement...")
QApplication.processEvents()
```

Gérer les erreurs réseau

Toujours prévoir :

```
if response.status_code != 200:
    self.label.setText("Erreur : API inaccessible")
```

Appeler des API avec authentification (token, clé)

Certaines API exigent une clé :

```
headers = {"Authorization": "Bearer TON_TOKEN"}
response = requests.get(url, headers=headers)
```

Exemple : Intégration d'API d'actualités (NewsAPI)

```
url = "https://newsapi.org/v2/top-headlines?country=fr&apiKey=TA_CLE_API"
data = requests.get(url).json()

for article in data["articles"]:
    print(article["title"])
```

Tu peux afficher ces titres dans une **QListWidget** dans PyQt6.

Intégrer une carte (Google Maps ou OpenStreetMap)

Avec **QWebView** :

```
self.view.setUrl(QUrl("https://www.openstreetmap.org"))
```

Ou avec HTML local :

```
html = "<iframe
src='https://www.google.com/maps?q=Kinshasa&output=embed'></iframe>"
self.view.setHtml(html)
```

Outils et concepts clés de ce module

Élément	Description
<code>requests</code>	Requêtes HTTP simples
<code>QNetworkAccessManager</code>	Requêtes asynchrones intégrées à Qt
<code>QWebEngineView</code>	Affiche une page Web dans une fenêtre Qt
<code>QUrl</code>	Représente une URL
<code>QNetworkRequest</code>	Objet Qt pour envoyer une requête
<code>setHtml()</code>	Charge du HTML directement
<code>QApplication.processEvents()</code>	Rafraîchit l'interface pendant un chargement

Chapitre 16 — Tableaux de bord et Data Visualization avec PyQt6

Objectif

À la fin de ce module, tu sauras :

Créer des **dashboards modernes** avec `QGridLayout`, `QFrame`, et `QSplitter`

Utiliser **QTableWidget** et **QTreeWidget** pour afficher les données

Intégrer **Matplotlib** et **PyQtGraph** pour des graphiques dynamiques

Créer des **graphiques Qt natifs** avec `QtCharts`

Mettre à jour les graphiques en **temps réel**

Concevoir une **interface de suivi et d'analyse complète**

Créer la base du tableau de bord

Un tableau de bord est une fenêtre principale avec plusieurs sections (widgets) :

- En-tête
- Menu latéral
- Zone centrale (contenu)
- Statistiques / Graphiques

Exemple :

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget,
QHBoxLayout, QVBoxLayout, QPushButton, QLabel, QFrame
import sys
```

```
class Dashboard(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Tableau de Bord PyQt6")
        self.resize(900, 600)
```

```
# --- Cadre principal ---
principal = QWidget()
layout = QHBoxLayout()
principal.setLayout(layout)
```

```

# --- Menu latéral ---
menu = QFrame()
menu.setStyleSheet("background-color:#2C3E50; color:white;")
menu_layout = QVBoxLayout()
menu.setLayout(menu_layout)
for btn_text in ["Accueil", "Rapports", "Statistiques",
"Paramètres"] :
    b = QPushButton(btn_text)
    b.setStyleSheet("background:none; color:white; font-
size:16px;")
    menu_layout.addWidget(b)

# --- Contenu principal ---
contenu = QFrame()
contenu.setStyleSheet("background-color:#ECF0F1;")
contenu_layout = QVBoxLayout()
contenu_layout.addWidget(QLabel("Tableau de bord principal"))
contenu.setLayout(contenu_layout)

layout.addWidget(menu, 1)
layout.addWidget(contenu, 4)
self.setCentralWidget(principal)

app = QApplication(sys.argv)
win = Dashboard()
win.show()
app.exec()

```

💡 Ce code crée un layout typique d'un tableau de bord professionnel.

Afficher des données tabulaires avec `QTableWidget`

```

from PyQt6.QtWidgets import QTableWidget, QTableWidgetItem

table = QTableWidget()
table.setRowCount(4)
table.setColumnCount(3)
table.setHorizontalHeaderLabels(["Nom", "Âge", "Filière"])

data = [
    ("Mabengo", "23", "Informatique"),
    ("Aline", "21", "Réseaux"),
    ("Yves", "22", "IA"),
    ("Sarah", "20", "Cybersécurité"),
]

for i, (nom, age, filiere) in enumerate(data):
    table.setItem(i, 0, QTableWidgetItem(nom))
    table.setItem(i, 1, QTableWidgetItem(age))
    table.setItem(i, 2, QTableWidgetItem(filiere))

```

Intégrer QtCharts (QChart, QChartView, QPieSeries, QBarSeries)

Installe d'abord :

```
pip install PyQt6-Charts
```

Exemple d'un graphique à barres :

```
from PyQt6.QtCharts import QChart, QChartView, QBarSet, QBarSeries
from PyQt6.QtCore import Qt

set0 = QBarSet("2024")
set0 << 10 << 30 << 20 << 15

set1 = QBarSet("2025")
set1 << 20 << 25 << 40 << 30

series = QBarSeries()
series.append(set0)
series.append(set1)

chart = QChart()
chart.addSeries(series)
chart.setTitle("Statistiques des étudiants")
chart.setAnimationOptions(QChart.AnimationOption.AllAnimations)

chartview = QChartView(chart)
chartview.setRenderHint(chartview.renderHints() | chartview.renderHints())
```

Ajouter un graphique circulaire (camembert)

```
from PyQt6.QtCharts import QPieSeries

series = QPieSeries()
series.append("Informatique", 40)
series.append("Réseaux", 25)
series.append("IA", 20)
series.append("Cybersécurité", 15)

chart = QChart()
chart.addSeries(series)
chart.setTitle("Répartition des filières")

chartview = QChartView(chart)
```

Tu peux intégrer `chartview` dans ton layout principal.

⚡ 5 Intégrer Matplotlib à PyQt6

Installe :

```
pip install matplotlib
```

Exemple :

```
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAff as
FigureCanvas
import matplotlib.pyplot as plt

class Graphique(FigureCanvas):
    def __init__(self):
        fig, ax = plt.subplots()
```

```
ax.plot([1,2,3,4], [10, 30, 20, 25], marker='o')
ax.set_title("Évolution des inscriptions")
super().__init__(fig)
```

Puis ajoute `Graphique()` dans ton `QVBoxLayout()` pour afficher le graphe.

Intégrer PyQtGraph pour graphiques dynamiques (temps réel)

Installe :

```
pip install pyqtgraph
```

Exemple :

```
import pyqtgraph as pg
from PyQt6.QtCore import QTimer

graph = pg.PlotWidget()
x = list(range(10))
y = [2,4,1,6,9,8,7,5,3,2]
line = graph.plot(x, y)

def update():
    y.append(y[-1] + pg.np.random.randint(-3,3))
    y.pop(0)
    line.setData(x, y)

timer = QTimer()
timer.timeout.connect(update)
timer.start(500)
```

Idéal pour afficher des **données en temps réel** (ex : température, CPU, réseau...).

Créer un Dashboard complet combinant table et graphiques

```
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout
from PyQt6.QtCharts import QChart, QChartView, QPieSeries
from PyQt6.QtWidgets import QTableWidgetItem, QTableWidgetItem
import sys

class TableauDeBord(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Dashboard Étudiants")
        self.resize(900, 500)

        layout = QHBoxLayout()

        # Tableau
        table = QTableWidgetItem()
```

```

table.setRowCount(3)
table.setColumnCount(2)
table.setHorizontalHeaderLabels(["Filière", "Nombre"])
data = [("Informatique", 40), ("Réseaux", 25), ("IA", 35)]
for i, (filiere, n) in enumerate(data):
    table.setItem(i, 0, QTableWidgetItem(filiere))
    table.setItem(i, 1, QTableWidgetItem(str(n)))

# Graphique
series = QPieSeries()
for filiere, n in data:
    series.append(filiere, n)

chart = QChart()
chart.addSeries(series)
chart.setTitle("Répartition des étudiants")
chartview = QChartView(chart)

layout.addWidget(table, 2)
layout.addWidget(chartview, 3)
self.setLayout(layout)

app = QApplication(sys.argv)
win = TableauDeBord()
win.show()
app.exec()

```

Interface professionnelle, données tabulaires + graphique synchronisé.

Mise en page avancée avec `QSplitter` et `QStackedWidget`

Exemple :

```

from PyQt6.QtWidgets import QSplitter, QStackedWidget

splitter = QSplitter()
splitter.addWidget(menu)
splitter.addWidget(contenu)

pages = QStackedWidget()
pages.addWidget(page_dashboard)
pages.addWidget(page_graphiques)

```

Te permet de naviguer entre plusieurs sections sans ouvrir de nouvelles fenêtres.

❑ 9 Styliser ton dashboard (feuilles de style Qt)

```

self.setStyleSheet("""
MainWindow { background-color: #F2F3F4; }
QPushButton {
    background-color: #3498DB; color: white; border-radius: 5px;
    padding: 8px; font-size:14px;
}
QPushButton:hover { background-color: #2980B9; }
QTableWidget { background:white; border-radius:6px; }
""")

```

Rafraîchir automatiquement les données (ex : API ou capteurs)

```
from PyQt6.QtCore import QTimer

def rafraichir():
    # Appel API ou mise à jour de graphique
    pass

timer = QTimer()
timer.timeout.connect(rafraichir)
timer.start(5000) # toutes les 5 secondes
```

Concepts clés

Élément	Description
<code>QChartView</code>	Conteneur de graphique Qt
<code>QBarSeries, QPieSeries</code>	Différents types de graphiques
<code>QTableWidget</code>	Table manuelle (idéal pour CRUD local)
<code>pyqtgraph.PlotWidget</code>	Graphiques en temps réel
<code>matplotlib</code>	Graphiques scientifiques
<code>QSplitter</code>	Division dynamique de zones
<code>QStackedWidget</code>	Gestion de plusieurs pages dans la même fenêtre

Exercice pratique

Crée un **Dashboard complet** pour la **gestion de paiement académique** :

- Table des étudiants avec montants payés / restants
- Graphique en barres pour le total par filière
- Graphique circulaire pour le pourcentage de paiement effectué
- Rafraîchissement automatique toutes les 10 secondes

Chapitre 17 : Gestion Multithreading, Réseau et Communication entre Processus

Objectifs

À la fin de ce module, tu sauras :

- Créer des applications PyQt6 **multithreadées** sans bloquer l'interface graphique.
- Utiliser les **sockets réseau (TCP/UDP)** pour échanger des données.
- Communiquer entre threads et processus avec les **signaux et files d'attente (QThread, QMutex, QSemaphore, etc.)**.
- Implémenter des **applications client-serveur** en PyQt6.

1. Le problème de blocage dans PyQt6

Quand tu exécutes une opération lourde (ex. téléchargement, calcul, requête réseau) dans le **thread principal**, ton interface devient figée.

Solution : utiliser des **threads secondaires** (via `QThread` ou `QRunnable`).

2. Classe `QThread`

Exemple de base :

```
from PyQt6.QtCore import QThread, pyqtSignal
from PyQt6.QtWidgets import QApplication, QLabel, QPushButton, QVBoxLayout,
QWidget
import time

class Worker(QThread):
    progress = pyqtSignal(int)  # Signal émis pendant le travail

    def run(self):
        for i in range(1, 6):
            time.sleep(1)
            self.progress.emit(i)

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.label = QLabel("Clique sur Démarrer")
        self.button = QPushButton("Démarrer")
        self.button.clicked.connect(self.start_thread)

        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(self.button)
        self.setLayout(layout)

    def start_thread(self):
        self.worker = Worker()
        self.worker.progress.connect(self.update_label)
        self.worker.start()

    def update_label(self, value):
        self.label.setText(f"Progression : {value}/5")

app = QApplication([])
window = Window()
window.show()
app.exec()
```

Points clés :

- `run()` → méthode exécutée dans le thread secondaire.
- `pyqtSignal` → communication entre thread et interface.
- `start()` → lance le thread.

3. Classe `QRunnable` et `QThreadPool`

`QRunnable` permet de lancer plusieurs tâches légères sans créer manuellement des threads.

```
from PyQt6.QtCore import QRunnable, QThreadPool
import time

class Task(QRunnable):
    def run(self):
        print("Tâche démarrée")
        time.sleep(3)
        print("Tâche terminée")

pool = QThreadPool.globalInstance()
pool.start(Task())
```

Propriétés :

- `QThreadPool.maxThreadCount()`
- `QThreadPool.activeThreadCount()`
- `QThreadPool.start(QRunnable)`

4. Synchronisation : `QMutex`, `QSemaphore`, `QWaitCondition`

Exemple `QMutex` :

```
from PyQt6.QtCore import QMutex, QThread
import time

mutex = QMutex()

class MyThread(QThread):
    def run(self):
        global mutex
        mutex.lock()
        print(f"{self.currentThread()} entre en section critique")
        time.sleep(2)
        print(f"{self.currentThread()} sort de la section critique")
        mutex.unlock()
```

5. Réseau avec `QTcpServer` et `QTcpSocket`

Exemple simple Client/Serveur TCP :

Serveur :

```
from PyQt6.QtNetwork import QTcpServer, QHostAddress

class Server(QTcpServer):
    def __init__(self):
        super().__init__()
        self.listen(QHostAddress.Any, 5000)
        self.newConnection.connect(self.on_new_connection)

    def on_new_connection(self):
        client = self.nextPendingConnection()
```

```

        client.readyRead.connect(lambda:
print(client.readAll().data().decode()))

server = Server()

```

Client :

```

from PyQt6.QtNetwork import QTcpSocket

client = QTcpSocket()
client.connectToHost("127.0.0.1", 5000)
client.write(b"Bonjour serveur PyQt6")

```

6. Communication inter-threads (Signaux personnalisés)

Exemple :

```

from PyQt6.QtCore import QObject, pyqtSignal, QThread
import time

class Worker(QObject):
    finished = pyqtSignal(str)

    def do_work(self):
        time.sleep(2)
        self.finished.emit("Travail terminé")

thread = QThread()
worker = Worker()
worker.moveToThread(thread)
thread.started.connect(worker.do_work)
worker.finished.connect(lambda msg: print(msg))
thread.start()

```

7. Threads et GUI

Important :

Tu ne dois jamais modifier des widgets **depuis un thread secondaire**.
 Passe toujours par les **signaux** pour mettre à jour l'interface.

8. Cas pratique : Téléchargement asynchrone

```

import requests
from PyQt6.QtCore import QThread, pyqtSignal
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton, QLabel,
QVBoxLayout

class DownloadThread(QThread):
    done = pyqtSignal(str)

    def run(self):
        url = "https://example.com"
        data = requests.get(url).text

```

```

        self.done.emit(f"Téléchargement de {len(data)} caractères terminé")

class App(QWidget):
    def __init__(self):
        super().__init__()
        self.label = QLabel("Prêt à télécharger")
        self.button = QPushButton("Télécharger")
        self.button.clicked.connect(self.download)
        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(self.button)
        self.setLayout(layout)

    def download(self):
        self.thread = DownloadThread()
        self.thread.done.connect(self.show_result)
        self.thread.start()

    def show_result(self, msg):
        self.label.setText(msg)

app = QApplication([])
window = App()
window.show()
app.exec()

```

9. Points clés à retenir

- `QThread` : exécution parallèle lourde.
- `QRunnable` : tâches légères via `QThreadPool`.
- `QMutex`, `QSemaphore` : contrôle de l'accès simultané.
- `QTcpSocket`, `QTcpServer` : communication réseau.
- Les **signaux et slots** sont le cœur de la communication entre threads.

Chapitre 18 : Connexion Internet, API REST, WebSockets et Requêtes Réseau

Objectifs

À la fin de ce module, tu sauras :

Intégrer des **requêtes HTTP (GET, POST, PUT, DELETE)** dans ton application PyQt6.

Consommer une **API REST (JSON)** directement depuis PyQt.

Utiliser `QNetworkAccessManager` pour les connexions asynchrones.

Travailler avec des **WebSockets** pour la communication temps réel.

Afficher les données reçues dans des widgets (`QTableWidget`, `QLabel`, etc.).

1. Le module `PyQt6.QtNetwork`

PyQt6 inclut un module réseau puissant pour gérer :

- les connexions HTTP/HTTPS,
- les sockets TCP/UDP,
- les requêtes REST,
- les WebSockets.

Les classes principales :

Classe	Description
QNetworkAccessManager	Envoie les requêtes HTTP/HTTPS
QNetworkRequest	Contient les informations d'une requête
QNetworkReply	Reçoit la réponse
QWebSocket	Communication bidirectionnelle en temps réel
QHostInfo, QNetworkInterface	Informations sur le réseau local

2. Faire une requête HTTP (GET)

```
from PyQt6.QtNetwork import QNetworkAccessManager, QNetworkRequest
from PyQt6.QtCore import QUrl
from PyQt6.QtWidgets import QApplication, QLabel, QVBoxLayout, QWidget
```

```
class MyApp(QWidget):
    def __init__(self):
        super().__init__()
        self.label = QLabel("Chargement des données...")
        layout = QVBoxLayout()
        layout.addWidget(self.label)
        self.setLayout(layout)

        self.network = QNetworkAccessManager()
        self.network.finished.connect(self.handle_response)
        url = QUrl("https://jsonplaceholder.typicode.com/todos/1")
        request = QNetworkRequest(url)
        self.network.get(request)

    def handle_response(self, reply):
        data = reply.readAll().data().decode()
        self.label.setText(data)
```

```
app = QApplication([])
window = MyApp()
window.show()
app.exec()
```

Ici, la méthode **asynchrone** évite de bloquer l'interface.

`finished` → signal émis à la fin de la requête.

3. Envoyer une requête POST

```
from PyQt6.QtNetwork import QNetworkAccessManager, QNetworkRequest
from PyQt6.QtCore import QUrl, QByteArray
import json
```

```
url = QUrl("https://jsonplaceholder.typicode.com/posts")
request = QNetworkRequest(url)
request.setHeader(QNetworkRequest.KnownHeaders.ContentTypeHeader,
"application/json")
```

```

data = json.dumps({
    "title": "PyQt6 POST",
    "body": "Envoi de données depuis PyQt6",
    "userId": 1
}).encode()

manager = QNetworkAccessManager()
reply = manager.post(request, QByteArray(data))

```

4. Gestion des erreurs réseau

```

def handle_response(self, reply):
    if reply.error():
        self.label.setText(f"Erreur : {reply.errorString()}")
    else:
        data = reply.readAll().data().decode()
        self.label.setText(data)

```

5. Consommer une API REST et afficher les données JSON

```

import json
from PyQt6.QtWidgets import QApplication, QTableWidgetItem, QTableWidgetItem,
QVBoxLayout, QWidget
from PyQt6.QtNetwork import QNetworkAccessManager, QNetworkRequest
from PyQt6.QtCore import QUrl

class ApiApp(QWidget):
    def __init__(self):
        super().__init__()
        self.table = QTableWidgetItem()
        layout = QVBoxLayout()
        layout.addWidget(self.table)
        self.setLayout(layout)

        self.manager = QNetworkAccessManager()
        self.manager.finished.connect(self.load_data)

        url = QUrl("https://jsonplaceholder.typicode.com/users")
        self.manager.get(QNetworkRequest(url))

    def load_data(self, reply):
        data = json.loads(reply.readAll().data().decode())
        self.table.setRowCount(len(data))
        self.table.setColumnCount(3)
        self.table.setHorizontalHeaderLabels(["ID", "Nom", "Email"])
        for i, user in enumerate(data):
            self.table.setItem(i, 0, QTableWidgetItem(str(user["id"])))
            self.table.setItem(i, 1, QTableWidgetItem(user["name"]))
            self.table.setItem(i, 2, QTableWidgetItem(user["email"]))

app = QApplication([])
window = ApiApp()
window.show()
app.exec()

```

Résultat : une table affichant les utilisateurs récupérés via HTTP.

6. Utiliser les WebSockets (communication en temps réel)

```
from PyQt6.QtWebSockets import QWebSocket
from PyQt6.QtCore import QUrl
from PyQt6.QtWidgets import QApplication, QLabel, QVBoxLayout, QWidget

class SocketApp(QWidget):
    def __init__(self):
        super().__init__()
        self.label = QLabel("Connexion au serveur WebSocket...")
        layout = QVBoxLayout()
        layout.addWidget(self.label)
        self.setLayout(layout)

        self.socket = QWebSocket()
        self.socket.connected.connect(self.on_connected)
        self.socket.textMessageReceived.connect(self.on_message)
        self.socket.open(QUrl("wss://echo.websocket.org"))

    def on_connected(self):
        self.label.setText("Connecté au serveur")
        self.socket.sendMessage("Salut depuis PyQt6 !")

    def on_message(self, msg):
        self.label.setText(f"Message reçu : {msg}")

app = QApplication([])
window = SocketApp()
window.show()
app.exec()
```

7. Classe `QNetworkReply` (propriétés importantes)

Propriété / Méthode	Description
<code>readAll()</code>	Lit la réponse complète
<code>error()</code>	Retourne l'erreur éventuelle
<code>errorString()</code>	Message d'erreur
<code>attribute()</code>	Accède aux en-têtes HTTP
<code>isFinished()</code>	Vérifie si la requête est terminée

8. Téléchargement de fichier avec barre de progression

```
from PyQt6.QtCore import QUrl
from PyQt6.QtWidgets import QApplication, QWidget, QProgressBar,
QVBoxLayout
from PyQt6.QtNetwork import QNetworkAccessManager, QNetworkRequest
```

```

class Downloader(QWidget):
    def __init__(self):
        super().__init__()
        self.progress = QProgressBar()
        layout = QVBoxLayout()
        layout.addWidget(self.progress)
        self.setLayout(layout)

        url = QUrl("https://speed.hetzner.de/100MB.bin")
        request = QNetworkRequest(url)

        self.manager = QNetworkAccessManager()
        self.reply = self.manager.get(request)
        self.reply.downloadProgress.connect(self.update_progress)

    def update_progress(self, received, total):
        if total > 0:
            self.progress.setValue(int(received * 100 / total))

app = QApplication([])
window = Downloader()
window.show()
app.exec()

```

9. Bonnes pratiques professionnelles

Toujours **vérifier l'état du réseau** avant une requête.

Ne jamais bloquer l'UI avec une opération réseau toujours utiliser `QNetworkAccessManager`.

Pour les API sécurisées utiliser les **en-têtes HTTP (Token, Authorization)**.

Les requêtes multiples peuvent être gérées avec un **QThreadPool**.

En production, loggue les erreurs via `QLoggingCategory`.

10. Exercice pratique final du module

Crée une application PyQt6 qui :

1. Se connecte à une API (par ex. <https://jsonplaceholder.typicode.com/posts>).
2. Affiche les 10 premiers titres dans une liste (`QListWidget`).
3. En cliquant sur un titre, affiche le contenu complet du post (`QTextEdit`).
4. Possède un bouton "Actualiser" pour recharger les données.

Thème	Contenu
Requêtes HTTP	GET, POST, PUT, DELETE
API REST	Lecture et affichage JSON
WebSockets	Communication bidirectionnelle

Téléchargement	Fichiers et progression
Gestion erreurs	Vérification et logs

Chapitre 19 : Intégration complète avec les Bases de Données (SQLite, MySQL, PostgreSQL)

Objectifs du module

À la fin de ce module, tu sauras :

Connecter ton application PyQt6 à **SQLite, MySQL ou PostgreSQL**.

Créer, lire, modifier et supprimer (CRUD) des données depuis PyQt6.

Lier les résultats à des widgets comme `QTableView`, `QComboBox`, `QLineEdit`.

Utiliser **`QSqlDatabase`, `QSqlQuery`, `QSqlTableModel`, `QSqlRelationalTableModel`**.

Gérer les transactions et la sécurité SQL.

1. Module PyQt6.QtSql

Classes principales :

Classe	Description
<code>QSqlDatabase</code>	Gère la connexion à la base de données
<code>QSqlQuery</code>	Exécute les requêtes SQL
<code>QSqlTableModel</code>	Représente une table SQL dans un modèle Qt
<code>QSqlRelationalTableModel</code>	Gère les relations entre tables
<code>QSqlQueryModel</code>	Modèle basé sur une requête personnalisée
<code>QSqlError</code>	Gère les erreurs SQL

2. Connexion à SQLite

```
from PyQt6.QtSql import QSqlDatabase, QSqlQuery

# Crée une connexion
db = QSqlDatabase.addDatabase("SQLITE")
db.setDatabaseName("etudiants.db")

if not db.open():
    print("Erreur de connexion :", db.lastError().text())
else:
    print("Connexion réussie")

# Crée une table
query = QSqlQuery()
query.exec("""
CREATE TABLE IF NOT EXISTS etudiants (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```

        nom TEXT,
        prenom TEXT,
        age INTEGER
    )
    """)

```

3. Insertion de données

```

query = QSqlQuery()
query.prepare("INSERT INTO etudiants (nom, prenom, age) VALUES (?, ?, ?)")
query.addBindValue("Mabengo")
query.addBindValue("Yasine")
query.addBindValue(23)
query.exec()

```

4. Lecture des données

```

query.exec("SELECT * FROM etudiants")
while query.next():
    id = query.value(0)
    nom = query.value(1)
    prenom = query.value(2)
    age = query.value(3)
    print(id, nom, prenom, age)

```

5. Affichage avec `QTableView` et `QSqlTableModel`

```

from PyQt6.QtWidgets import QApplication, QTableView
from PyQt6.QtSql import QSqlTableModel

app = QApplication([])

db = QSqlDatabase.addDatabase("QSQLITE")
db.setDatabaseName("etudiants.db")
db.open()

model = QSqlTableModel()
model.setTable("etudiants")
model.select()

view = QTableView()
view.setModel(model)
view.show()

app.exec()

```

Avantage : `QSqlTableModel` se met à jour automatiquement après insertion/suppression.

6. Ajout, modification et suppression depuis `QSqlTableModel`

```

# Ajout d'une ligne
record = model.record()
record.setValue("nom", "Kalala")
record.setValue("prenom", "Marie")

```

```
record.setValue("age", 21)
model.insertRecord(-1, record)

# Suppression
model.removeRow(0)
model.submitAll()
```

7. Transactions (sécurité des données)

```
db.transaction()
try:
    query.exec("UPDATE etudiants SET age = 25 WHERE id = 1")
    query.exec("INSERT INTO etudiants (nom, prenom, age) VALUES ('Nono',
'Paul', 19)")
    db.commit()
except:
    db.rollback()
```

`commit()` valide toutes les requêtes,
`rollback()` annule tout en cas d'erreur.

8. Connexion à MySQL

```
db = QSqlDatabase.addDatabase("QMYSQL")
db.setHostName("localhost")
db.setDatabaseName("universite")
db.setUserName("root")
db.setPassword("motdepasse")

if db.open():
    print("Connecté à MySQL !")
else:
    print("Erreur :", db.lastError().text())
```

Nécessite `mysqlclient` ou `PyMySQL` installé et les pilotes Qt MySQL disponibles.

9. Connexion à PostgreSQL

```
db = QSqlDatabase.addDatabase("QPSQL")
db.setHostName("localhost")
db.setDatabaseName("ecole")
db.setUserName("postgres")
db.setPassword("motdepasse")
db.open()
```

10. Requêtes personnalisées avec `QSqlQueryModel`

```
from PyQt6.QtSql import QSqlQueryModel
from PyQt6.QtWidgets import QTableView

model = QSqlQueryModel()
model.setQuery("SELECT nom, prenom, age FROM etudiants WHERE age > 20")

view = QTableView()
view.setModel(model)
```

```
view.show()
```

11. Relations entre tables (`QSqlRelationalTableModel`)

Exemple :

- Table **etudiants**(id, nom, filiere_id)
- Table **filieres**(id, nom_filiere)

```
from PyQt6.QtSql import QSqlRelationalTableModel, QSqlRelation

model = QSqlRelationalTableModel()
model.setTable("etudiants")
model.setRelation(2, QSqlRelation("filieres", "id", "nom_filiere"))
model.select()
```

Qt affichera automatiquement le **nom de la filière** au lieu de l'ID.

12. Liaison des champs SQL avec des widgets (`QDataWidgetMapper`)

```
from PyQt6.QtWidgets import QLineEdit, QVBoxLayout, QWidget
from PyQt6.QtSql import QDataWidgetMapper

class Formulaire(QWidget):
    def __init__(self, model):
        super().__init__()
        self.nom = QLineEdit()
        self.prenom = QLineEdit()

        layout = QVBoxLayout()
        layout.addWidget(self.nom)
        layout.addWidget(self.prenom)
        self.setLayout(layout)

        mapper = QDataWidgetMapper()
        mapper.setModel(model)
        mapper.addMapping(self.nom, 1)
        mapper.addMapping(self.prenom, 2)
        mapper.toFirst()
```

13. Bonnes pratiques

- ✓ Toujours tester `db.isOpen()` avant chaque requête.
- ✓ Ne jamais injecter de texte brut dans une requête SQL → utiliser `prepare()` et `bindValue()`.
- ✓ Fermer proprement la base (`db.close()`) à la fin.
- ✓ Gérer les erreurs avec `db.lastError()` et `query.lastError()`.
- ✓ Pour les grandes bases, utiliser des **requêtes paginées** (LIMIT/OFFSET).

14. Mini-projet pratique

Créer une application de gestion d'étudiants :

1. Base SQLite avec les tables :
 - o `etudiants(id, nom, prenom, age, filiere_id)`
 - o `filieres(id, nom_filiere)`
2. Interface PyQt6 :
 - o `QTableView` pour la liste.
 - o Formulaire (`QLineEdit` + `QComboBox`) pour ajouter/modifier.
 - o Boutons “Ajouter”, “Modifier”, “Supprimer”.
3. Liaison via `QSqlRelationalTableModel` et `QDataWidgetMapper`.

Objectif : une gestion **entièrement dynamique** et synchronisée avec la base.

15. Résumé du module

Thème	Contenu
Connexions	SQLite, MySQL, PostgreSQL
Classes principales	<code>QSqlDatabase</code> , <code>QSqlQuery</code> , <code>QSqlTableModel</code>
Modèles Qt	<code>TableModel</code> , <code>RelationalModel</code> , <code>QueryModel</code>
Transactions	<code>commit</code> , <code>rollback</code>
Liaison	<code>QDataWidgetMapper</code>
Sécurité	<code>prepare()</code> , <code>bindValue()</code>

Chapitre 20 : Architecture Professionnelle, MVC, Fichiers de Configuration, Logs & Création d'un exécutable (.exe)

Objectifs

À la fin de ce module, tu sauras :

Concevoir une application PyQt6 structurée selon le modèle **MVC (Model-View-Controller)**.

Gérer la **configuration de ton application** avec des fichiers `.ini` ou `.json`.

Mettre en place un système de **journalisation (logs)** professionnel.

Créer un **exécutable Windows (.exe)** avec `PyInstaller`.

Organiser ton projet PyQt6 pour être propre, scalable et maintenable.

1. Architecture MVC dans PyQt6

Pourquoi utiliser MVC ?

- **Model** : gère les données et la logique (SQLite, API, fichiers, etc.)

- **View** : affiche les données (fenêtres, widgets, formulaires).
- **Controller** : fait le lien entre Model et View.

Structure professionnelle d'un projet PyQt6

```

projet_pyqt6/
├── main.py
├── controllers/
│   └── main_controller.py
├── models/
│   ├── database_model.py
│   └── user_model.py
├── views/
│   ├── main_window.ui
│   └── main_window.py
├── config/
│   └── settings.ini
├── logs/
│   └── app.log
└── requirements.txt

```

2. Exemple d'architecture MVC complète

models/database_model.py

```

from PyQt6.QtSql import QSqlDatabase, QSqlQuery

class DatabaseModel:
    def __init__(self, db_name="app.db"):
        self.db = QSqlDatabase.addDatabase("SQLITE")
        self.db.setDatabaseName(db_name)
        self.db.open()
        self.init_tables()

    def init_tables(self):
        query = QSqlQuery()
        query.exec("""
        CREATE TABLE IF NOT EXISTS utilisateurs (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            nom TEXT,
            email TEXT
        )
        """)

    def ajouter_utilisateur(self, nom, email):
        query = QSqlQuery()
        query.prepare("INSERT INTO utilisateurs (nom, email) VALUES (?,
?)")
        query.addBindValue(nom)
        query.addBindValue(email)
        return query.exec()

```

views/main_window.py

```
from PyQt6.QtWidgets import QWidget, QVBoxLayout, QPushButton, QLineEdit,
QLabel
```

```
class MainView(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Application MVC PyQt6")

        self.nom = QLineEdit()
        self.nom.setPlaceholderText("Nom")
        self.email = QLineEdit()
        self.email.setPlaceholderText("Email")
        self.bouton = QPushButton("Ajouter utilisateur")
        self.resultat = QLabel()

        layout = QVBoxLayout()
        layout.addWidget(self.nom)
        layout.addWidget(self.email)
        layout.addWidget(self.bouton)
        layout.addWidget(self.resultat)
        self.setLayout(layout)
```

controllers/main_controller.py

```
from models.database_model import DatabaseModel
from views.main_window import MainView

class MainController:
    def __init__(self):
        self.model = DatabaseModel()
        self.view = MainView()

        self.view.bouton.clicked.connect(self.ajouter_utilisateur)

    def ajouter_utilisateur(self):
        nom = self.view.nom.text()
        email = self.view.email.text()
        if self.model.ajouter_utilisateur(nom, email):
            self.view.resultat.setText("Utilisateur ajouté avec succès ✔")
        else:
            self.view.resultat.setText("Erreur lors de l'ajout ✖")

    def show(self):
        self.view.show()
```

main.py

```
from PyQt6.QtWidgets import QApplication
from controllers.main_controller import MainController

app = QApplication([])
controller = MainController()
controller.show()
app.exec()
```

3. Fichiers de configuration (settings.ini)

Exemple de fichier :

```
[APP]
theme = dark
language = fr

[DATABASE]
db_name = app.db
```

Lecture et écriture avec configparser

```
import configparser

config = configparser.ConfigParser()
config.read("config/settings.ini")

theme = config["APP"]["theme"]
db_name = config["DATABASE"]["db_name"]

print(f"Thème : {theme}, Base : {db_name}")

# Modifier et sauvegarder
config["APP"]["theme"] = "light"
with open("config/settings.ini", "w") as f:
    config.write(f)
```

4. Fichiers de configuration au format JSON

```
import json

data = {
    "theme": "dark",
    "language": "fr",
    "database": "app.db"
}

with open("config/settings.json", "w") as f:
    json.dump(data, f, indent=4)

# Lecture
with open("config/settings.json", "r") as f:
    config = json.load(f)

print(config["theme"])
```

5. Journalisation (Logging) professionnelle

```
import logging
import os

# Crée le dossier logs si inexistant
os.makedirs("logs", exist_ok=True)

logging.basicConfig(
    filename="logs/app.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s"
)

logging.info("Application démarrée")
```

```
logging.warning("Attention : Base de données non trouvée")
logging.error("Erreur critique")
```

Utilisation dans le contrôleur :

```
import logging

logging.info(f"Nouvel utilisateur : {nom} - {email}")
```

6. Gestion des erreurs globales avec `sys.excepthook`

```
import sys
import logging

def handle_exception(exc_type, exc_value, exc_traceback):
    logging.error("Erreur non gérée", exc_info=(exc_type, exc_value,
exc_traceback))

sys.excepthook = handle_exception
```

7. Création d'un exécutable (.exe)

Avec PyInstaller

Installation :

```
pip install pyinstaller
```

Création de l'exécutable :

```
pyinstaller --noconsole --onefile --icon=icon.ico main.py
```

- `--noconsole` : cache la console.
- `--onefile` : tout dans un seul .exe.
- `--icon=icon.ico` : ajoute une icône personnalisée.

L'exécutable se trouve dans :

```
dist/main.exe
```

8. Inclusion des ressources (icônes, images, fichiers .ui)

Utilise le **Qt Resource System (.qrc)** :

Exemple `resources.qrc`

```
<RCC>
  <qresource prefix="/images">
    <file>icon.png</file>
  </qresource>
</RCC>
```

Compilation :

```
pyrcc6 resources.qrc -o resources_rc.py
```

Utilisation :

```
self.setWindowIcon(QIcon(":/images/icon.png"))
```

9. Organisation professionnelle (récapitulatif)

Élément	Dossier recommandé
Interface (UI)	/views/
Logique	/controllers/
Base de données	/models/
Ressources	/resources/
Fichiers de config	/config/
Logs	/logs/

10. Mini-projet final du module

Crée une **application PyQt6 complète** de gestion de notes :

1. Architecture MVC complète.
2. Configuration stockée dans un fichier `.ini`.
3. Base SQLite (table : `notes (id, titre, contenu, date)`).
4. Journalisation des opérations dans `logs/app.log`.
5. Génération d'un **.exe** portable avec ton icône personnalisée.

Thème	Contenu
Architecture	MVC propre et modulaire
Configurations	<code>.ini</code> et <code>.json</code>
Journalisation	logging professionnel
Gestion erreurs	<code>sys.excepthook</code>
Exécutable	PyInstaller
Organisation	Dossiers structurés