



BOOMERANG
CONSULTING

FORMATION

IT - Digital - Management

24/03/2024

Bienvenue sur cette formation

Boomerang Consulting

- Formateur : Timothée Demares
- Développeur et formateur python depuis 3 ans
- Téléphone : 0762457593
- Email : timothee.demares@gmail.com
- Certification 2Ai Concept Expert Trainer

Fort de 20 années d'expertises dans la formation professionnelle continue, notamment en informatique, Boomerang Consulting est un organisme qui cherche à vous offrir la qualité optimum pour vos formations avec :

- Un accueil de qualité et de proximité,
- Un service commercial, administratif et pédagogique à votre disposition,
- Un positionnement technologique étendu avec des formateurs experts, qualifiés et certifiés pour certains,
- Une qualité de services et une réponse à vos besoins grâce aux contenus pédagogiques **SUR-MESURE**.



- L'engagement pour la qualité en étant certifié Qualiopi.
- Centre de formation spécialisé dans les domaines de l'IT, Ressources humaines et développement personnel.

- Des formateurs certifiés et expérimentés.
- Un taux de satisfaction client supérieur à 93% suite aux évaluations de fin de stage.

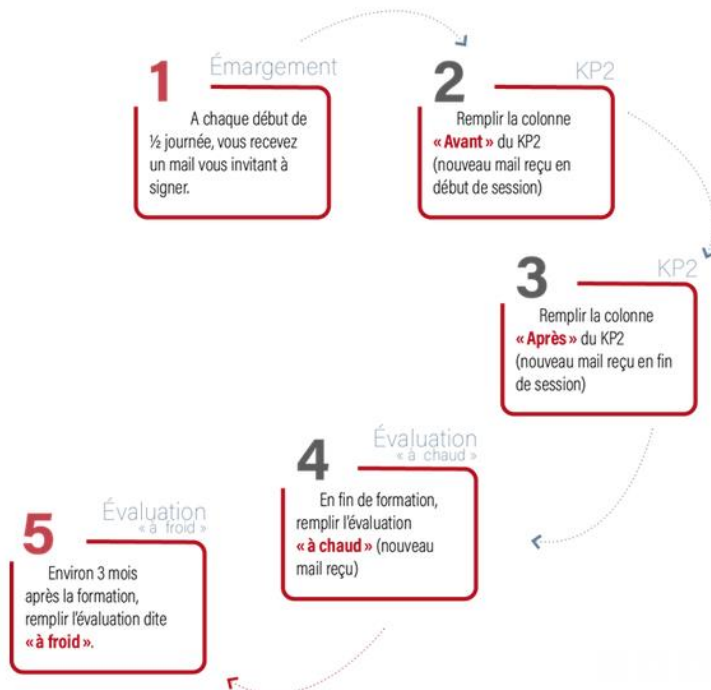
- 9h - 17h
- 15 mn de pause le matin
- 1h de pause déjeuner
- 15 mn de pause l'après midi
- Dernier jour à 16h30 (si le plan de cours est terminé uniquement)

4 documents sont à remplir :

1. La feuille d'émargements (en chaque début de demi-journée)
2. Le KP2 (avant la formation puis après la formation)
3. L'évaluation à chaud (après la formation)
4. L'évaluation à froid (environ 3 mois après la formation)

Voici schématisées ci-dessous les différentes étapes (1-2-3-4)

L'étape 5 se réalisera entre 3 et 6 mois après le training.



- Base Python
- POO
- Bases html
- Bases css
- Droits admin
- Sur quels IDE avez-vous l'habitude de travailler ?
- Quelles versions de python pouvez vous utiliser ?
- Est-ce vous connaissez le design pattern MVC ou MVP ?
- Avez-vous l'habitude d'utiliser l'IA dans votre quotidien de développeur ?
- Quelles sont vos attentes à l'issue de cette formation ?

A l'issue de cette formation, vous serez capable de :

- Créer une application python avec PyQt6
- Intégrer html et css dans une application Qt
- Créer, positionner des composants Qt et les rendre interactif
- Mettre en œuvre une architecture logicielle adaptée à Qt
- Utiliser Qt Designer

Matin et après midi

- Première chose à faire : signer les feuilles d'émargement. Merci de bien vouloir faire des signatures conformes.
- Les croix et initiales ne sont pas autorisées.

- Dernier jour de la formation : le centre de formation à l'obligation contractuelle de fournir vos évaluations à votre entreprise avant 15h, donc au retour de la pause déjeuner, 2 ou 3 h avant la fin de la formation, je vous ferai remplir les évaluations formateur.

- Equilibre Théorie / Pratique, environ 50/50.
- Un cours classique :
 - Présentation des objectifs
 - Cours théoriques avec exemples
 - TP à faire individuellement ou en groupe
 - Correction
 - Résumé des points clefs à retenir

- Support de cours en lien avec un contenu Github
- Lien github: <https://github.com/orgs/CoursQtTdemares/repositories>
- 1 notion = 1 chapitre = 1 comit Github

Support de cours et outils pédagogiques

- Qualiopi : fournir le support de cours et les énoncés des tp de validation des acquis.
- Les énoncés des TP de validation des acquis sont présents sur ces slides.
- Les corrigés seront sur Github ou sur un Drive et accessibles en fin de formation.

- Google Forms de demi-journée pour la validation des acquis et l'adaptabilité.

A chaque fin de demi journée, les stagiaires devront répondre aux questions suivantes par l'intermédiaire d'un Google Form :

- Le rythme vous convient il ?
- L'équilibre théorie pratique vous convient il ?
- Les notions abordées jusque là sont elles acquises ?
- Y a t'il des notions sur lesquelles revenir avant de poursuivre le cours ?

- Chapitre 1 : Introduction
- Chapitre 2 : Principes généraux
- Chapitre 3 : Stratégies de placement
- Chapitre 4 : Qt designer
- Chapitre 5 : Architecture mvc
- Chapitre 6 : Aspects avancés
- TP de validation des acquis

Valider le démarrage de la formation

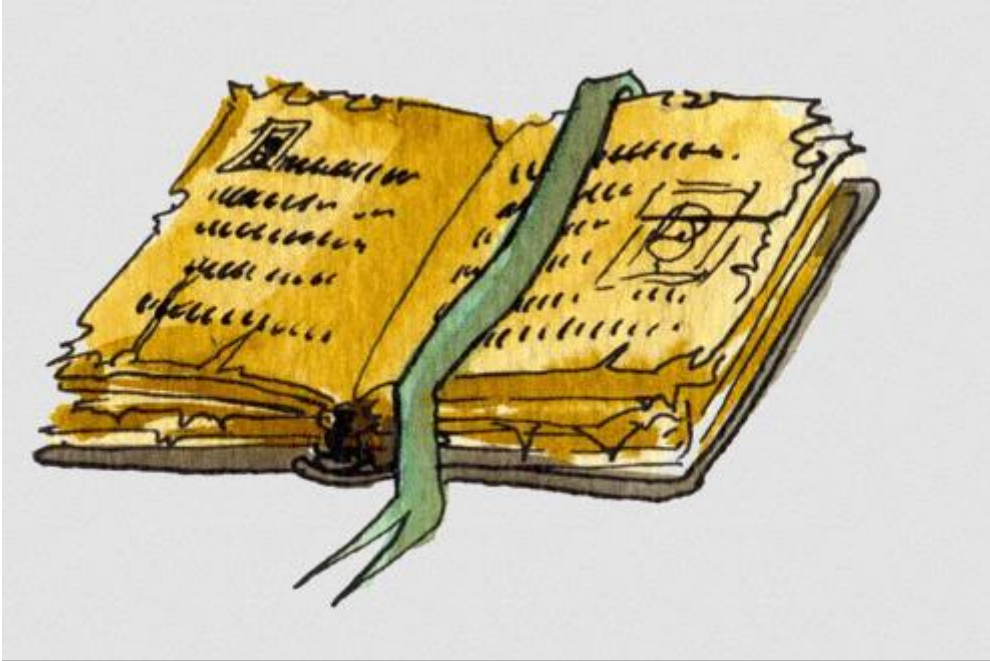
- Confirmer le démarrage de la formation
- Mail à envoyer le premier jour de la formation au plus tard à 10h30.

Chapitre 1 : Introduction

Objectifs de ce chapitre

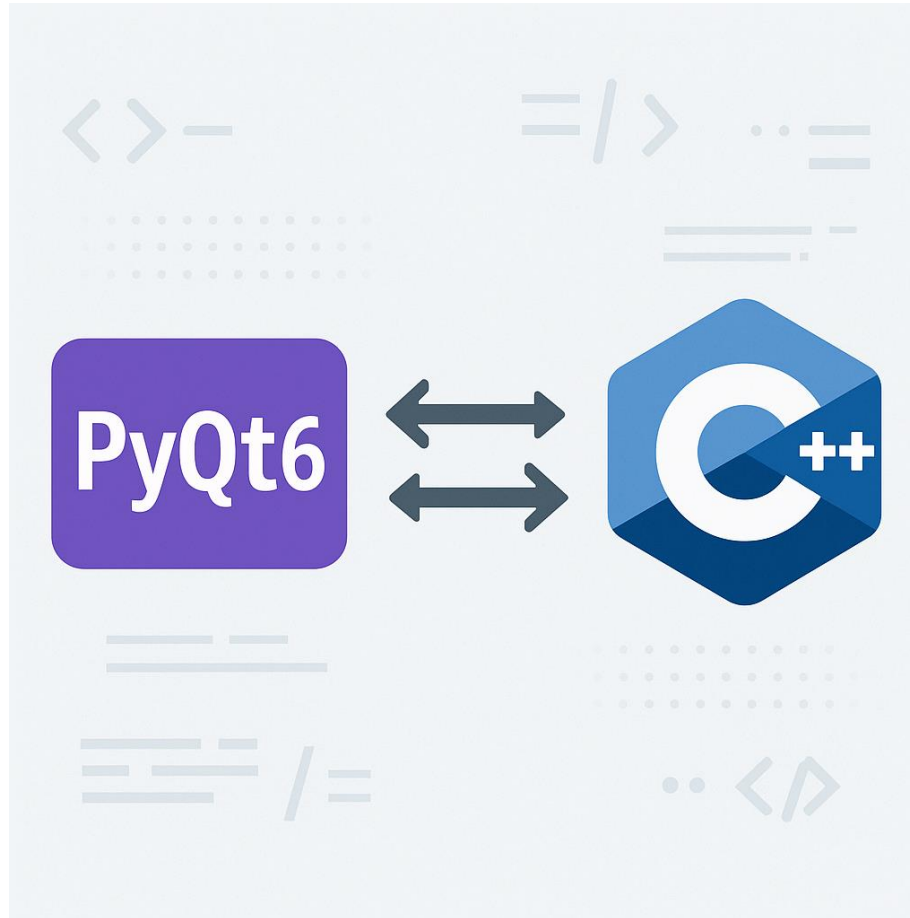
- Comprendre l'évolution historique des interfaces graphiques
- Situer Qt dans le contexte historique des frameworks GUI
- Comprendre ce qu'est Qt et le rôle du binding PyQt6
- Distinguer PyQt6 de ses alternatives (PyQt5, PySide6)
- Installer et configurer un environnement de développement PyQt6
- Utiliser VSCode pour développer des applications Qt
- Naviguer efficacement dans la documentation Qt/PyQt6
- Créer votre première application Qt fonctionnelle

1.1 Contexte historique



- Paradigme WIMP
- Histoire de Qt
- Applications célèbres

1.2 Binding



1.3 Fonctionnement de base

```
Premier pas pyqt6

import sys

from PyQt6.QtWidgets import QApplication, QLabel

# Un seul QApplication par processus
app = QApplication(sys.argv)

# ... création des widgets ...
label = QLabel("Hello, PyQt6!")
label.show() # Un label par exemple

# Démarrage de la boucle d'événements
sys.exit(app.exec())
```

1.4 Modules principaux de PyQt6

QtCore

- Event loops
- Fichiers threads
- Signaux / slots

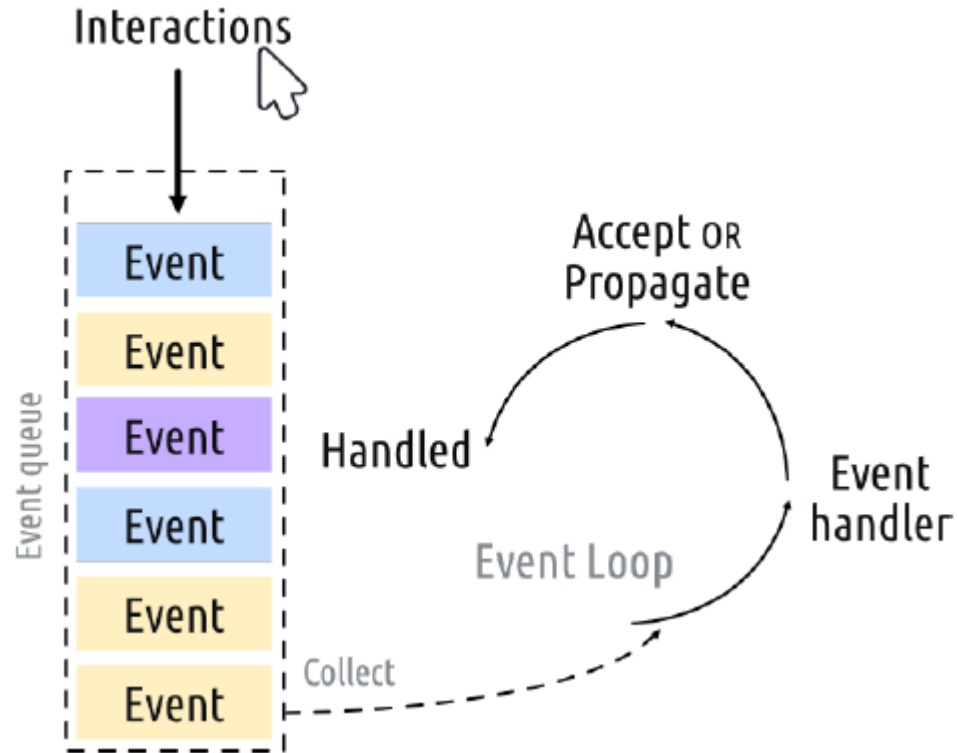
QtGui

- Images, polices
- Dessin
- Couleurs, styles

QtWidgets

- Fenêtres, boutons, zones de texte
- Layout, placements

1.6 Event loop





Alternatives à PyQt6

 **PySide6** **PyQt5**

Différentes versions

```
# PyQt5 (ancien style)
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication

app = QApplication([])
label.setAlignment(Qt.AlignCenter) # Enum non scopé
app.exec_() # Ancienne méthode

# PyQt6 (style moderne)
from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import QApplication, QLabel

app = QApplication([])
label = QLabel("Hello, PyQt6!")
label.setAlignment(Qt.AlignmentFlag.AlignCenter) # Enum scopé
label.show()
app.exec() # Nouvelle méthode

# PySide6 (style moderne)
from PySide6.QtCore import Qt
from PySide6.QtWidgets import QApplication, QLabel

app = QApplication([])
label = QLabel("Hello, PySide6!")
label.setAlignment(Qt.AlignmentFlag.AlignCenter)
label.show()
app.exec()
```

[Lien TP1](#)

TP1 - Installation et validation de l'environnement

Objectif : Disposer d'un environnement Python fonctionnel avec uv, exécuter un mini script PyQt6 et valider que VSCode détecte l'interpréteur.

- 1) Créer l'espace de travail du TP
- 2) Installer uv (gestionnaire Python rapide)
- 3) Initialiser un projet minimal
- 4) Ajouter PyQt6 comme dépendance
- 5) Créer un premier script de validation

[Lien TP2](#)

TP2 - Configuration VSCode et premier debug

Durée : 20 minutes

Objectif : Configurer VSCode pour Python/uv, poser un point d'arrêt, déboguer un script PyQt6 et inspecter une variable.

Pré-requis : TP1 réussi (projet uv avec script hello_qt).

- 1) Extensions VSCode indispensables
- 2) Sélection de l'interpréteur
- 3) Créer une configuration de débogage
- 4) Poser un breakpoint et démarrer le débogueur
- 5) Pas à pas et Watches
- 6) Télémétrie rapide

Documentation officielle

PyQt6 (Riverbank Computing)

<https://www.riverbankcomputing.com/static/Docs/PyQt6/>

Qt 6 Documentation (C++)

<https://doc.qt.io/qt-6/>

Ressources d'apprentissage

<https://www.pythonguis.com/>

<https://doc.qt.io/qt-6/examples-widgets.html>

<https://realpython.com/python-pyqt-gui-calculator/>

[Lien TP3](#)

TP3 - Navigation dans la documentation

Durée : 15 minutes

Objectif : Savoir trouver rapidement une information dans la documentation Qt/PyQt et l'appliquer à un cas simple.

Pré-requis : Avoir un projet PyQt6 fonctionnel (TP1) et VSCode configuré (TP2).

- 1) Identifier les bonnes ressources
- 2) Trouver une propriété utile de QLabel
- 3) Trouver un signal d'intérêt sur un bouton
- 4) Comprendre la hiérarchie des classes
- 5) Mini challenge doc

[Lien TP4](#)

TP4 - Première application personnalisée

Durée : 30 minutes

Objectif : Assembler une petite fenêtre PyQt6 avec un layout, un champ de saisie, un bouton et une zone d'affichage, en respectant une structure de projet simple pilotée par uv.

Pré-requis : TP1 à TP3.

- 1) Structure minimale
- 2) Disposition
- 3) Comportement
- 4) Petites finitions
- 5) Point d'entrée et exécution
- 6) Test manuel

Introduction : ce qu'il faut retenir

✓ Concepts essentiels

- Qt est un framework C++ multiplateforme, PyQt6 en est le binding Python
- Une seule instance `QApplication` par processus
- La boucle d'événements gère les interactions utilisateur
- Les signaux/slots permettent la communication entre composants

✓ Bonnes pratiques

- Toujours terminer par `sys.exit(app.exec())`
- Utiliser un environnement virtuel pour l'isolation
- Préférer PyQt6 pour les nouveaux projets
- Consulter la documentation Qt C++ puis transposer

✓ Configuration VSCode

- Sélectionner le bon interpréteur Python (celui du venv)
- Configurer le debug avec `launch.json`
- Installer les extensions Python appropriées

📖 Ressources indispensables

- Documentation PyQt6 : API Python spécifique
- Documentation Qt 6 : Référence complète
- Python GUIs : Tutoriels pratiques

C'est le moment de versionner :

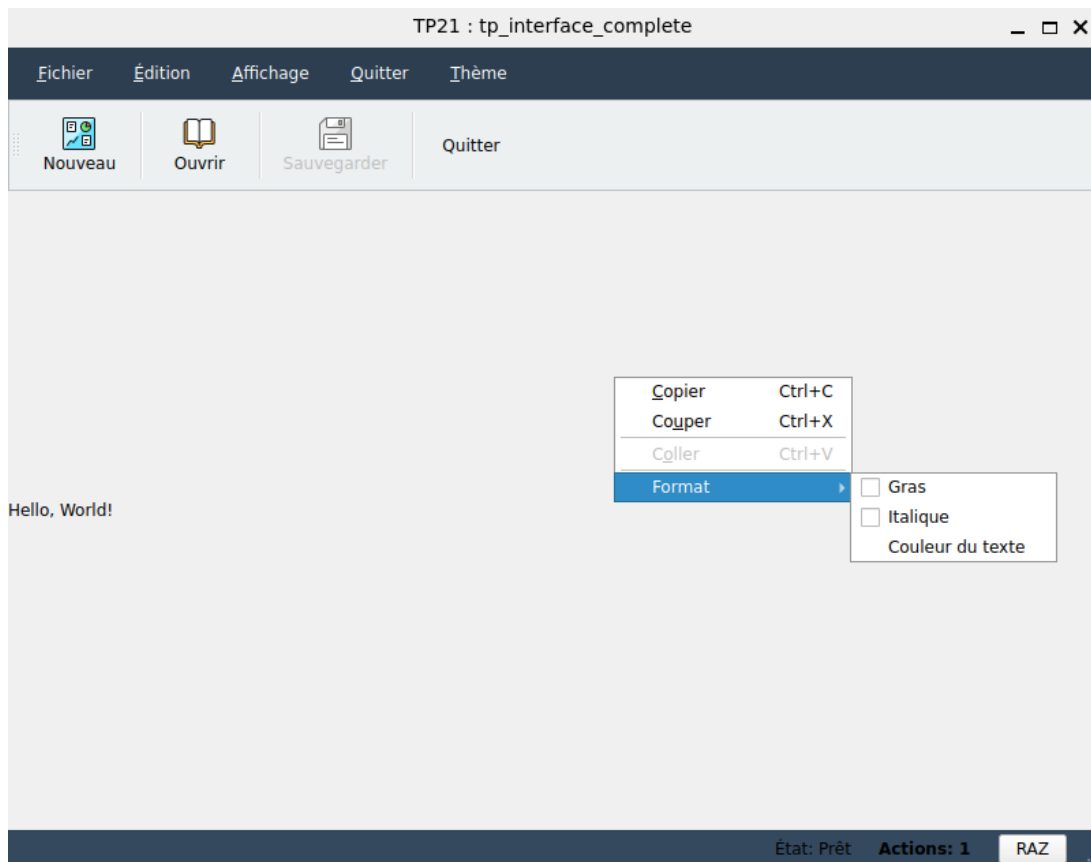
- <https://github.com/CoursQtTdemares>
- Le commentaire du commit est “chapitre 1 Introduction”

Chapitre 2 : Principes généraux

Objectifs de ce chapitre

- Structurer une application Qt complète avec fenêtre principale
- Créer et organiser des barres de menus, d'outils et de statut
- Intégrer des styles CSS pour personnaliser l'apparence
- Implémenter des menus contextuels interactifs
- Gérer l'interconnexion entre les différents éléments d'interface

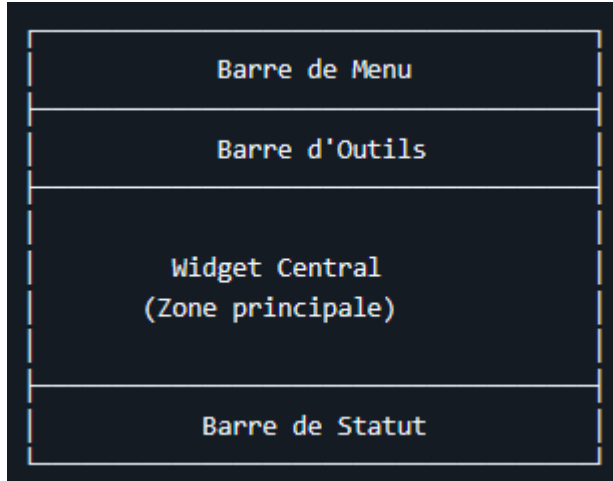
2.1 But à atteindre



Fondation de notre application Qt



2.3 Les 4 zones principales



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

class MainWindow(QMainWindow):
    def __init__(self) -> None:
        super().__init__()
        self.setWindowTitle("Les 4 zones de QMainWindow")
        self.setGeometry(100, 100, 600, 400)

        self.setup_central_widget()
        self.setup_menu_bar()
        self.setup_tool_bar()
        self.setup_status_bar()

    def setup_central_widget(self) -> None:
        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        layout = QVBoxLayout()
        central_widget.setLayout(layout)

        self.label = QLabel("👋 Bienvenue dans l'architecture Qt !")
        self.label.setAlignment(Qt.AlignmentFlag.AlignCenter)

        self.button = QPushButton("Tester les interactions")
        self.button.clicked.connect(self.handle_button_click)

        layout.addWidget(self.label)
        layout.addWidget(self.button)

    def setup_menu_bar(self) -> None:
        if (menubar := self.menuBar()) is not None:
            file_menu = menubar.addMenu("&Fichier")
            file_menu.addAction("Nouveau").triggered.connect(self.handle_new_file)
            file_menu.addAction("Ouvrir").triggered.connect(self.handle_open_file)

    def setup_tool_bar(self) -> None:
        if (toolbar := self.addToolBar("Principal")) is not None:
            toolbar.addAction("Nouveau").triggered.connect(self.handle_toolbar_new)

    def setup_status_bar(self) -> None:
        if (status_bar := self.statusBar()) is not None:
            status_bar.showMessage("Application prête")
```

2.4 Cycle de vie

```
Cycle de vie

import sys
from PyQt6.QtWidgets import QApplication

def main() -> int:
    # 1. Création de l'application
    app = QApplication(sys.argv)

    # 2. Configuration globale (optionnel)
    app.setApplicationName("Mon App")
    app.setApplicationVersion("1.0")

    # 3. Création de la fenêtre principale
    window = MainWindow()

    # 4. Affichage de la fenêtre
    window.show()

    # 5. Démarrage de la boucle d'événements
    return app.exec()

if __name__ == "__main__":
    sys.exit(main())
```

2.5 Communication par signaux et slots

```
Si:ple exemple singal/slot sur un bouton

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

class MainWindow(QMainWindow):
    def __init__(self) -> None:
        super().__init__()
        self.setWindowTitle("Mon Application")

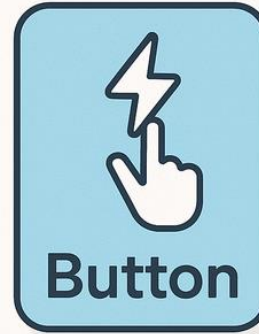
        # Créer un bouton
        button = QPushButton("Appuyez sur moi !")

        # CONNEXION : signal 'clicked' -> slot 'handle_click'
        button.clicked.connect(self.handle_click)

        self.setCentralWidget(button)

    def handle_click(self) -> None:
        """Slot personnalisé qui reçoit le signal clicked"""
        print("👉 Bouton cliqué !")

app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```



2.6 Les widgets de base

QLineEdit	QPushButton	QLabel	QCheckBox	QComboBox
saisie de texte	Déclencher des actions	Informar l'utilisateur	Les options à cocher	Choisir dans une liste


```
actions bad

# ❌ Duplication du code - MAUVAISE approche
def setup_naive_interface(self) -> None:
    # Menu
    menu_save = self.file_menu.addAction("Sauvegarder")
    menu_save.triggered.connect(self.save_document)

    # Barre d'outils
    toolbar_save = QPushButton("Save")
    toolbar_save.clicked.connect(self.save_document)
    self.toolbar.addWidget(toolbar_save)

    # Raccourci clavier - code séparé !
    shortcut = QShortcut(QKeySequence("Ctrl+S"), self)
    shortcut.activated.connect(self.save_document)
```

```
actions good

from PyQt6.QtGui import QAction, QKeySequence

def setup_smart_interface(self) -> None:
    # ✅ UNE SEULE définition pour TOUTE l'interface
    self.save_action = QAction("&Sauvegarder", self)
    self.save_action.setShortcut("Ctrl+S")
    self.save_action.setStatusTip("Sauvegarder le document")
    self.save_action.triggered.connect(self.save_document)

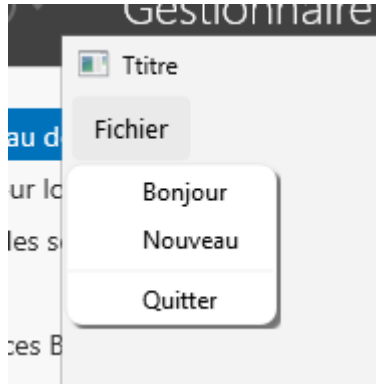
    # Maintenant on peut utiliser cette action PARTOUT :
    self.file_menu.addAction(self.save_action) # Menu
    self.toolbar.addAction(self.save_action)   # Barre d'outils
    # Le raccourci est automatiquement géré !
```

2.8 Barres d'outils (QToolBar)



- Accès rapide
- Efficace
- Visible
- Personnalisable

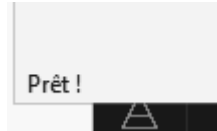
2.9 Barres de menus (QMenuBar)



Navigation principale

- Exhaustif
- Organisation par catégorie

2.10 Barre de statut



- Information d'état
- Messages temporaires

[Lien TP1](#)

TP1 - Application avec interface complète

Durée : 45 minutes

Objectif : Créer une application PyQt6 structurée avec QMainWindow, incluant des barres de menus, d'outils et de statut, et implémenter les actions de base (Nouveau, Ouvrir, Sauvegarder).

Pré-requis : Chapitre 1 terminé et environnement PyQt6 fonctionnel.

- 1) Créer la structure du projet
- 2) Classe MainWindow héritant de QMainWindow
- 3) Barre de menus avec menu "Fichier"
- 4) Barre d'outils synchronisée
- 5) Barre de statut informative
- 6) Gestion de l'état des actions
- 7) Messages d'aide contextuelle

Ce que ca permet de faire

- Couleurs
- Typographie
- Espacement
- ... (Comme le web)

Avantages

- Séparation des responsabilités
- Styles uniformes dans l'application
- Avoir plusieurs thèmes

[Lien TP2](#)

TP2 - Personnalisation avec CSS

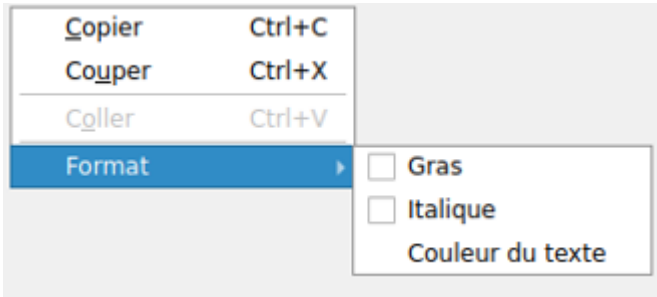
Durée : 30 minutes

Objectif : Appliquer des styles CSS personnalisés à une application PyQt6 et implémenter un système de thèmes dynamiques (clair/sombre).

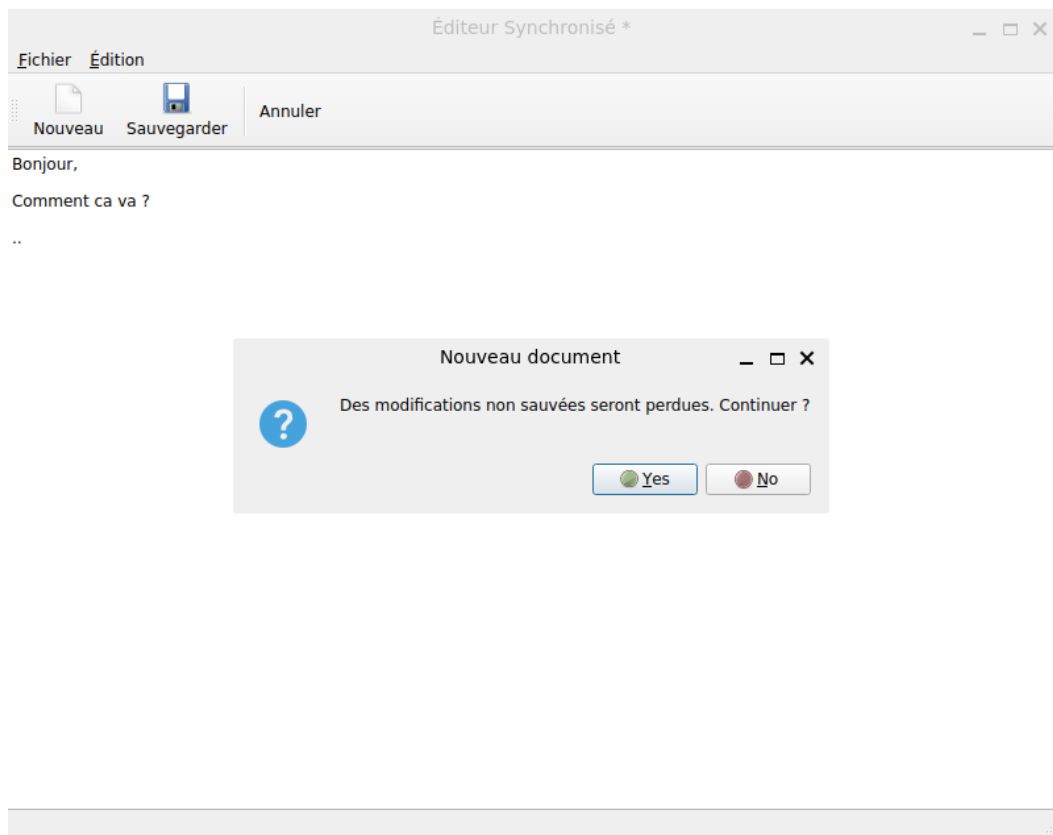
Pré-requis : TP1 terminé avec une application QMainWindow fonctionnelle.

- 1) Fichier CSS externe
 - 2) Styles de base pour QMainWindow
 - 3) Personnalisation de la barre de menus
 - 4) Style de la barre d'outils
 - 5) Système de thèmes
 - 6) Thème sombre complet
- Styles pour la barre de statut

2.12 Menus contextuels



- S'ouvre sur un clic droit
- Selon la zone cliquée
- Exécute des actions



[Lien TP3](#)

TP3 - Menus contextuels avancés

Durée : 20 minutes

Objectif : Créer des menus contextuels intelligents qui s'adaptent au contexte d'utilisation et gérer différents types d'interactions utilisateur.

Pré-requis : TP1 et TP2 terminés avec une interface stylisée.

- 1) Activation des menus contextuels
- 2) Menu contextuel de base
- 3) Logique conditionnelle
- 4) Sous-menus contextuels
- 5) Menus contextuels différenciés
- 6) Séparateurs et organisation
- 7) Raccourcis dans les menus contextuels

[Lien TP4](#)

TP4 - Synchronisation des composants

Durée : 25 minutes

Objectif : Interconnecter les différents éléments d'interface (menus, barres d'outils, actions) et implémenter un système de communication par signaux personnalisés.

Pré-requis : TP1, TP2 et TP3 terminés avec une application complète.

- 1) Actions partagées entre composants
- 2) Menu Affichage avec contrôles d'interface
- 3) Communication bidirectionnelle
- 4) Système de notifications interne

Introduction : ce qu'il faut retenir

✓ Architecture d'application

- `QMainWindow` structure l'interface en zones logiques
- Le widget central est obligatoire
- Les barres (menus, outils, statut) sont optionnelles mais recommandées

✓ Bonnes pratiques d'interface

- Partager les `QAction` entre menus et barres d'outils
- Utiliser des raccourcis clavier cohérents
- Fournir des messages d'aide dans la barre de statut
- Gérer l'état des actions (activé/désactivé)

✓ Personnalisation visuelle

- CSS permet une personnalisation avancée de l'apparence
- Qt supporte un sous-ensemble de HTML 4 pour le contenu riche
- Les thèmes améliorent l'expérience utilisateur

✓ Interaction utilisateur

- Menus contextuels adaptés au contexte
- Barres d'outils pour l'accès rapide aux fonctions courantes
- Barre de statut pour le feedback en temps réel

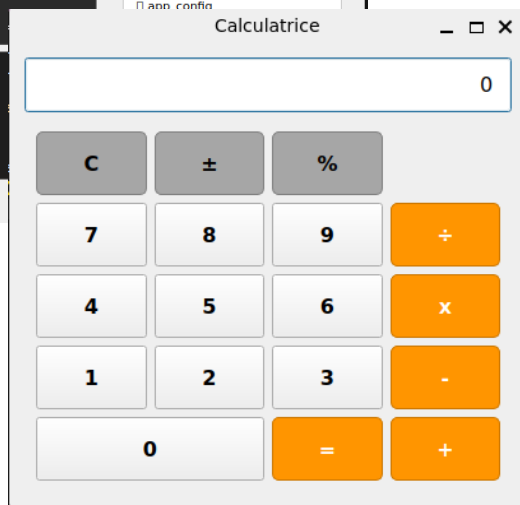
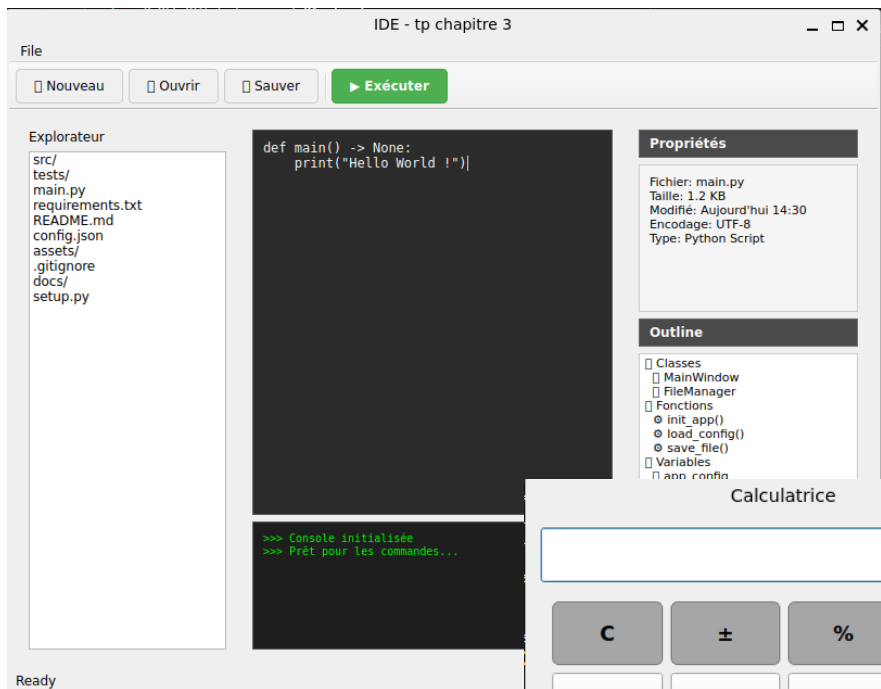
C'est le moment de versionner :

- <https://github.com/CoursQtTdemares>
- Le commentaire du commit est “chapitre 2 principes généraux”

Chapitre 3 : Stratégies placement

Objectifs de ce chapitre

- Organiser les widgets avec les layouts horizontaux, verticaux et en grille
- Maîtriser les layouts imbriqués et les techniques avancées
- Gérer les politiques de taille et l'espacement des widgets
- Créer des interfaces adaptatives qui s'ajustent automatiquement
- Appliquer les bonnes pratiques d'organisation d'interface



Formulaire - tp chapitre 3

Nom :

Prénom :

Email :

Rue :

Code postal :

Ville :

Pays :

Age :

Newsletter : ☐

Commentaire :

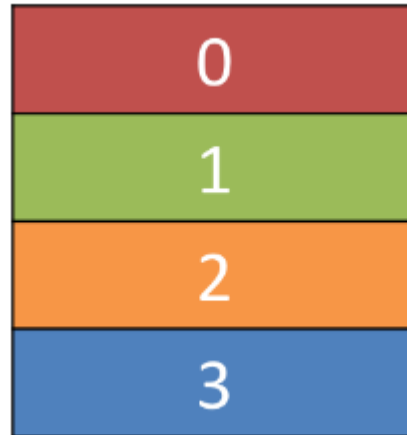


QHBoxLayout	QVBoxLayout	QGridLayout	QFormLayout
Horizontale	Verticale	Grille	Étiquette-champ
Barres d'outils, boutons alignés	Formulaires, menus, listes	Formulaires complexes, calculatrices	Saisie de données structurées

QHBoxLayout



QVBoxLayout



[Lien TP1](#)

TP1 - Formulaire avec layouts de base

Durée : 30 minutes

Objectif : Créer un formulaire d'inscription complet en utilisant les layouts verticaux et horizontaux, avec validation des entrées utilisateur.

Pré-requis : Maîtrise des bases PyQt6 (chapitres 1-2).

- 1) Créer la structure du projet
- 2) Créer la fenêtre principale
- 3) Section informations personnelles
- 4) Section adresse
- 5) Section préférences
- 6) Boutons d'action
- 7) Validation simple

QGridLayout

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

[Lien TP2](#)

TP2 - Interface en grille avancée

Durée : 30 minutes

Objectif : Concevoir une calculatrice fonctionnelle avec QGridLayout, en gérant les widgets multi-cellules et l'espacement.

Pré-requis : TP1 terminé et concepts des layouts de base acquis.

- 1) **Projet calculatrice**
- 2) **Fenêtre et affichage**
- 3) **Grille des boutons principaux**
- 4) **Bouton zéro étendu**
- 5) **Colonne des opérateurs**
- 6) **Ligne de fonctions**
- 7) **Logique de base**
- 8) **Fonction Clear**

Layouts imbriqués



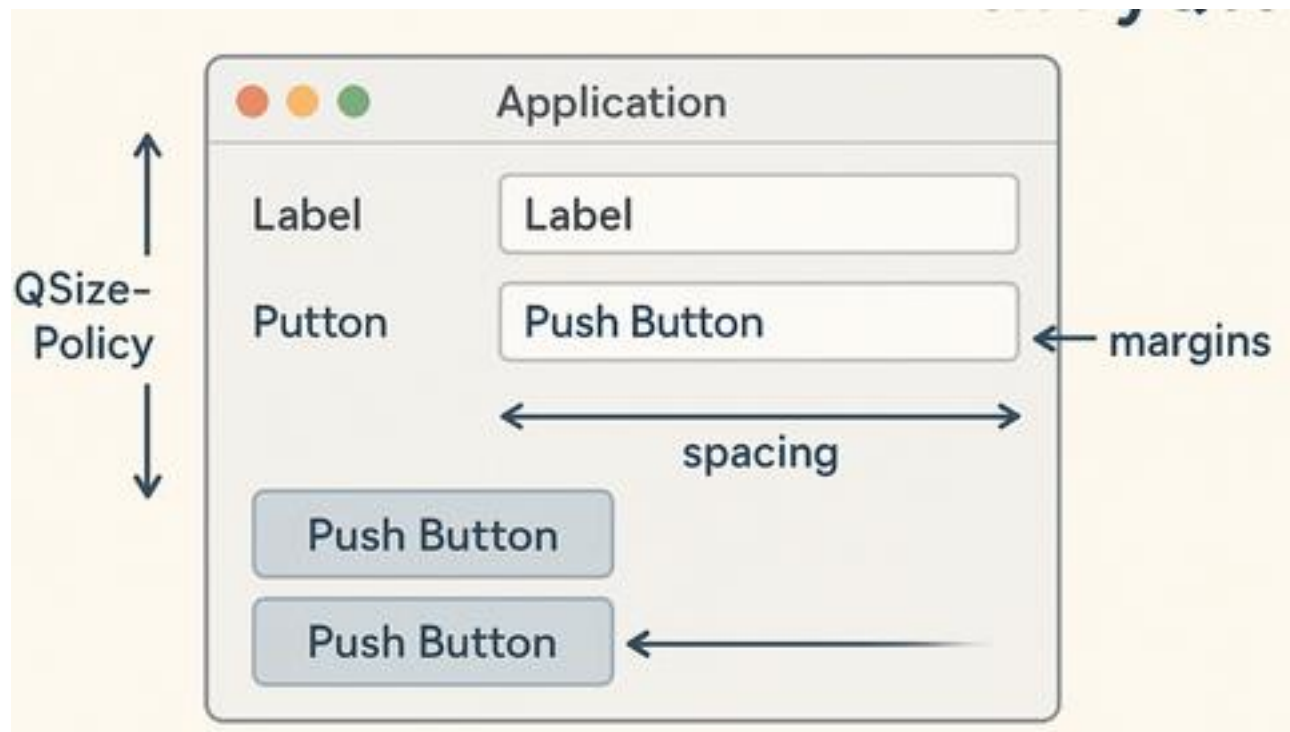
Arbre de décision

Combien de widgets à organiser ?

- ├─ 1-3 widgets simples
 - ├─ Côte à côte → QHBoxLayout
 - └─ Empilés → QVBoxLayout
- ├─ 4-8 widgets
 - ├─ Formulaire simple → QVBoxLayout + QHBoxLayout imbriqués
 - ├─ Grille régulière → GridLayout
 - └─ Paires étiquette-champ → QFormLayout
- └─ 9+ widgets complexes
 - ├─ Zones distinctes → Layouts imbriqués
 - ├─ Interface modulaire → GridLayout principal + sous-layouts
 - └─ Application complète → Architecture en couches

Selon le contexte

Contexte	Layout recommandé	Justification
Barre d'outils	<code>QHBoxLayout</code>	Actions alignées, accès rapide
Formulaire de saisie	<code>QVBoxLayout</code> ou <code>QFormLayout</code>	Flux de lecture naturel (haut→bas)
Panneau de configuration	<code>QGridLayout</code>	Organisation logique en sections
Interface de jeu/calculatrice	<code>QGridLayout</code>	Disposition régulière des boutons
Dashboard	Layouts imbriqués	Zones fonctionnelles distinctes



[Lien TP3](#)

TP3 - Layouts imbriqués complexes

Durée : 30 minutes

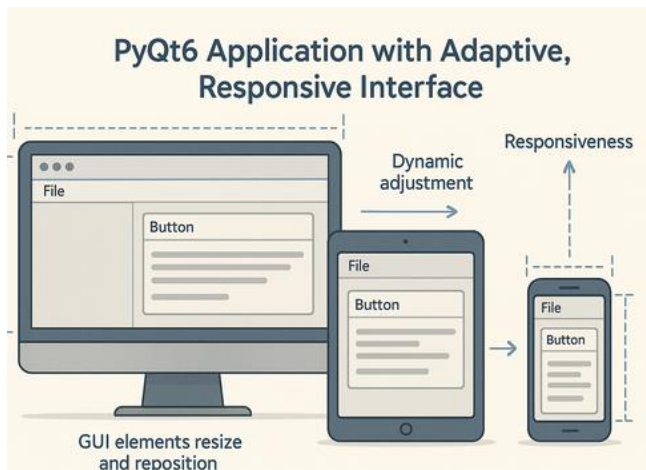
Objectif : Créer une interface style IDE avec panneaux multiples et zones redimensionnables, en utilisant des layouts imbriqués.

Pré-requis : TP1 et TP2 terminés, compréhension des layouts de base.

- 1) Structure du projet IDE**
- 2) Layout principal à trois zones**
- 3) Sidebar de navigation (gauche)**
- 4) Zone centrale divisée**
- 5) Panneau de propriétés (droite)**
- 6) Barre d'outils intégrée**

Balise HTML essentielles (HTML 4)

Balise	Usage	Exemple
<code><h1></code> à <code><h6></code>	Titres	<code><h2>Titre</h2></code>
<code><p></code>	Paragraphe	<code><p>Texte</p></code>
<code></code> , <code></code>	Gras	<code>Important</code>
<code><i></code> , <code></code>	Italique	<code><i>Accentué</i></code>
<code></code> , <code></code> , <code></code>	Listes	<code>Item</code>
<code><table></code> , <code><tr></code> , <code><td></code> , <code><th></code>	Tableaux	<code><table><tr><td>Cellule</td></tr></table></code>
<code></code>	Couleur	<code>Rouge</code>
<code><blockquote></code>	Citation	<code><blockquote>Citation</blockquote></code>



Techniques de mise en œuvre

Technique	Mécanisme	Usage typique
Layouts flexibles	Facteurs d'étirement, politiques de taille	Redimensionnement continu
Seuils de taille	<code>resizeEvent()</code> , breakpoints	Changements de mode d'affichage
Masquage progressif	<code>setVisible()</code> conditionnel	Simplification pour petits écrans
Réorganisation	Layouts multiples, <code>setLayout()</code>	Changement de structure

[Lien TP4](#)

TP4 - Interface responsive

Durée : 30 minutes

Objectif : Développer une interface qui s'adapte automatiquement au redimensionnement et gérer différents modes d'affichage selon la taille.

Pré-requis : TP1 à TP3 terminés, compréhension des layouts avancés.

- 1) Application de blog responsive
- 2) Header adaptatif
- 3) Zone de contenu principale
- 4) Détection du redimensionnement
- 5) Mode tablette (largeur < 800px)

Introduction : ce qu'il faut retenir

✓ Choix du layout approprié

- `QHBoxLayout` : Organisation horizontale, barres d'outils, boutons
- `QVBoxLayout` : Formulaires, listes verticales, organisation séquentielle
- `QGridLayout` : Formulaires complexes, interfaces tabulaires, calculatrices

✓ Politiques de taille et espacement

- Utiliser les politiques de taille pour contrôler le comportement d'étirement
- Gérer l'espacement avec `setSpacing()` et `setContentsMargins()`
- Exploiter `addStretch()` pour créer des espaces flexibles

✓ Layouts imbriqués

- Combiner différents types de layouts pour des interfaces complexes
- Organiser logiquement l'interface en zones fonctionnelles
- Maintenir la cohérence visuelle malgré la complexité

✓ Responsivité et adaptation

- Prévoir le comportement lors du redimensionnement
- Tester sur différentes résolutions d'écran
- Utiliser les signaux de redimensionnement pour l'adaptation dynamique

C'est le moment de versionner :

- <https://github.com/CoursQtTdemares>
- Le commentaire du commit est “chapitre 3 stratégie placement”

Chapitre 4 : Qt Designer

Objectifs de ce chapitre

- Installer et configurer Qt Designer dans un environnement Python moderne
- Maîtriser l'interface et les outils essentiels de Qt Designer
- Créer des interfaces graphiques de manière visuelle et efficace
- Intégrer les fichiers .ui dans vos applications Python (chargement dynamique vs compilation)
- Appliquer les bonnes pratiques de conception d'interface avec Designer

Installation et configuration

```
uv python pin 3.11
```

```
[project]
requires-python = ">=3.11,<3.13"
dependencies = [
    "pyqt6==6.3.1",
    "pyqt6-tools>=6.3.1.3.3",
    "setuptools>=80.9.0",
]
```

```
uv run pyqt6-tools designer
```

Barre de menus et d'outils		
Widget Box <ul style="list-style-type: none">• Layouts• Buttons• Input• Display• Containers	Zone de conception (Canvas)	Property Editor <ul style="list-style-type: none">• Properties• Signal/Slot Editor
Object Inspector		Resource Browser

Chargement et intégration en Python

```
from PyQt6 import QtWidgets, uic

# Chargement direct du fichier .ui
window = uic.loadUi(ui_path)
window.show()
```

Ajout de la logique applicative

```
from ui_mainwindow import Ui_MainWindow

class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self) -> None:
        super().__init__()
        self.setupUi(self)
```

[Lien TP1](#)

TP1 - Première interface avec Designer

Durée : 20 minutes

Objectif : Maîtriser le workflow de base Qt Designer → Python avec chargement dynamique.

Pré-requis : Chapitre 4 maîtrisé, Qt Designer installé via `uv run pyqt6-tools designer`.

- 1) Lancement de Qt Designer
- 2) Création de l'interface simple
- 3) Configuration des propriétés importantes
- 4) Application d'un layout
- 5) Sauvegarde du fichier .ui
- 6) Chargement dynamique en Python
- 7) Connexion du signal au slot
- 8) Test et finalisation

[Lien TP1](#)

TP2 - Compilation avec pyuic6

Durée : 15 minutes

Objectif : Comprendre la compilation .ui → Python et comparer les deux approches d'intégration.

Pré-requis : TP1 terminé avec simple_interface.ui fonctionnel.

- 1) Compilation du fichier .ui
- 2) Examen du code généré
- 3) Création de la classe d'application
- 4) Adaptation des connexions
- 5) Personnalisation post-setupUi
- 6) Test des deux versions
- 7) Analyse des performances
- 8) Choix de l'approche

Introduction : ce qu'il faut retenir

✓ Concepts essentiels

- Qt Designer sépare **conception visuelle** et **logique métier**
- **Chargement dynamique** (`uic.loadUi`) : simple et rapide pour débiter
- **Compilation** (`pyuic6`) : meilleure pour la production

✓ Workflow de base

1. Concevoir l'interface dans Qt Designer
2. Configurer la propriété `objectName` des widgets
3. Charger ou compiler l'interface
4. Connecter les signaux aux slots en Python

✓ Deux méthodes d'intégration

- `uic.loadUi()` : Idéal pour apprendre et prototyper
- `pyuic6` : Recommandé pour les projets professionnels

C'est le moment de versionner :

- <https://github.com/CoursQtTdemares>
- Le commentaire du commit est “chapitre 4 qt designer”

Chapitre 5 : Architecture MVC

Objectifs de ce chapitre

- Comprendre les concepts fondamentaux de l'architecture Model-View de Qt
- Distinguer les responsabilités du modèle et de la vue dans une application Qt
- Implémenter des modèles personnalisés
- Créer des applications avec synchronisation automatique entre données et interface
- Gérer les signaux de modification de modèles pour des mises à jour en temps réel
- Implémenter la persistance de données dans une architecture Model-View
- Construire des interfaces complexes avec QListView et QTableView

Modèle

- Gérer les données
- Contient la structure et les règles de validation
- Ne connaît pas l'interface utilisateur

Et le controller ?

Dans Qt c'est notre classe
QMainWindows, il est intégré
à la vue.

Vue

- Présenter les données
- Contient les widgets d'affichage et d'interaction
- Plusieurs vues peuvent partager le même modèle

Todo List MVC exemple

Ma Todo List

Ma Todo List

+ Ajouter

teststatus: True
coucoustatus: False

Terminer

Supprimer

Lien TP1

TP1 -

Introduction : ce qu'il faut retenir

C'est le moment de versionner :

- <https://github.com/CoursQtTdemares>
- Le commentaire du commit est “chapitre 1 Introduction”

Chapitre 1 : Introduction

Objectifs de ce chapitre

- Comprendre l'évolution historique des interfaces graphiques
- Situer Qt dans le contexte historique des frameworks GUI
- Comprendre ce qu'est Qt et le rôle du binding PyQt6
- Distinguer PyQt6 de ses alternatives (PyQt5, PySide6)
- Installer et configurer un environnement de développement PyQt6
- Utiliser VSCode pour développer des applications Qt
- Naviguer efficacement dans la documentation Qt/PyQt6
- Créer votre première application Qt fonctionnelle

1.1 Contexte historique

[Lien TP1](#)

TP1 - Installation et validation de l'environnement

Objectif : Disposer d'un environnement Python fonctionnel avec uv, exécuter un mini script PyQt6 et valider que VSCode détecte l'interpréteur.

- 1) Créer l'espace de travail du TP
- 2) Installer uv (gestionnaire Python rapide)
- 3) Initialiser un projet minimal
- 4) Ajouter PyQt6 comme dépendance
- 5) Créer un premier script de validation

Introduction : ce qu'il faut retenir

C'est le moment de versionner :

- <https://github.com/CoursQtTdemares>
- Le commentaire du commit est “chapitre 1 Introduction”



BOOMERANG
CONSULTING

TP de validation des acquis Variables, Tableaux et Fonctions



boomerang-consulting.com

TP de validation des acquis

- Enoncé du TP de validation des acquis portant sur les 3 ou 4 dernières notions abordées.

Le dernier jour de la formation au retour de la pause déjeuner avant les évaluations.

- Je partage plusieurs livres numériques sur des technos et langages informatiques (lien ici vers mon drive)
- Je met sur ce slide une liste de ressources qui pourraient aider les stagiaires à monter en compétence après cette formation
- je donne aux stagiaires un ensemble d'exemples de code sur angular que j'ai mis sur un stack blitz
- Je donne aux stagiaires une version du crm avec d'avantage de tests unitaires que ce que l'on a pu écrire en formation
- Je donne aux stagiaires une liste avec les liens vers des IA que j'ai testé et qui peuvent présenter un réel intérêt pour eux.

- Revenons sur les questions hors plan de cours que vous m'avez posé durant la formation pour y répondre

- Lien vers la formation suivante que vous recommandez aux stagiaires (par ex angular avancé si vous avez donné la formation angular initiation)
- La formation angular avancée, pourquoi ?

**Merci d'avoir suivi cette formation
Boomerang Consulting et à très bientôt !**

- Merci d'avoir participé à cette formation Boomerang Consulting.
- Envoie du Bilan formation.

Timothée, Demares

Formateur externe

timothee.demares@gmail.com

<https://www.linkedin.com/in/tdemares/>

Et encore merci !