Voici une proposition d’introduction structurée, prête à être utilisée en séance, avec 50% de pratique minimum. Elle couvre: Présentation du binding PyQt, Installation de Qt/PyQt, Utilisation de VS Code, et Documentation en ligne.

Plan pédagogique (1h45 à 2h)

* 20 min Présentation du binding PyQt
* 25 min Installation Qt/PyQt (avec validation)
* 40 min VS Code: configuration, exécution, debug
* 20 min Documentation en ligne et exercices de recherche
* Ratio pratique: ≥50% (4 PT)

1. Présentation du binding PyQt Objectifs

* Comprendre ce qu’est Qt et ce que PyQt apporte à Python
* Situer PyQt par rapport à PySide, PyQt5/6, modules QtWidgets/QtCore/QtGui

## Présentation du bonding PyQt

### Objectifs

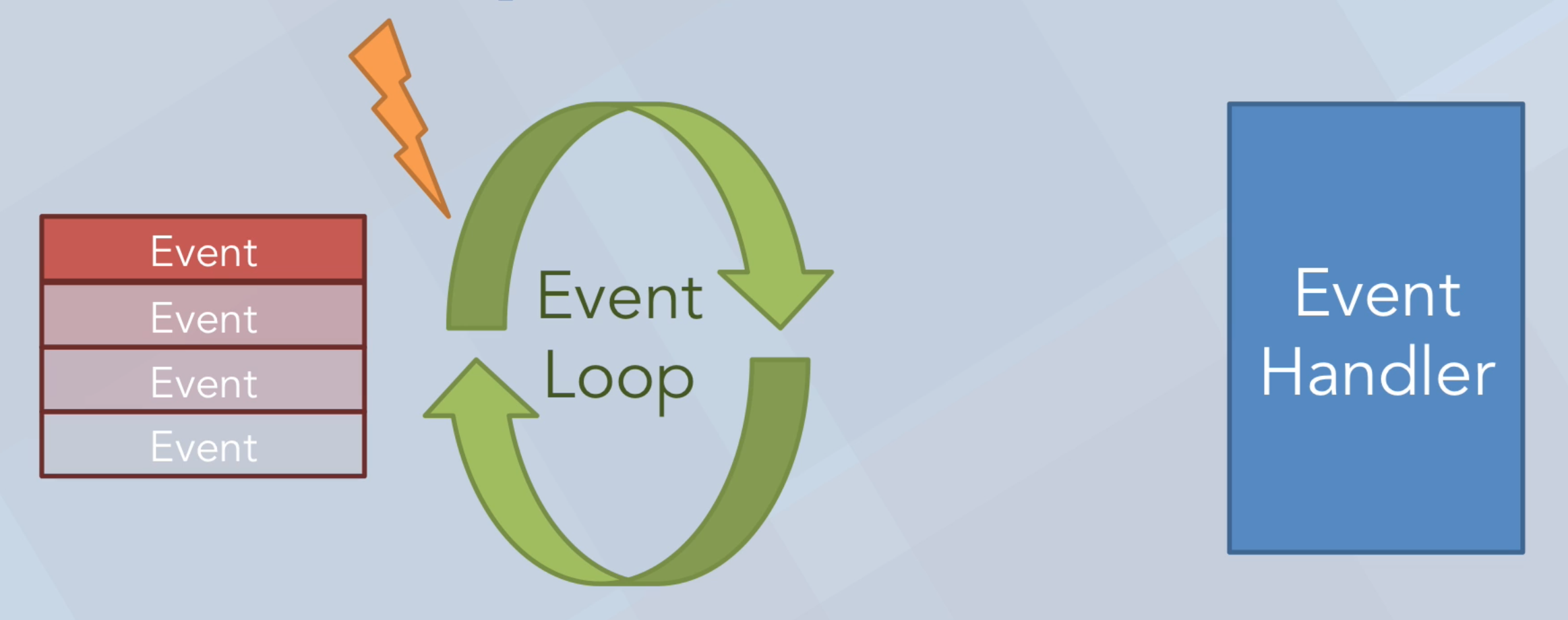
* Comprendre ce qu’est Qt et ce que signifie “binding” pour Python.
* Situer PyQt dans l’écosystème (PyQt5/6 vs PySide6), les modules clés et le modèle événementiel.
* Configuration de l’environnement python pour pyQt
* Premiers scripts « Hello World »
* Documentation en ligne

### Contexte et Définition

* Qt: framework C++ multiplateforme pour interfaces graphiques et applications desktop (Windows, macOS, Linux). Points forts: stabilité, riche bibliothèque de widgets, internationalisation, impression, réseau, SQL, multimédia.
* Binding: couche qui expose l’API C++ de Qt aux langages de haut niveau. Pour PyQt, le binding est généré par SIP (outil de Riverbank) et mappe classes, méthodes, énumérations, signaux/slots vers Python.

### Architecture Qt côté Python

* Application Qt: un seul objet QApplication par processus. Il crée la boucle d’événements (event loop) qui distribue les événements aux widgets. L’application se lance avec app.exec() et se termine quand la dernière fenêtre est fermée ou sur quit().
* Modules principaux utilisés en PyQt6:
  + QtCore: base non graphique (boucle d’événements, QTimer, QDateTime, fichiers, threads, signaux/slots).
  + QtGui: éléments graphiques bas niveau (QIcon, QPixmap, QPainter, polices, raccourcis).
  + QtWidgets: widgets classiques (QMainWindow, QPushButton, QLabel, QLineEdit, layouts).
  + Autres notables: QtNetwork, QtSql, QtMultimedia, QtSvg. Nous nous concentrons d’abord sur QtWidgets.
* Modèle événementiel: tout passe par la boucle d’événements. Une opération longue dans le thread GUI gèlera l’UI (on abordera QThread/QTimer plus tard). Les signaux/slots sont le mécanisme de notification asynchrone de Qt.



### PyQt6 vs PyQt5 vs PySide6

* PyQt6 (recommandé ici): mappe Qt 6.x. Enums et flags “scopés” (namespacés). Certaines méthodes renommées. app.exec() est la norme.
* PyQt5: mappe Qt 5.x. Enums non scopés, API légèrement différente. Présent dans de nombreux projets legacy. Parfois app.exec\_() dans vieux exemples.
* PySide6: binding officiel par Qt (généré via Shiboken). API très proche de PyQt6. Licences plus permissives pour certains cas (LGPL). Si vous avez des exigences de licence pour distribution propriétaire sans achat de licence commerciale PyQt, PySide6 peut être mieux adapté.
* Licences (résumé haut niveau, pas du conseil légal): PyQt est GPL/commercial; PySide est LGPL. Le choix dépend de votre stratégie de distribution. Dans ce cours: PyQt6.
* Compatibilité: la doc Qt C++ est la référence commune. La transposition vers Python est quasi mécanique.

### Installation de uv

<https://docs.astral.sh/uv/getting-started/installation/>

### Tp : Création du premier projet

1. Créer un dossier vide
2. Ouvrir un terminal
3. Commandes initiales :
   * uv init
   * uv venv
   * uv add pyqt6
   * uv venv
4. Lire la documentation, créer un script basique à l’aide de la documentation https://www.pythonguis.com/tutorials/pyqt6-creating-your-first-window/

Contenu

* Qt: toolkit GUI multiplateforme (Windows, macOS, Linux). Architecture: boucle d’événements, signaux/slots, widgets, QML/QtQuick, modules (QtCore, QtGui, QtWidgets, QtNetwork, etc.).
* PyQt: binding Python maintenu par Riverbank Computing.
  + Versions: PyQt6 (Qt 6.x), PyQt5 (Qt 5.x). On utilisera PyQt6 par défaut.
  + Alternative: PySide6 (binding officiel Qt). Points de vigilance: licence, API très proche, types annotés plus complets côté PySide, mais on reste sur PyQt.
* Concepts clés:
  + Application Qt: un seul QApplication par process, boucle d’événements via app.exec().
  + Signaux/Slots: mécanisme d’événements (ex.: bouton.clicked.connect(handler)).
  + Enums et API PyQt6: enums « scopés » (Qt.AlignmentFlag.AlignCenter), renommages par rapport à PyQt5.
  + Deux familles UI: QtWidgets (notre focus) et QtQuick/QML (hors périmètre intro).

PT1 (10–15 min)

* QCM/échange éclair: choisir entre PyQt6 et PyQt5 selon besoins (OS legacy, dépendances spécifiques), identifier 3 modules Qt utiles à un CRUD desktop, expliquer « signal/slot » en 1 phrase.

2. Installation des librairies Qt et PyQt Objectifs

* Installer un environnement Python isolé
* Installer PyQt6 et valider avec un “Hello Qt”

Pré-requis techniques

* Python 3.10 à 3.12 recommandé
* Outils: pip, venv (ou conda si déjà standard en entreprise)

Étapes (toutes plateformes)

* Créer le projet et un venv:
  + Windows
    - py -3 -m venv .venv
    - .venv\Scripts\activate
  + macOS/Linux
    - python3 -m venv .venv
    - source .venv/bin/activate
* Mettre à jour l’outillage et installer PyQt6:
  + python -m pip install --upgrade pip wheel
  + pip install pyqt6
* Vérifier l’installation:
  + Créer hello\_qt.py
    - Contenu: from PyQt6.QtWidgets import QApplication, QLabel import sys

app = QApplication(sys.argv) label = QLabel("Bonjour Qt") label.resize(300, 50) label.show() sys.exit(app.exec())

* + Exécuter: python hello\_qt.py (une fenêtre « Bonjour Qt » doit apparaître)

Notes et dépannage

* Linux: installer au besoin les libs xcb si erreur “could not load the Qt platform plugin xcb”.
* macOS: privilégier Python officiel (python.org) pour éviter conflits de frameworks.
* Optionnel: Qt Designer
  + Avec PyQt6 via paquets tiers (pyqt6-tools) ou l’installateur Qt officiel. Pour l’intro on code les UI à la main.

PT2 (10–15 min)

* Objectif: valider l’environnement
* Livrable: exécution de hello\_qt.py, capture écran + pip freeze > requirements.txt

3. Utilisation de l’IDE VS Code Objectifs

* Configurer VS Code pour un projet PyQt
* Exécuter et déboguer une app Qt
* Mettre en place le lint/format

Extensions recommandées

* Python (Microsoft) + Pylance
* Black ou Ruff (format/lint)
* Optionnel: Qt-related (préview .ui/QML si vous en utilisez)

Configuration de base

* Ouvrir le dossier du projet dans VS Code
* Sélectionner l’interpréteur du venv: Ctrl+Shift+P > Python: Select Interpreter > .venv
* Paramètres utiles (Settings / ou settings.json):
  + "python.terminal.activateEnvironment": true
  + "python.analysis.typeCheckingMode": "basic" (ou "strict" si vous aimez)
  + Si Black: "python.formatting.provider": "black"

Debug: launch.json (exemple)

* .vscode/launch.json: { "version": "0.2.0", "configurations": [ { "name": "Python: Qt app", "type": "python", "request": "launch", "program": "${workspaceFolder}/src/app.py", "console": "integratedTerminal", "justMyCode": true, "env": { "QT\_API": "PyQt6" } } ] }

Exemple d’app pour debugger (src/app.py)

* Contenu minimal: from PyQt6.QtWidgets import QApplication, QMainWindow, QLabel from PyQt6.QtGui import QAction import sys

class MainWindow(QMainWindow): def **init**(self): super().**init**() self.setWindowTitle("Hello Qt") self.setCentralWidget(QLabel("Bienvenue")) exit\_action = QAction("Quitter", self) exit\_action.triggered.connect(self.close) menu = self.menuBar().addMenu("Fichier") menu.addAction(exit\_action) self.statusBar().showMessage("Prêt")

def main(): app = QApplication(sys.argv) w = MainWindow() w.show() sys.exit(app.exec())

if **name** == "**main**": main()

Conseils

* Placez des breakpoints dans **init** et sur la méthode connectée au signal (ex.: triggered).
* Utilisez la palette de commandes pour « Debug: Start Debugging ».

Qualité de code (optionnel rapide)

* pyproject.toml minimal pour Black/Ruff: [tool.black] line-length = 100 target-version = ["py310"]

[tool.ruff] line-length = 100 select = ["E","F","I"] ignore = []

PT3 (20–25 min)

* Créer la structure src/, ajouter app.py
* Configurer launch.json et exécuter en debug
* Ajouter un QAction “À propos…” qui affiche un message (QMessageBox.information) et placer un breakpoint dans le slot

4. Utilisation des documentations en ligne Objectifs

* Savoir trouver rapidement la bonne info (PyQt vs Qt C++)
* Lire une signature C++ et la transposer en Python
* Connaître les ressources de référence

Ressources clés

* PyQt6 (Riverbank): <https://www.riverbankcomputing.com/static/Docs/PyQt6/>
  + Index des modules/classes, différences vs PyQt5, notes sur enums/flags
* Qt 6 (C++): <https://doc.qt.io/qt-6/>
  + Référence exhaustive (Widgets, signaux/slots, exemples)
  + Astuce: la doc C++ s’applique à 95%. Mapper: QString → str, QList<T> → list[T], enums scopés via Qt.XxxFlag
* Tutoriels et exemples:
  + Exemples Qt Widgets: <https://doc.qt.io/qt-6/examples-widgets.html>
  + Site “Python GUIs” (PyQt): <https://www.pythonguis.com/>
* Aide in-code:
  + help(QMainWindow), dir(objet), print(type(objet))
  + Pylance “Go to Definition” pour naviguer dans les wrappers

Méthode de recherche

* Requête ciblée: site:doc.qt.io QMainWindow QAction signals
* Identifier la classe, ses signaux (ex.: QLineEdit.textChanged), ses propriétés, ses enums (Qt.WindowType, Qt.AlignmentFlag)
* Transposition:
  + C++: connect(signal, slot) → Python: signal.connect(slot)
  + C++: void clicked(bool checked = false) → Python: clicked(bool)

PT4 (15–20 min)

* Trouver dans la doc:
  + Les signaux de QLineEdit qui notifient une modification (réponse attendue: textChanged, textEdited)
  + L’enum pour centrer un texte (Qt.AlignmentFlag.AlignCenter)
  + La doc de QStatusBar et comment afficher un message temporaire (showMessage("…", 2000))
* Implémenter:
  + Remplacer le QLabel central par un QLineEdit
  + Connecter textChanged pour mettre le texte courant dans la status bar
  + Centrer le QLabel si vous le gardez

Livrables attendus (fin d’intro)

* Environnement fonctionnel (venv + PyQt6)
* VS Code configuré (interpréteur, debug)
* Mini-app QMainWindow avec menu Fichier > Quitter, action À propos, status bar
* Une courte note listant 3 liens de doc consultés et ce que vous y avez trouvé

Variantes et options

* Si besoin de compatibilité legacy: pip install pyqt5 et adapter les enums (non scopés), app.exec\_() au lieu de app.exec() dans certains exemples anciens.
* Environnements alternatifs: conda create -n qt python=3.11; conda install pyqt
* Qt Designer: possible via “pyqt6-tools” pour prototypage UI visuel (recommandé après avoir maîtrisé le code manuel)

Résumé des points d’attention

* Un seul QApplication par process
* Toujours terminer par sys.exit(app.exec())
* PyQt6: enums/flags scopés, noms parfois différents de PyQt5
* VS Code: sélectionner le bon interpréteur et lancer via le terminal intégré pour bénéficier de l’activation du venv
* La doc Qt C++ reste la référence, la doc PyQt complète la cartographie côté Python

Souhaitez-vous que je transforme ce plan en supports “pas-à-pas” pour une séance de 2h avec slides + fiches PT prêtes à remettre aux stagiaires ?