03/09/2023

Python pour l'ingénieur

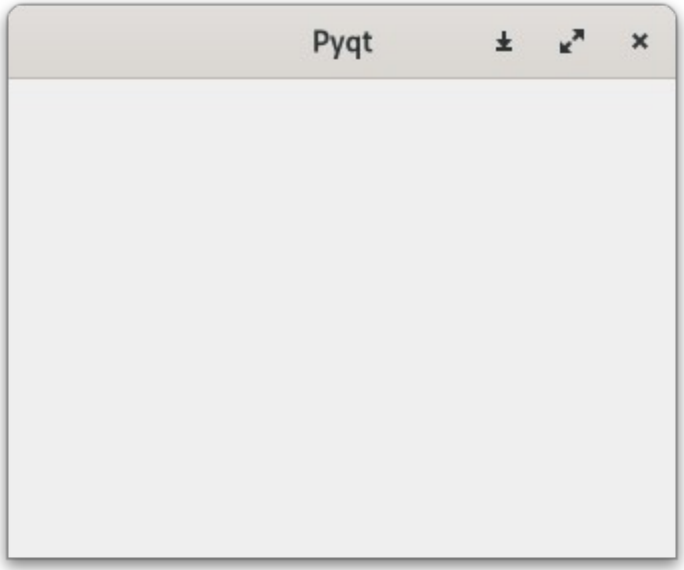
IHM avec PyQt

Module de formation

© Copyright 2023 Cyril Keime & Arnaud Bêche

Créer une fenêtre simple

```python
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

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

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

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

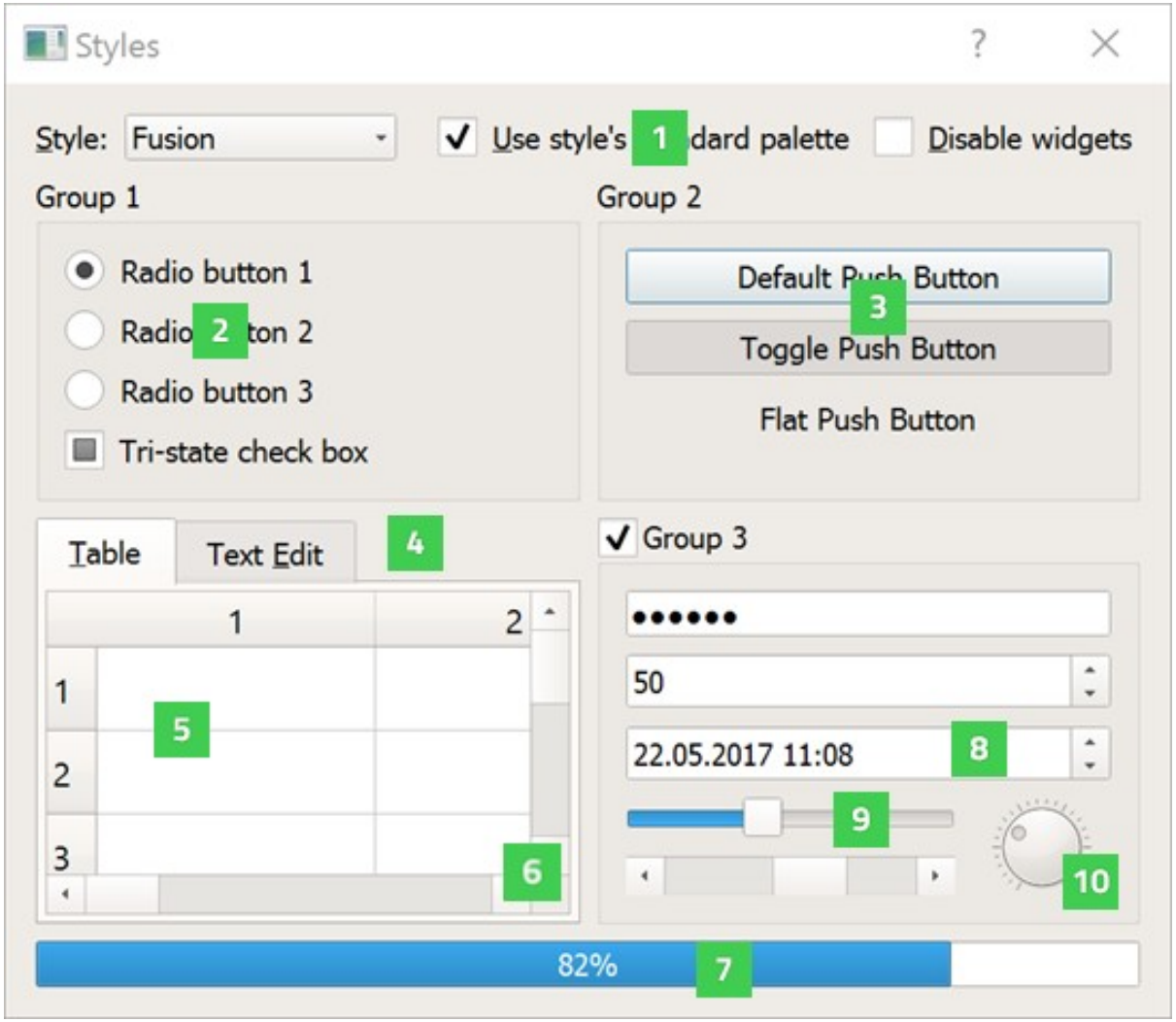


Points clés :

- Boucle d'événements
- Dérivation de QWidget

© Copyright 2023 Cyril Keime & Arnaud Bêche

Widgets classiques (1/2)



(1) QCheckBox

(2) QRadioButton

(3) QPushButton

(4) QTabWidget

(5) QTableWidget

(6) QScrollBar

(7) QProgressBar

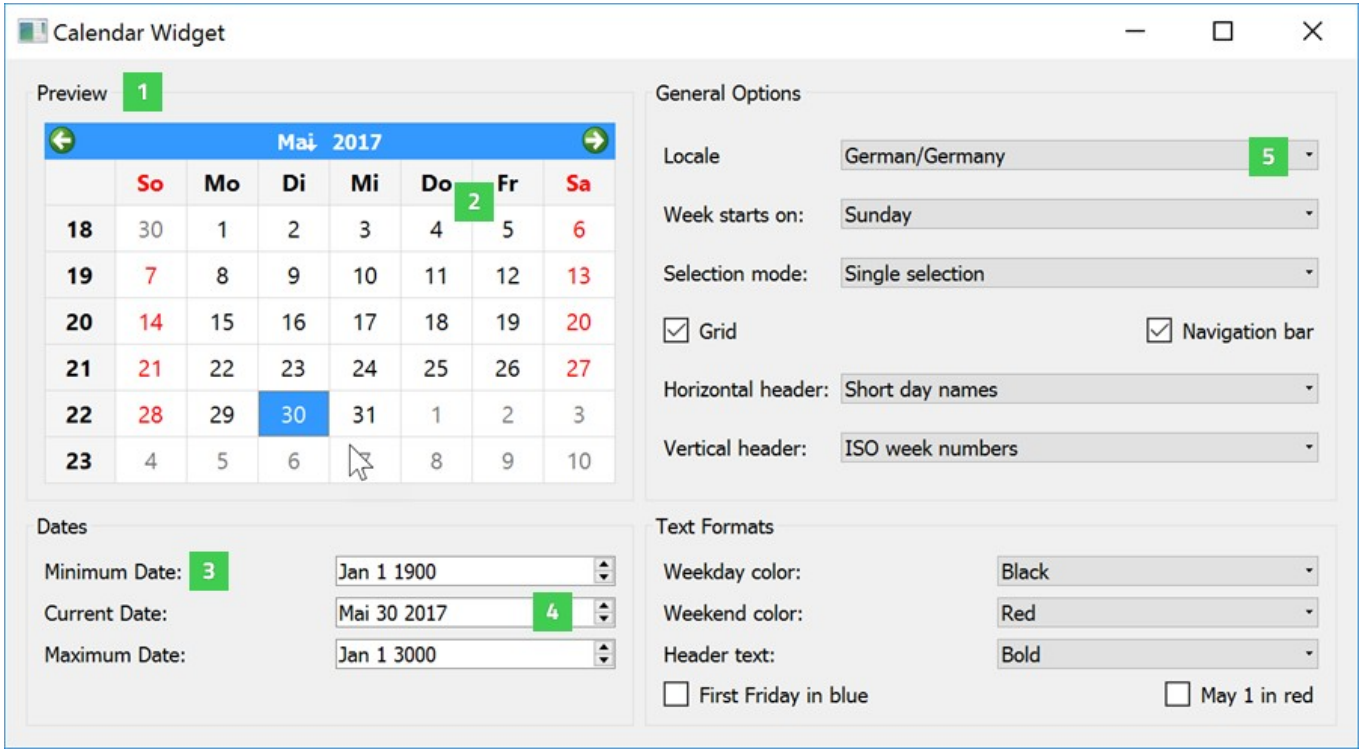
(8) QDateTimeEdit

(9) QSlider

(10) QDial

© Copyright 2023 Cyril Keime & Arnaud Bêche

Widgets classiques (2/2)



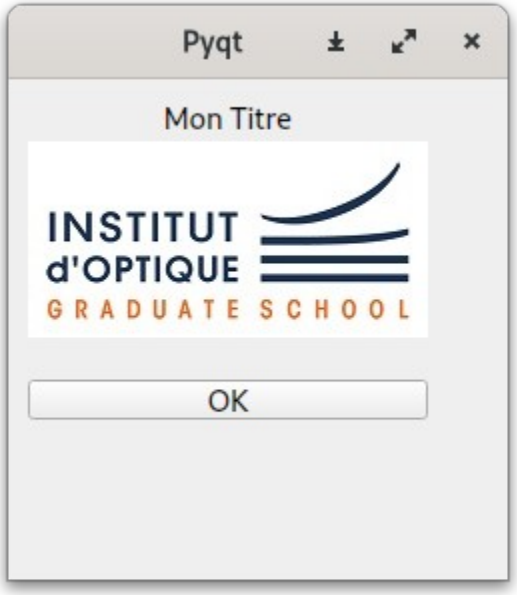
(1) QGroupBox
(2) QCalendarWidget
(3) QLabel
(4) QDateEdit
(5) QComboBox

Points clés :

- Consulter la doc de Qt pour connaître les constructeurs, les méthodes, les signaux et slots associés à ces widgets (voir dernière slide)

Créer et placer des widgets

```
class MaFenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')
        self.resize(250, 250)

        label = QLabel('Mon Titre', self)
        label.setAlignment(Qt.AlignHCenter)
        label.setGeometry(10, 10, 200, 20)

        image = QLabel(self)
        image.setPixmap(QPixmap('logo.jpg'))
        image.setGeometry(10, 30, 200, 100)

        bouton = QPushButton('OK', self)
        bouton.setGeometry(10, 150, 200, 20)
```



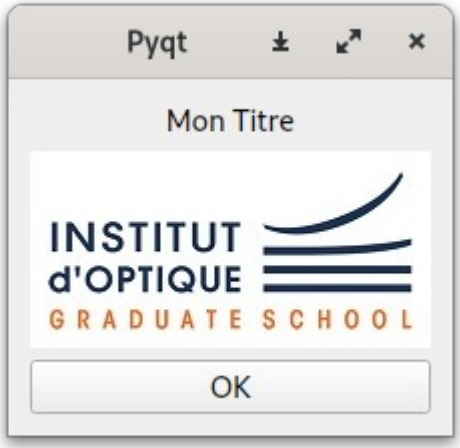
Points clés :

- Notion de widget
- Bibliothèque de widgets

© Copyright 2023 Cyril Keime & Arnaud Bêche

Placement avec Layout

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



Types de Layouts :

- QHBoxLayout
- QVBoxLayout
- QGridLayout
- QFormLayout

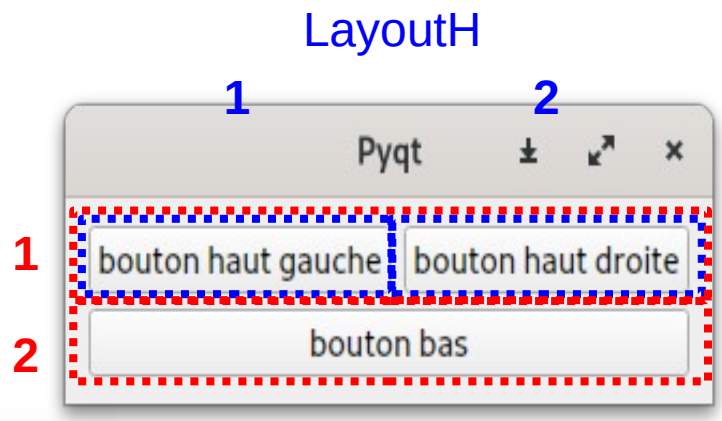
Points clés :

- Privilégier les layouts pour organiser une interface

© Copyright 2023 Cyril Keime & Arnaud Bêche

Layouts imbriqués

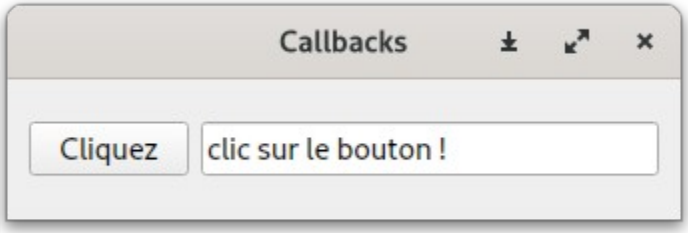
```
class MaFenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')
        bouton1 = QPushButton('bouton haut gauche')
        bouton2 = QPushButton('bouton haut droite')
        bouton3 = QPushButton('bouton bas')
        layoutH = QHBoxLayout()
   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



© Copyright 2023 Cyril Keime & Arnaud Bêche

Slot et Signal

```
class MaFenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Callbacks")
        self.bouton = QPushButton('Cliquez', self)
        self.texte = QLineEdit(self)
        self.bouton.clicked.connect(self.on_bouton)
                    signal              slot ou callback
        layout = QHBoxLayout()
        layout.addWidget(self.bouton)
        layout.addWidget(self.texte)
        self.setLayout(layout)

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



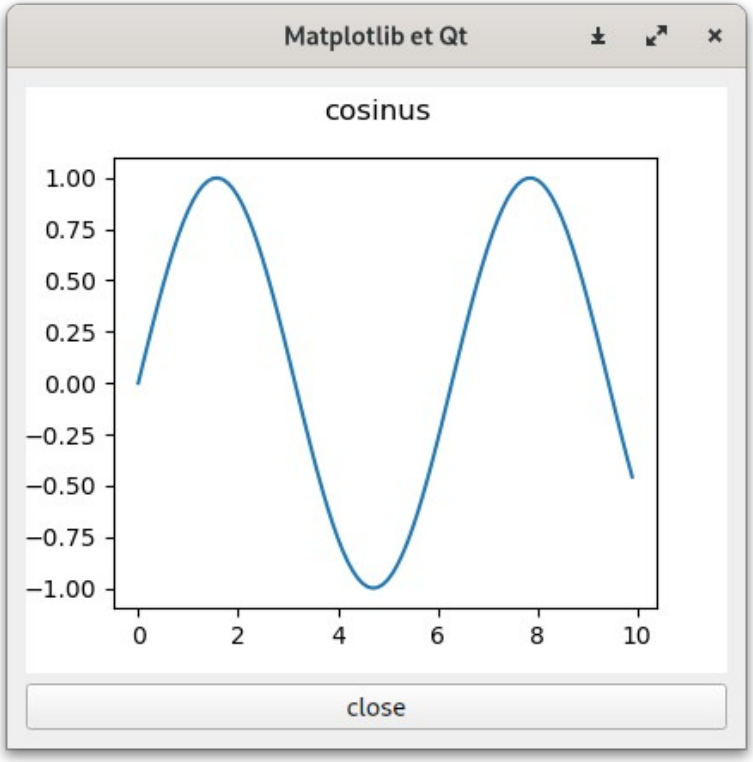
Points clés :

- Pas de parenthèses pour définir la callback : on n'appelle pas la fonction, on fournit son nom à connect
- Créer les widgets comme attributs de la classe pour pouvoir les manipuler dans les méthodes

© Copyright 2023 Cyril Keime & Arnaud Bêche

Intégrer un graph Matplotlib

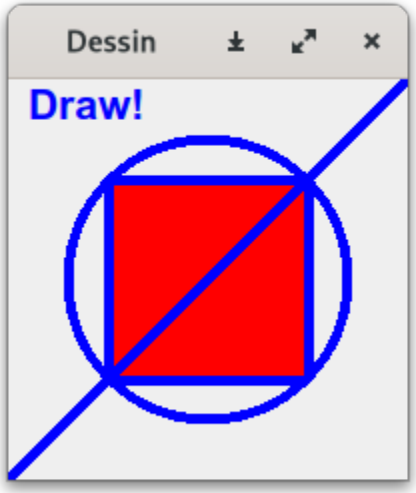
```
from matplotlib.backends.backend_qt5agg import
FigureCanvasQTAgg as FigureWidget

class MaFenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Matplotlib et Qt")
        button = QPushButton("close")
        button.clicked.connect(self.close)
        fig = self.get_figure()
        fig_widget = FigureWidget(fig)
        layout = QVBoxLayout()
        layout.addWidget(fig_widget)
        layout.addWidget(button)
        self.setLayout(layout)

    def get_figure(self):
        x = np.arange(0, 10, 0.1)
        fig = plt.figure()
        fig.suptitle("cosinus")
        ax = fig.add_subplot()
        ax.plot(x, np.sin(x))
        return fig
```



© Copyright 2023 Cyril Keime & Arnaud Bêche

Dessiner

```
class MaFenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Dessin")
        self.resize(200, 200)

    def paintEvent(self, event):
        painter = QPainter(self)
        font = QFont("Arial", 16, QFont.Bold)
        painter.setFont(font)
        pen = QPen(QColor(0, 0, 255), 5)
        painter.setPen(pen)
        painter.drawText(10, 20, "Draw!")
        painter.drawEllipse(30, 30, 140, 140)
        painter.setBrush(QColor(255, 0, 0))
        painter.drawRect(50, 50, 100, 100)
        painter.drawLine(0, 200, 200, 0)
        painter.end()
```



Points clés :

- Pas d'appel direct de paintEvent()

© Copyright 2023 Cyril Keime & Arnaud Bêche

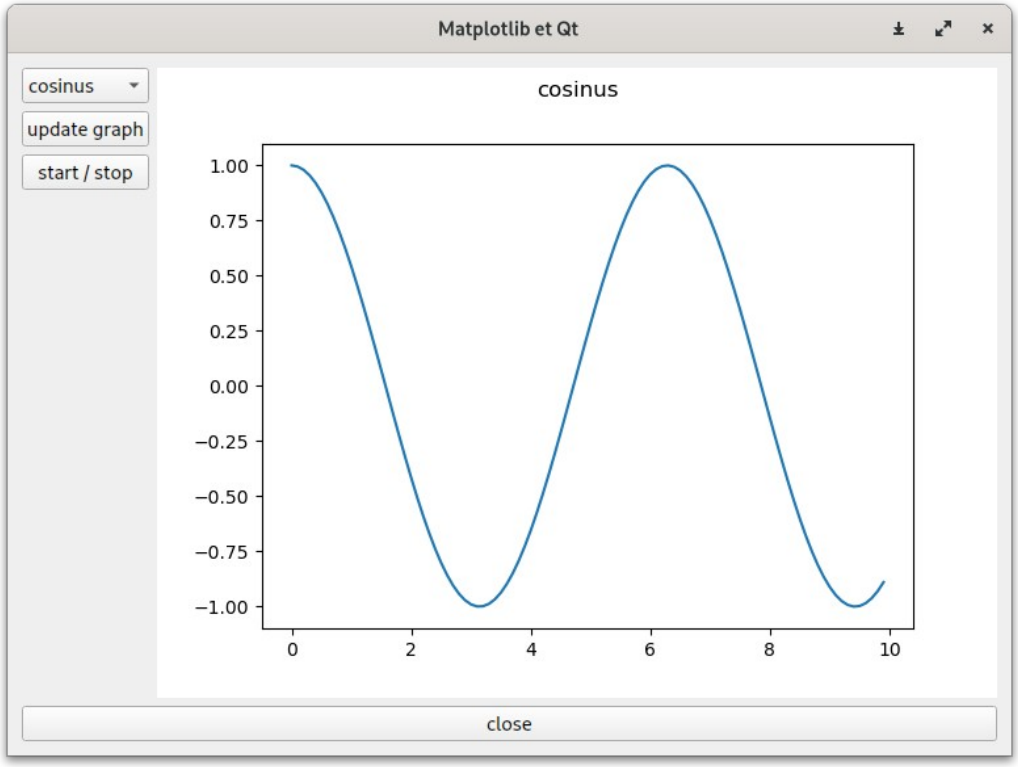
Animer avec un Timer

```
class MaFenetre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Dessin")
        self.resize(200, 200)
        self.x = 0

        self.timer = QTimer()
        self.timer.timeout.connect(self.refresh)
        self.timer.start(100)

    def paintEvent(self, event):
        painter = QPainter(self)
        font = QFont("Arial", 16, QFont.Bold)
        painter.setFont(font)
        pen = QPen(QColor(0, 0, 255), 5)
        painter.setPen(pen)
        painter.drawText(self.x, 100, "Draw!")
        painter.end()

    def refresh(self):
        self.x += 1
        self.update()
```

Points clés :

- Pour faire une animation, utiliser un Timer et pas une boucle
- Demander une mise à jour du dessin avec update()

Exercice



- <u>1re partie</u>
  - Repartir du code de la page 9
  - Créer des layouts pour organiser la fenêtre comme illustré ci-contre.
  - Ajouter un combobox pour choisir la courbe (cosinus ou sinus)
  - Ajouter un bouton pour mettre à jour le graph après un changement du choix de courbe
  - Coder la callback associée au bouton
- <u>2e partie</u>
  - Créer un timer. A chaque top du timer, la phase de la courbe est incrémentée et la courbe est mise à jour.
  - Ajouter un bouton start/stop. A chaque clic surle bouton, le timer est démarré ou arrêté.

© Copyright 2023 Cyril Keime & Arnaud Bêche

Ressources en ligne

- Documentation officielle
  - Qt5 (C++) : https://doc.qt.io/qt-5/classes.html
  - PyQt5 : https://www.riverbankcomputing.com/static/Docs/PyQt5/

- Aide en ligne
  - https://stackoverflow.com/
  - https://chat.openai.com/