

Python

Tutoriels relatifs à la programmation Python

- [Faire une interface graphique avec PySide 6](#)
- [Travailler avec des fichiers](#)

Faire une interface graphique avec PySide 6

Sur cette page nous allons voir l'utilisation des widgets les plus utiles pour faire une interface graphique simple en Python. Pour plus de détails, se référer à la [documentation de PySide 6](#) qui est très complète.

Pour installer PySide il suffit d'utiliser cette commande :

```
pip install PySide6
```

1. Hello World

Commençons d'abord par un classique "Hello World" version interface graphique.

```
# Importation des bibliothèques
import sys
from PySide6.QtWidgets import *

class MainWindow(QMainWindow):

    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)

        # Définition du titre de la fenêtre
        self.setWindowTitle("Hello!")

        # Ajout des widgets
        self.edit = QLineEdit()
        self.edit.setPlaceholderText("Quel est votre nom ?")
        self.button = QPushButton("Dis-moi bonjour")

        # Création d'une disposition verticale QVBoxLayout
        layout = QVBoxLayout()
```

```

# On ajoute les widgets créé à la disposition
# Le champ edit sera donc au-dessus du bouton
layout.addWidget(self.edit)
layout.addWidget(self.button)

# Création d'un widget principal qui va tout contenir
central_widget = QWidget()
central_widget.setLayout(layout)

self.setCentralWidget(central_widget)

# On lie le bouton à l'exécution d'une fonction "bonjour" que l'on va définir plus bas
self.button.clicked.connect(self.bonjour)

def bonjour(self):
    # Affichage de la ligne dans la console
    print(f"Bonjour {self.edit.text()}")

    # Ouverture d'une boîte de dialogue
    QMessageBox.information(self, "Information", f"Bonjour {self.edit.text()} !")

if __name__ == '__main__':
    # Create the Qt Application
    app = QApplication(sys.argv)

    # Create and show the form
    window = MainWindow()
    window.show()

    # Run the main Qt loop
    sys.exit(app.exec())

```

La classe `MainWindow` définit donc la fenêtre principale de notre application et c'est dans la fonction `__init__` que nous allons définir tous les éléments de l'interface, ainsi que leurs dispositions les uns par rapport aux autres.

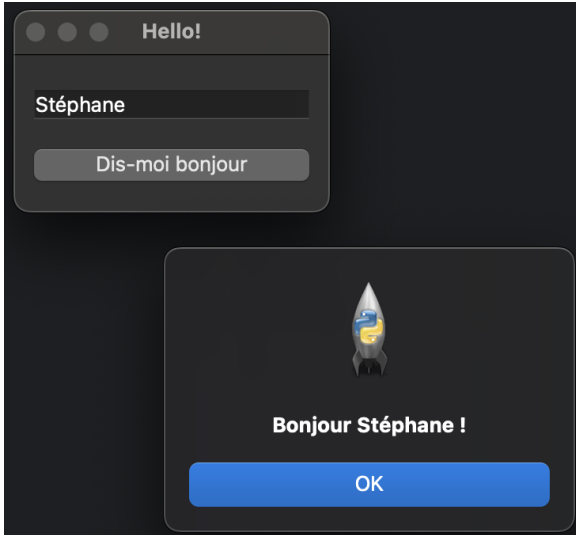
L'instruction `super(MainWindow, self).__init__(parent)` permet d'hériter toutes les caractéristiques et fonctions de la classe parent de `QMainWindow`.

Le widget [QLineEdit](#) permet de créer un champ éditables par l'utilisateur, pour récupérer une valeur. Le widget [QPushButton](#) permet de créer un bouton standard. Le widget [QMessageBox.information](#)

permet d'ouvrir une boîte de dialogue à caractère informatif qui se valide en cliquant tout simplement sur un bouton OK.

Chaque widget possède ses propres caractéristiques et fonctions qui sont décrits dans la documentation (cf lien de chaque widget). C'est ainsi que, par exemple, on peut récupérer la valeur du champ QLineEdit grâce à sa fonction `text()` (lignes 38 et 41).

Lorsque l'on exécute le code on obtient l'application suivante :



2. Hello World v2

Voyons maintenant une interface un peu plus complexe avec de nouveaux widgets et une nouvelle façon de disposer les éléments :

```
def __init__(self, parent=None):
    super(MainWindow, self).__init__(parent)

    # Définition du titre de la fenêtre
    self.setWindowTitle("Hello!")
```

```
# Ajout des widgets
label_nom = QLabel("Nom")
self.nom = QLineEdit()
self.nom.setPlaceholderText("Nom")
label_prenom = QLabel("Prénom")
self.prenom = QLineEdit()
self.prenom.setPlaceholderText("Prénom")
label_age = QLabel("Âge")
self.age = QSpinBox()
self.button = QPushButton("Dis-moi bonjour")

# Création d'un groupe de widgets
groupe = QGroupBox("Identification")

# Création d'une disposition en grille
grille = QGridLayout()

grille.addWidget(label_nom, 1, 1)
grille.addWidget(self.nom, 1, 2)
grille.addWidget(label_prenom, 2, 1)
grille.addWidget(self.prenom, 2, 2)
grille.addWidget(label_age, 3, 1)
grille.addWidget(self.age, 3, 2)

# On définit la grille comme disposition du groupe
groupe.setLayout(grille)

# Création d'une disposition verticale QVBoxLayout
layout = QVBoxLayout()

# On ajoute les widgets créé à la disposition
# Le champ edit sera donc au-dessus du bouton
layout.addWidget(groupe)
layout.addWidget(self.button)

# Création d'un widget principal qui va tout contenir
central_widget = QWidget()
central_widget.setLayout(layout)
```

```
self.setCentralWidget(central_widget)
```

```
# On lie le bouton à l'exécution d'une fonction "bonjour" que l'on va définir plus bas
```

```
self.button.clicked.connect(self.bonjour)
```



C'est en imbriquant plusieurs dispositions que l'on obtient des interfaces plus travaillées et plus complexes.

Ici on a également introduit 2 nouveaux widgets : [QLabel](#) pour afficher du texte non-éditable et [QSpinBox](#) pour un champ nombre incrémentable.

Quelques autres widgets intéressants à connaître :

- Menu déroulant : [QComboBox](#)
- Case à cocher : [QCheckBox](#)
- Slider : [QSlider](#)
- Liste d'éléments : [QListWidget](#)

Travailler avec des fichiers

Ouvrir, créer ou modifier un fichier avec Python et la bibliothèque PySide est très facile.

Avec le widget [QFileDialog](#), PySide permet d'ouvrir une boîte de dialogue système pour sélectionner un ou plusieurs fichiers. La fonction `getSaveFileName()` de cet objet permet par exemple de récupérer le nom du fichier à enregistrer, tandis que `getOpenFileNames()` permet de récupérer les noms de fichiers sélectionnés dans la boîte de dialogue.

Exemple d'ouverture d'un fichier en écriture :

```
# Ouverture de la boîte de dialogue
file_dialog = QFileDialog(self)

# Récupération du nom du fichier que l'on veut créer
output_file_name = file_dialog.getSaveFileName()[0]

# Ouverture du fichier en écriture ("w" pour write)
output_file = open(output_file_name, "w")

# On écrit dans le fichier
output_file.write("Hello World!\n")

# On ferme le fichier
output_file.close()
```

Lire un fichier est tout aussi simple :

```
# Boîte de dialogue pour ouvrir un fichier
file_dialog = QFileDialog(self)
file_dialog.setFileMode(QFileDialog.FileMode.ExistingFiles)

# getOpenFileName() retourne le chemin complet du fichier sélectionné
file_name = file_dialog.getOpenFileNames(self, "Select files...")

# On ouvre le fichier en lecture ("r" pour read)
file = open(file_name, "r")

# On lit le fichier ligne par ligne et on l'affiche dans le terminal
```

```
for line in file:
```

```
    print(line)
```

```
# On ferme le fichier
```

```
file.close()
```