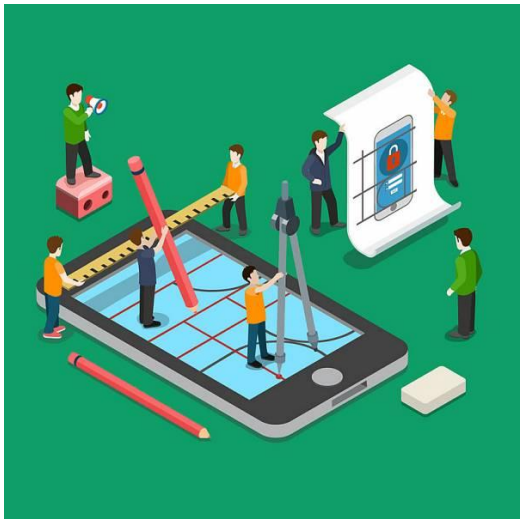
CIP de FP BATOI

DESENROTLLAMENT D'INTERFÍCIES

2N DAM

SA2: Creació i us de components.



PySide Python for Qt

Qt

WIDGETS DE FORMULARIS EN PYSIDE 6

Contenido

1. Etiquetes QLabel

Comencem aquest tour amb les etiquetes, un dels widgets més senzills de Qt. Es tracta d'una peça de text que es pot posicionar a la nostra aplicació. Podem assignar el text en crear-les o mitjançant el mètode setText():

```
from PySide6.QtWidgets import QApplication, QMainWindow, QLabel
from PySide6.QtCore import QSize

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setMinimumSize(QSize(480, 320))

        # widget etiqueta
        etiqueta = QLabel("Soy una etiqueta")
        # establecemos el widget central
        self.setCentralWidget(etiqueta)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

Les etiquetes permeten configurar una font a través de la qual controlar el tamany i altres atributs.

Podem recuperar la font per defecte i augmentar-ne el seu tamany:

```
# recuperamos la fuente por defecto
fuente = etiqueta.font()
# establecemos un tamaño
fuente.setPointSize(24)
# la asignamos a la etiqueta
etiqueta.setFont(fuente)
```

O podem utilitzar una font del sistema, encara que per això hem de crear una instància de la classe QFont:

```
from PySide6.QtGui import QFont  # nuevo

# cargamos una fuente del sistema
fuente = QFont("Comic Sans MS", 24)
# la asignamos a la etiqueta
etiqueta.setFont(fuente)
```

Les etiquetes també ens permeten alinear-les respecte al seu contenidor, per això necessitem importar les definicions estàndard de Qt:

```
from PySide6.QtCore import QSize, Qt  # editado

# establecemos unas flags de alineamiento
etiqueta.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
```

Aquestes son les posibilitats d'alineació:

- **Qt.AlignHCenter | Qt.AlignVCenter** → texto centrado totalmente.
- **Qt.AlignHCenter | Qt.AlignTop** → texto centrado horizontalmente, pero arriba.
- **Qt.AlignLeft | Qt.AlignTop** → texto arriba a la izquierda.
- **Qt.AlignRight | Qt.AlignBottom** → texto abajo a la derecha.

Una altra cosa molt útil que permeten les etiquetes és carregar imatges al seu interior, per això utilitzem un objecte de tipus QPixemap o mapa de píxels creat a partir d'una imatge i l'assignem a l'etiqueta:

```
from PySide6.QtGui import QFont, QPixmap  # editado

# creamos la imagen
imagen = QPixmap("naturaleza.jpg")
# la asginamos a la etiqueta
etiqueta.setPixmap(imagen)
```

Quan establim naturaleza.jpg, el programa espera que aquest recurs es trobi al mateix directori des d'on s'executa l'script. Per això hem de tenir en compte que és possible executar un script sense estar al seu mateix directori i si ho fem aquests recursos no es trobaran.

Alternativament podem utilitzar el mòdul Path de Python per generar una ruta absoluta al recurs concret a partir de l'script actual i resoldre el problema per sempre. Us recomane crear una funció com la següent per facilitar la reutilització:

```
def absPath(file):
    # Devuelve la ruta absoluta a un fichero desde el propio
script
```

```
    return str(Path(__file__).parent.absolute() / file)

print(absPath("naturaleza.jpg"))
imagen = QPixmap(absPath("naturaleza.jpg"))
```

- **Path(__file__).parent** obtiene la **carpeta** donde está ese archivo.
- **.absolute()** convierte esa ruta en una ruta **absoluta completa**, sin "../" ni referencias relativas.
- **/ file** usa el operador / sobre objetos Path, que en pathlib sirve para **unir rutas** (por ejemplo "/home/usuario/scripts" / "datos.json" → "/home/usuario/scripts/datos.json").
- **str(...)** convierte el objeto Path resultante a una **cadena de texto**.

Sigua com sigua la nostra imatge ja es mostrà, però si volguéssim que es reescale al costat de la mida de la finestra hauríem d'establir l'atribut scaledContents a True:

```
# hacemos que se escale con la ventana
etiqueta.setScaledContents(True)
```

2. Casilles QCheckBox

Seguim el tour veient les caselles de verificació o checkboxes:

```
from PySide6.QtWidgets import QApplication, QMainWindow, QCheckBox  # edited


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # creamos una casilla y la establecemos de widget central
        casilla = QCheckBox("Casilla de verificación")
        self.setCentralWidget(casilla)


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

Serveixen com a alternadors per saber si una opció està marcada o no. Podem connectar un senyal stateChanged per saber quan canvia i consultar-ne el valor:

```
# señal para detectar cambios en la casilla
casilla.stateChanged.connect(self.estado_cambiado)

def estado_cambiado(self, estado):
```

```
    print(estado)
```

Fixeu-vos que curiosament els estats numèrics de la casella són 0 i 2. Quan és zero l'està desmarcada i quan és dos està marcada.

I per curiositat, si és desmarcat 0 i marcat 2, a què fa referència l'estat 1? Aquest estat s'anomena triestat i indica que una casella no està estrictament ni marcada ni desmarcada, sinó en estat neutre:

```
# establecemos el triestado por defecto, también funcionan los otros
dos
casilla.setCheckState(Qt.PartiallyChecked)

def estado_cambiado(self, estado):
        if (estado==2):
            print("Casilla marcada")
        if (estado==0):
            print("Casilla desmarcada")
        if (estado==1):
            print("Casilla parcialmente desmarcada")
```

També podem desactivar la casella utilitzant el seu mètode setEnabled, que és comú a la majoria de widgets:

```
# la podemos desactivar
casilla.setEnabled(False)
```

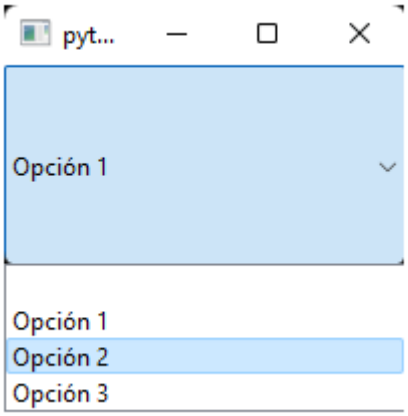
3. Botons radials QRadioButton

Molt semblants a les caselles són els botons radials, però aquests només es poden marcar o desmarcar, no tenen estat neutre:

```
from PySide6.QtWidgets import QApplication, QMainWindow, QRadioButton

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # creamos un botón radial y lo establecemos de widget central
        radial = QRadioButton("Botón radial")
        self.setCentralWidget(radial)

        # señal para detectar cambios en el botón
        radial.toggled.connect(self.estado_cambiado)

        # Podemos activarla por defecto
        radial.setChecked(True)

        # consultamos el valor actual
        print("¿Activada?", radial.isChecked())
```

```
    def estado_cambiado(self, estado):
        if estado:
            print("Radial marcado")
        else:
            print("Radial desmarcado")
```

4. Desplegables QComboBox



Els desplegables són llistes d'opcions de les quals podeu seleccionar una única opció.

Per afegir opcions s'utilitza el mètode addItems:

```
from PySide6.QtWidgets import QApplication, QMainWindow,
QComboBox  # edited

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # creamos un desplegable
        desplegable = QComboBox()
        self.setCentralWidget(desplegable)

        desplegable.addItems(["Opción 1", "Opción 2", "Opción
3"])
```

Depenent de si volem consultar l'índex o el valor en canviar podem fer servir un senyal currentTextChanged o currentIndexChanged:

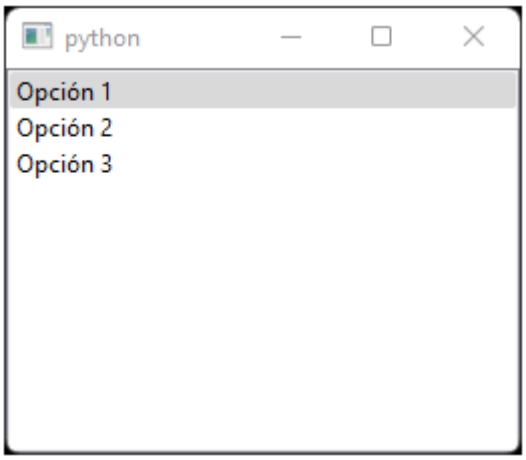
```
desplegable.currentIndexChanged.connect(self.indice_cambiado)
desplegable.currentTextChanged.connect(self.texto_cambiado)

def indice_cambiado(self, indice):
    print("Nuevo índice ->", indice)
```

```
def texto_cambiado(self, texto):
    print("Nuevo texto ->", texto)
```

I per comprovar l'opció seleccionada:

```
# consultamos el valor actual
print("Índice actual ->", desplegable.currentIndex())
print("Texto actual ->", desplegable.currentText())
```

5. Llistes QListWidget



Les llistes són molt semblants als desplegables, però aquí les opcions no estan ocultes ni hi ha cap activa per defecte. En lloc d'índexs manegen un tipus de valor anomenat QItem i el senyal de canvi aquí és currentItemChanged:

```
from PySide6.QtWidgets import QApplication, QMainWindow,
QListWidget

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # creamos una lista
        lista = QListWidget()
        self.setCentralWidget(lista)

        # Añadimos algunas opciones
        lista.addItems(["Opción 1", "Opción 2", "Opción 3"])

        # Y algunas señales
        lista.currentItemChanged.connect(self.item_cambiado)
```

```
    def item_cambiado(self, item):
        # Conseguimos el texto del ítem con su método text()
        print("Nuevo ítem ->", item.text())
```

I per aconseguir l'ítem actual farem servir:

```
print(lista.currentItem())
```

6. Camps de text QLineEdit

Els camps de text són els widgets que permeten capturar contingut escrit per l'usuari:

```
from PySide6.QtWidgets import QApplication, QMainWindow,
QLineEdit

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # creamos un campo de texto
        texto = QLineEdit()
        self.setCentralWidget(texto)

        # Probamos algunas opciones
        texto.setMaxLength(10)
        texto.setPlaceholderText("Escribe máximo 10 caracteres")
```

Podem provar alguns senyals per practicar:

```
# Probamos algunas señales
texto.textChanged.connect(self.texto_cambiado)
texto.returnPressed.connect(self.enter_presionado)

def texto_cambiado(self, texto):
    print("Texto cambiado ->", texto)

def enter_presionado(self):
    # al presionar enter recuperamos el texto a partir del
widget central
    texto = self.centralWidget().text()
    print("Enter presionado, texto ->", texto)
```

7. Camps numèrics QSpinBox i QDoubleSpinBox

A diferència dels camps de text, els numèrics forcen l'usuari a escriure números i proveeixen mètodes per controlar-los. Trobem per emmagatzemar sencers i decimals, comencem pels sencers:

```
from PySide6.QtWidgets import QApplication, QMainWindow,
QSpinBox

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # creamos un campo numérico entero
        numero = QSpinBox()
        self.setCentralWidget(numero)

        # Probamos algunas opciones
        numero.setMinimum(0)
        numero.setMaximum(10)
        numero.setRange(0, 10)
        numero.setSingleStep(1)

        # Probamos algunas señales
        numero.valueChanged.connect(self.valor_cambiado)

    def valor_cambiado(self, numero):
        # al presionar enter recuperamos el texto a partir del
widget central
        print("Valor cambiado ->", numero)
```

Els camps numèrics permeten establir prefixos i sufixos, útils per manejar monedes i mesures:

```
numero.setPrefix("$")
numero.setSuffix("%")
```

Pel que fa als decimals són exactament el mateix però utilitzant el widget QDoubleSpinBox:

- QSpinBox -> QDoubleSpinBox

- `numero = QSpinBox()` -> `numero = QDoubleSpinBox()`
- `numero.setSingleStep(1)` -> `numero.setSingleStep(0.5)`

A tots dos widgets podem establir un valor per defecte amb el mètode setValue i recuperar-lo amb value:

```
# Establecer y recuperar el valor
numero.setValue(3.14)
print(numero.value())
```

8. Credits

Aquests apunts són un material de suport per als alumnes de DAM i han sigut desenvolupats per Herctor Docs i publicats sota una Llicència CC BY 4.0. Estos apunts han sigut modificats i adaptats per Iván Martos.