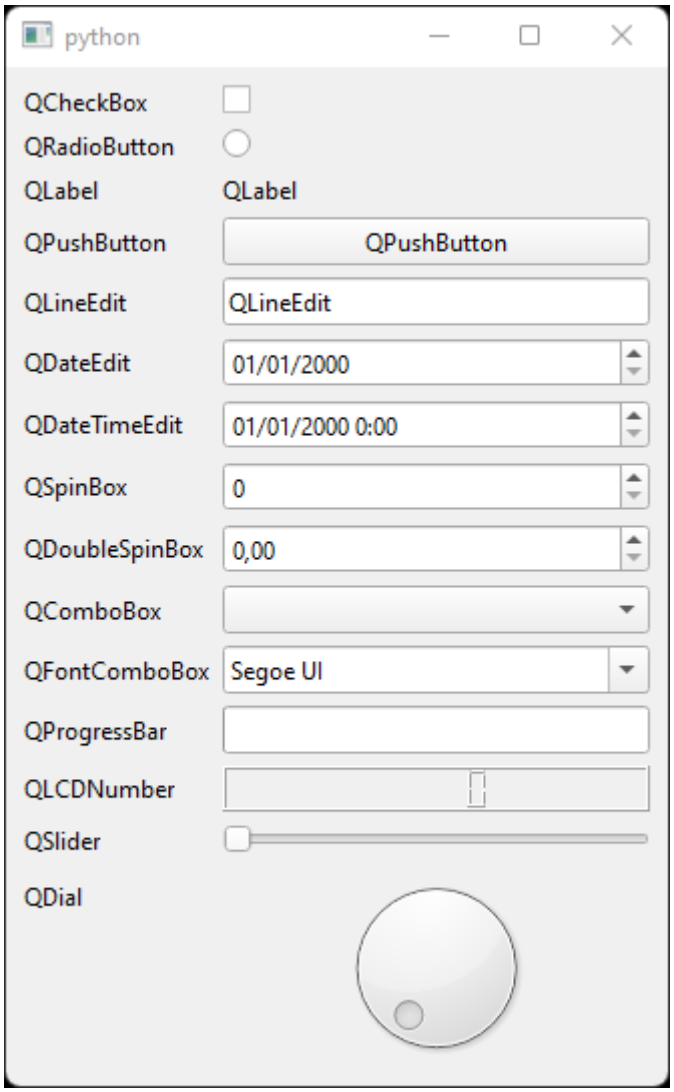
Estilos de aplicación



Los estilos modifican la estética de los componentes. Por defecto Qt aplica estilos específicos para cada plataforma para integrar la aplicación visualmente, esa es la razón por la que el mismo programa se verá diferente en Windows, Linux y Mac.

Los estilos se pueden personaliar para no hacerlos dependientes de la plataforma y de hecho el propio Qt tiene un tema llamado **Fusion** que provee una estética multiplataforma y moderna.

He preparado un formulario con todo tipo de widgets para que podamos apreciar los cambios visuales al cambiar los estilos:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QFormLayout, QWidget, QLineEdit,
QSpinBox)
from PySide6.QtCore import Qt
import sys
```

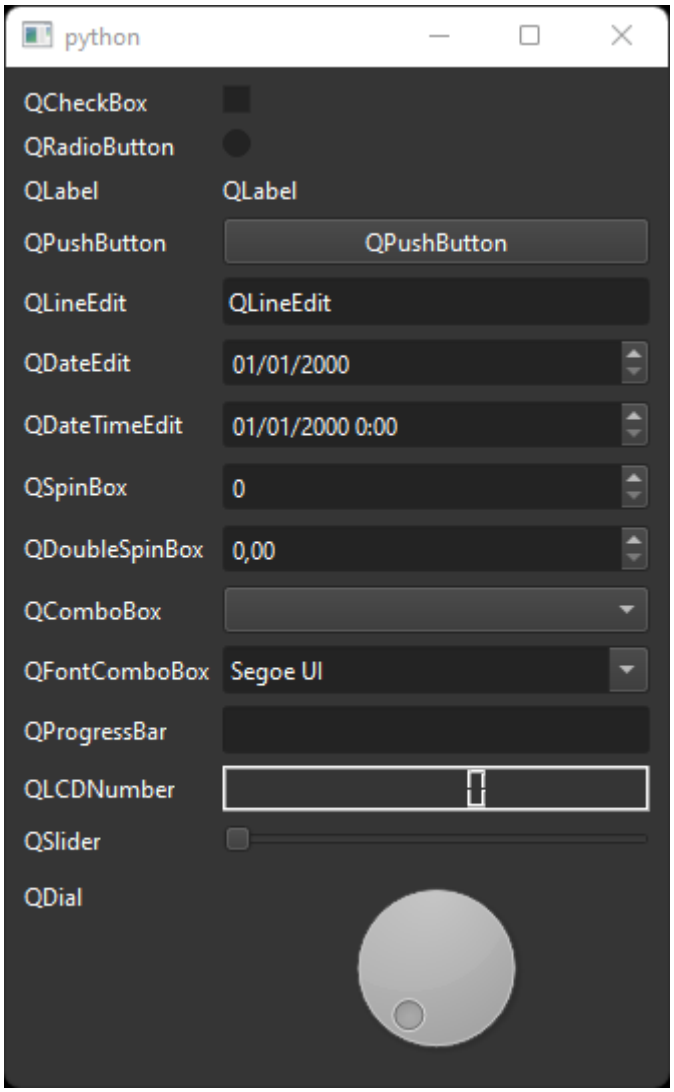
```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        formulario = QFormLayout()

        formulario.addRow("Nombre", QLineEdit("Hector"))
        formulario.addRow("Email", QLineEdit(text="hola@ejemplo.com"))
        formulario.addRow("Edad", QSpinBox(value=32))

        widget = QWidget()
        widget.setLayout(formulario)

        self.setCentralWidget(widget)


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

Para activar el tema `Fusion` basta con llamar al método `setStyle` de la aplicación:

```
if __name__ == "__main__":
    app = QApplication(sys.argv)

    # estilo fusion
    app.setStyle("Fusion")

    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

Como podréis observar son cambios muy sutiles que por lo menos en Windows cuesta apreciar a simple vista, pero así nos aseguramos de visualizar la misma estética en todas las plataformas.

Paletas de colores (QPalette)



La selección de colores que utiliza Qt para dibujar los componentes se maneja en paletas.

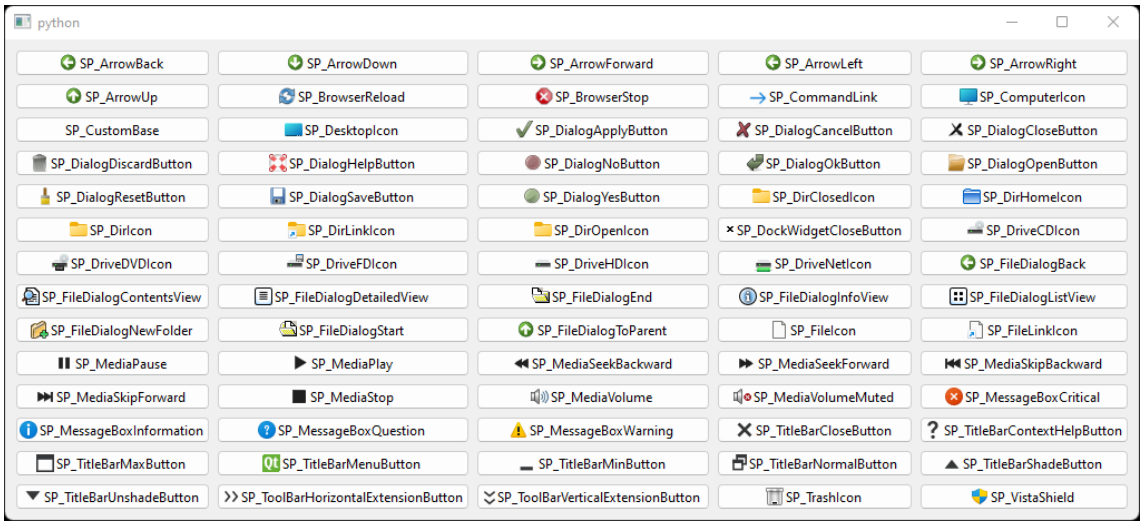
Vamos a experimentar con la paleta de colores de la aplicación de la lección anterior:

```
from PySide6.QtGui import QPalette, QColor  # nuevo

if __name__ == "__main__":
    app = QApplication(sys.argv)

    # creamos nuestra paleta de colores
    paleta = QPalette()
    paleta.setColor(QPalette.Window, QColor(51, 51, 51))
    paleta.setColor(QPalette.WindowText, QColor(235, 235, 235))

    # activamos la paleta en la aplicación
    app.setPalette(paleta)

    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

Como véis las paletas tienen accesores para establecer los colores de los diferentes componentes. He encontrado una paleta llamada Dark Fusion bastante chula para cambiar la apariencia a modo oscuro, os la dejo en los recursos, así como un enlace a la documentación con los atributos configurables de la paleta:

```
if __name__ == "__main__":
    app = QApplication(sys.argv)

    # dark fusion
https://gist.github.com/lschmierer/443b8e21ad93e2a2d7eb
    app.setStyle("Fusion")
    dark_fusion = QPalette()
    dark_fusion.setColor(QPalette.Window, QColor(53, 53, 53))
    dark_fusion.setColor(QPalette.WindowText, Qt.white)
    dark_fusion.setColor(QPalette.Base, QColor(35, 35, 35))
    dark_fusion.setColor(QPalette.AlternateBase, QColor(53, 53, 53))
    dark_fusion.setColor(QPalette.ToolTipBase, QColor(25, 25, 25))
    dark_fusion.setColor(QPalette.ToolTipText, Qt.white)
    dark_fusion.setColor(QPalette.Text, Qt.white)
    dark_fusion.setColor(QPalette.Button, QColor(53, 53, 53))
    dark_fusion.setColor(QPalette.ButtonText, Qt.white)
    dark_fusion.setColor(QPalette.BrightText, Qt.red)
    dark_fusion.setColor(QPalette.Link, QColor(42, 130, 218))
    dark_fusion.setColor(QPalette.Highlight, QColor(42, 130, 218))
    dark_fusion.setColor(QPalette.HighlightedText, QColor(35, 35, 35))
    dark_fusion.setColor(QPalette.Active, QPalette.Button, QColor(53,
53, 53))
    dark_fusion.setColor(QPalette.Disabled, QPalette.ButtonText,
Qt.darkGray)
    dark_fusion.setColor(QPalette.Disabled, QPalette.WindowText,
Qt.darkGray)
    dark_fusion.setColor(QPalette.Disabled, QPalette.Text,
Qt.darkGray)
    dark_fusion.setColor(QPalette.Disabled, QPalette.Light, QColor(53,
53, 53))
    # activamos la paleta en la aplicación
    app.setPalette(dark_fusion)

    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

Utilización de iconos



Anteriormente ya hemos utilizado algunos iconos cargándolos como recursos externos, pero Qt incluye un set de iconos predeterminados. Podemos hacer uso de ellos de la siguiente forma:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QPushButton, QStyle)  # edited
import sys

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

        # recuperamos el icono de la librería estandard de la ventana
        icono = self.style().standardIcon(QStyle.SP_DialogSaveButton)
        # lo podemos asignar a un botón
        boton = QPushButton(icono, "Botón guardar")

        self.setCentralWidget(boton)
```

He creado un pequeño programa para visualizar los iconos de la librería estandar dinámicamente, os lo adjunto en los recursos:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QPushButton, QStyle,
QGridLayout)
import sys

iconos = ['SP_ArrowBack', 'SP_ArrowDown', 'SP_ArrowForward',
'SP_ArrowLeft', 'SP_ArrowRight', 'SP_ArrowUp', 'SP_BrowserReload',
'SP_BrowserStop', 'SP_CommandLink', 'SP_ComputerIcon',
'SP_CustomBase', 'SP_DesktopIcon', 'SP_DialogApplyButton',
'SP_DialogCancelButton', 'SP_DialogCloseButton',
'SP_DialogDiscardButton', 'SP_DialogHelpButton', 'SP_DialogNoButton',
'SP_DialogOkButton', 'SP_DialogOpenButton', 'SP_DialogResetButton',
'SP_DialogSaveButton', 'SP_DialogYesButton', 'SP_DirClosedIcon',
'SP_DirHomeIcon', 'SP_DirIcon', 'SP_DirLinkIcon', 'SP_DirOpenIcon',
'SP_DockWidgetCloseButton', 'SP_DriveCDIcon', 'SP_DriveDVDIcon',
'SP_DriveFDIcon', 'SP_DriveHDIcon', 'SP_DriveNetIcon',
'SP_FileDialogBack', 'SP_FileDialogContentsView',
'SP_FileDialogDetailedView', 'SP_FileDialogEnd',
'SP_FileDialogInfoView', 'SP_FileDialogListView',
'SP_FileDialogNewFolder', 'SP_FileDialogStart',
```

```
'SP_FileDialogToParent', 'SP_FileIcon', 'SP_FileLinkIcon',
'SP_MediaPause', 'SP_MediaPlay', 'SP_MediaSeekBackward',
'SP_MediaSeekForward', 'SP_MediaSkipBackward', 'SP_MediaSkipForward',
'SP_MediaStop', 'SP_MediaVolume', 'SP_MediaVolumeMuted',
'SP_MessageBoxCritical', 'SP_MessageBoxInformation',
'SP_MessageBoxQuestion', 'SP_MessageBoxWarning',
'SP_TitleBarCloseButton', 'SP_TitleBarContextHelpButton',
'SP_TitleBarMaxButton', 'SP_TitleBarMenuButton',
'SP_TitleBarMinButton', 'SP_TitleBarNormalButton',
'SP_TitleBarShadeButton', 'SP_TitleBarUnshadeButton',
'SP_ToolBarHorizontalExtensionButton',
'SP_ToolBarVerticalExtensionButton', 'SP_TrashIcon', 'SP_VistaShield']


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

        # creo un layaout en cuadrícula
        layout = QGridLayout()

        # recorro los iconos con un contador de posicion
        for contador, nombre in enumerate(iconos):
            # recupero el icono a partir de su nombre
            icono = self.style().standardIcon(getattr(QStyle, nombre))
            # creo un botón con el icono y su nombre del icono
            boton = QPushButton(icono, nombre)
            # añado el boton en una cuadrícula de 5 columnas
            # divido el contador entre 5 para conseguir la fila
            # con el módulo de la divisón entre 5 conseguiré la
columna
            layout.addWidget(boton, contador // 5, contador % 5)

        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
```

Si necesitamos más iconos podemos imporarlos de recursos externos como ya vimos, o utilizar una colección como `qtawesome`. Lo malo es que a fecha de creación del curso esta colección todavía no está soportada por `PySide6`, podemos usarla cambiando las importaciones a `PySide2` si lo tenemos instalado:

```
pip install pyside2 qtawesome
```

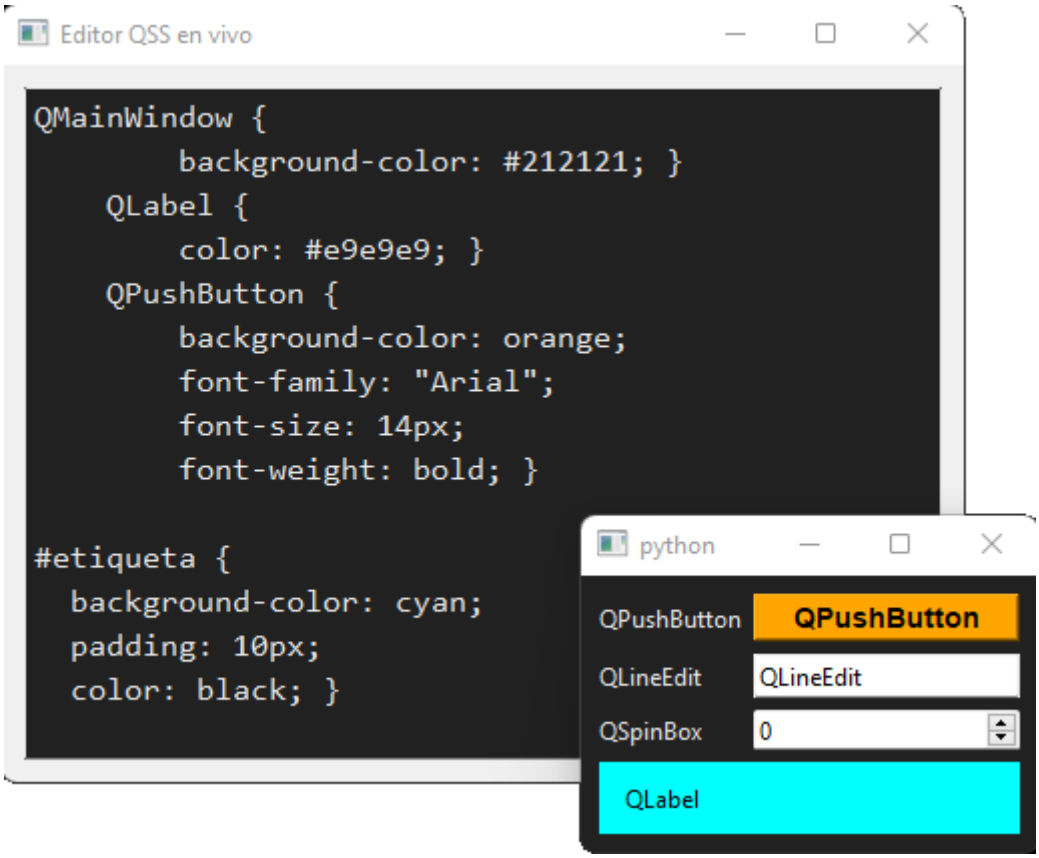
Para recuperar los iconos es muy fácil:

```
from PySide2.QtWidgets import (
    QApplication, QMainWindow, QWidget, QPushButton)  # edited PySide2
import sys
import qtawesome as qta


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

        # recuperamos un icono de qta y lo añadimos a un botón
        icono = qta.icon('fa5b.github')
        boton = QPushButton(icono, "Github")
```

```
        self.setCentralWidget(boton)
```

Podéis encontrar información sobre como personalizar los iconos y las colecciones disponibles en el repositorio de qtawesome.

QSS: Qt Style Sheets



Lo último que veremos sobre tematización son las hojas de estilo de Qt, o abreviadas QSS.

Si sabéis algo de programación web seguro que os suena el lenguaje `CSS`, pues `QSS` es una forma de añadir estilo a los widgets utilizando prácticamente la misma sintaxis.

En esta práctica vamos a personalizar unos cuantos widgets básicos dentro de un layout estilo formulario:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QFormLayout, QWidget, QLabel,
QRadioButton,
    QCheckBox, QLineEdit, QSpinBox, QPushButton, QPlainTextEdit)
from pathlib import Path
import sys
```

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


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

        formulario = QFormLayout()
        formulario.addRow("QCheckBox", QCheckBox())
        formulario.addRow("QRadioButton", QRadioButton())
        formulario.addRow("QLabel", QLabel("QLabel"))
        formulario.addRow("QPushButton", QPushButton("QPushButton"))
        formulario.addRow("QLineEdit", QLineEdit("QLineEdit"))
        formulario.addRow("QSpinBox", QSpinBox())

        widget = QWidget()
        widget.setLayout(formulario)
        self.setCentralWidget(widget)
```

La forma más sencilla de establecer los estilos es a través del método `setSyleSheets` del widget principal, pues con él podemos dar estilo a todo lo que contiene:

```
# estilos QSS
self.setStyleSheet("""
    QMainWindow {
        background-color: #212121; }
    QLabel {
        color: #e9e9e9; }
    QPushButton {
        background-color: orange;
        font-family: "Arial";
        font-size: 14px;
        font-weight: bold; }
""")
```

Ahora bien, estos estilos son globales y afectan a todas las instancias. Si queremos estilizar una sola instancia podemos otorgarle un identificador:

```
etiqueta = QLabel("QLabel")
etiqueta.setObjectName("etiqueta")
formulario.addRow(etiqueta)
```

Y referirnos a ella en QSS usando la almohadilla igual que en CSS:

```
"""
#etiqueta {
  background-color: cyan;
  padding: 10px;
  color: black; }
"""
```

La verdad es que este tema abarca mucho y no quiero extenderme, os dejaré la documentación con todas las propiedades disponibles y cada uno que profundice en la medida de lo necesario.

Lo que sí quiero compartir con vosotros es un pequeño widget para probar estilos en vivo, se basa en crear una subventana con un pequeño editor. Esto os

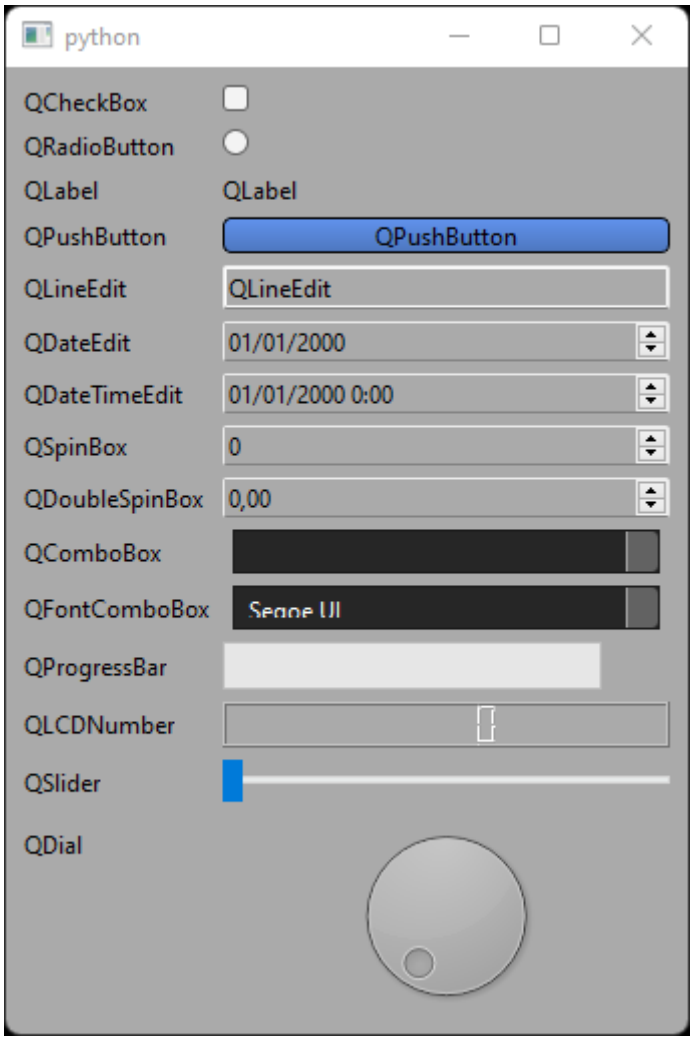
ayudará a tematizar vuestros programas sin tener que guardar y ejecutar el código mil veces:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QFormLayout, QWidget, QLabel,
    QLineEdit, QSpinBox, QPushButton, QPlainTextEdit, QVBoxLayout)
import sys


class EditorQSS(QWidget):
    def __init__(self, parent):
        super().__init__()
        self.parent = parent
        self.resize(480, 320)
        self.setWindowTitle("Editor QSS en vivo")

        self.editor = QPlainTextEdit()
        self.editor.setStyleSheet(
            "background-color: #212121; color: #e9e9e9; font-family:
Consolas; font-size: 16px; ")
        self.editor.setFont("Consolas")
        self.editor.textChanged.connect(self.actualizar_estilos)

        layout = QVBoxLayout()
        layout.addWidget(self.editor)
        self.setLayout(layout)

        self.show()

    def actualizar_estilos(self):
        qss = self.editor.toPlainText()
        try:
            self.parent.setStyleSheet(qss)
        except:
            pass
```

Simplemente tenemos que crear una instancia en nuestro programa pasándole como parámetro el widget de la ventana y ya podemos empezar a probar estilos:

```
# editor QSS en vivo
self.editorQSS = EditorQSS(self)
```

Cargando ficheros QSS



En esta última lección vamos a cargar ficheros QSS para no tener que escribir el código en el propio programa. He preparado un buen puñado de estilos que he encontrado por Internet, os los adjunto en los recursos, sentíos libres de utilizarlos respetando las directrices de cada creador, cuya fuente encontraréis en la cabecera de cada fichero.

Tengo un programa ya preparado para empezar a trabajar:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QFormLayout, QWidget, QLabel,
    QRadioButton, QCheckBox, QLineEdit, QSpinBox, QDoubleSpinBox,
    QPushButton, QComboBox, QFontComboBox, QDateEdit, QDateTimeEdit,
    QLCDNumber, QProgressBar, QDial, QSlider)
from PySide6.QtCore import Qt
from pathlib import Path
import sys


def absPath(file):
    return str(Path(__file__).parent.absolute() / file)


class MainWindow(QMainWindow):
    def __init__(self):
```

```
        super().__init__()

        formulario = QFormLayout()

        formulario.addRow("QCheckBox", QCheckBox())
        formulario.addRow("QRadioButton", QRadioButton())
        formulario.addRow("QLabel", QLabel("QLabel"))
        formulario.addRow("QPushButton", QPushButton("QPushButton"))
        formulario.addRow("QLineEdit", QLineEdit("QLineEdit"))
        formulario.addRow("QDateEdit", QDateEdit())
        formulario.addRow("QDateTimeEdit", QDateTimeEdit())
        formulario.addRow("QSpinBox", QSpinBox())
        formulario.addRow("QDoubleSpinBox", QDoubleSpinBox())
        formulario.addRow("QComboBox", QComboBox())
        formulario.addRow("QFontComboBox", QFontComboBox())
        formulario.addRow("QProgressBar", QProgressBar())
        formulario.addRow("QLCDNumber", QLCDNumber())
        formulario.addRow("QSlider", QSlider(Qt.Horizontal))
        formulario.addRow("QDial", QDial())

        widget = QWidget()
        widget.setLayout(formulario)

        self.setCentralWidget(widget)


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

Para cargar los estilos me ayudaré de la función `absPath`, abriré los ficheros de estilo como si fueran texto, leeré su contenido y lo volcaré al método `setStyleSheet` de la ventana:

```
def cargarQSS(self, file):
    # guardamos la ruta absoluta al fichero
    path = absPath(file)
    # intentamos abrirlo y volcar el contenido
    try:
        with open(path) as styles:
            self.setStyleSheet(styles.read())
    # si hay algún fallo lo capturamos con una excepción genérica
    except:
        print("Error abriendo estilos", path)
```

Solo resta llamar al método y probar algunos temas:

```
# cargamos los estilos del fichero
self.cargarQSS("qss/Ubuntu.qss")
self.cargarQSS("qss/ElegantDark.qss")
self.cargarQSS("qss/ChatBee.qss")
self.cargarQSS("qss/EasyCode.qss")
```

Con esto tenéis toneladas de referencias para tematizar vuestros programas.