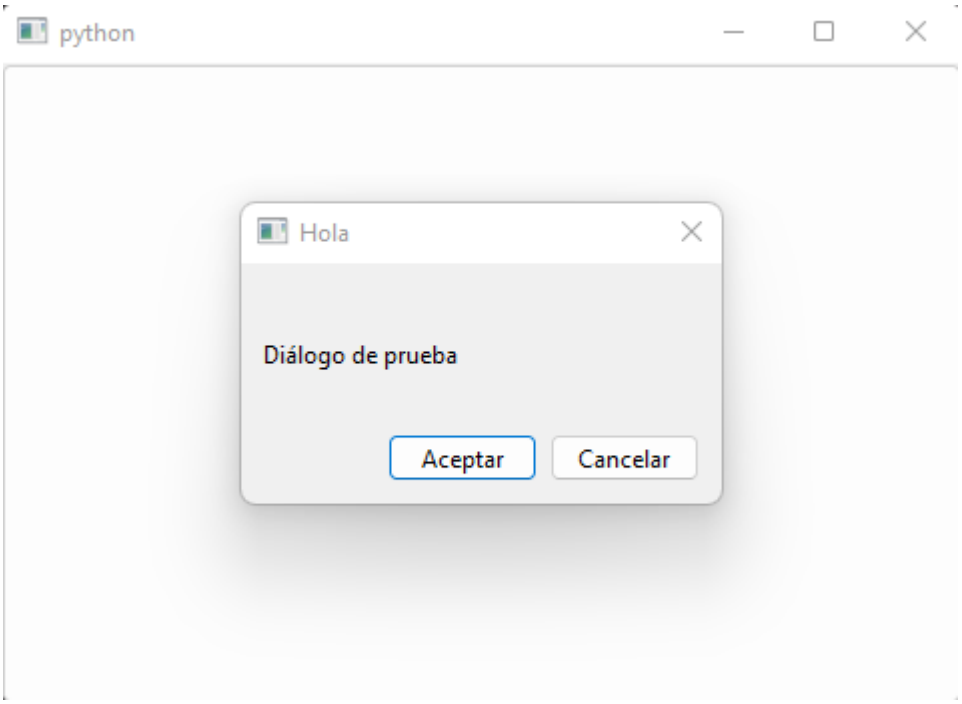
Diálogos personalizados QDialog



Todos los diálogos parten de la clase `QDialog` y están pensados para mostrar o pedir información al usuario en una nueva ventana temporal. Al estar pensados para aparecer de forma emergente, los diálogos requieren una acción desencadenante para aparecer.

Vamos a partir de un diálogo vacío para ver su funcionamiento:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QDialog)
import sys


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.resize(480, 320)  # forma alternativa de redimensionar un
widget

        boton = QPushButton("Mostrar diálogo")
        boton.clicked.connect(self.boton_clicado)
        self.setCentralWidget(boton)

    def boton_clicado(self):
        dialogo = QDialog(self)
        dialogo.setWindowTitle("Hola")
        dialogo.exec_()


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

```
    sys.exit(app.exec_())
```

El diálogo `QDialog` está pensado para extenderlo, vamos a personalizar algunas opciones creando nuestro propio diálogo heredando de esta clase:

```
class Dialogo(QDialog):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Hola")

def boton_clicado(self):
    dialogo = Dialogo()
    dialogo.exec_()
```

Lo más esencial que nos permite un diálogo es interactuar con él a través de botones. Estos botones se configuran en su propia caja de botones llamada `QDialogButtonBox` que normalmente pondremos dentro de un layout donbe organizar el espacio:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow,
    QPushButton, QLabel, QVBoxLayout,
    QDialog, QDialogButtonBox)
import sys


class Dialogo(QDialog):
    def __init__(self):
        super().__init__()
        self.resize(240, 120)
        self.setWindowTitle("Hola")

        # creamos un layout y lo establecemos en el widget
        layout = QVBoxLayout()
        self.setLayout(layout)

        # podemos añadir una etiqueta
        layout.addWidget(QLabel("Diálogo de prueba"))

        # creamos unos botones predeterminados
        botones = QDialogButtonBox(
            QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

        # y los añadimos al layout
        layout.addWidget(botones)
```

Los acciones básicas contra un `QDialog` son aceptarlo o denegarlo, ambas se deben configurar como señales en la caja de botones usando los métodos `accept` y `reject`:

```
# configuramos unas señales predeterminadas
botones.accepted.connect(self.accept)
botones.rejected.connect(self.reject)
```

El sistema entenderá que el botón "Ok" indica aceptar el mensaje y "Cancel" lo contrario, retornando la respuesta al exterior para poder actuar en consecuencia:

```
def boton_clicado(self):
    dialogo = Dialogo()
    respuesta = dialogo.exec_()
    if respuesta:
```

```
        print("Diálogo aceptado")
    else:
        print("Diálogo denegado")
```

Si deseamos que un diálogo emerja sobre la ventana donde se ha creado hay que pasarle la instancia del padre al crearlo:

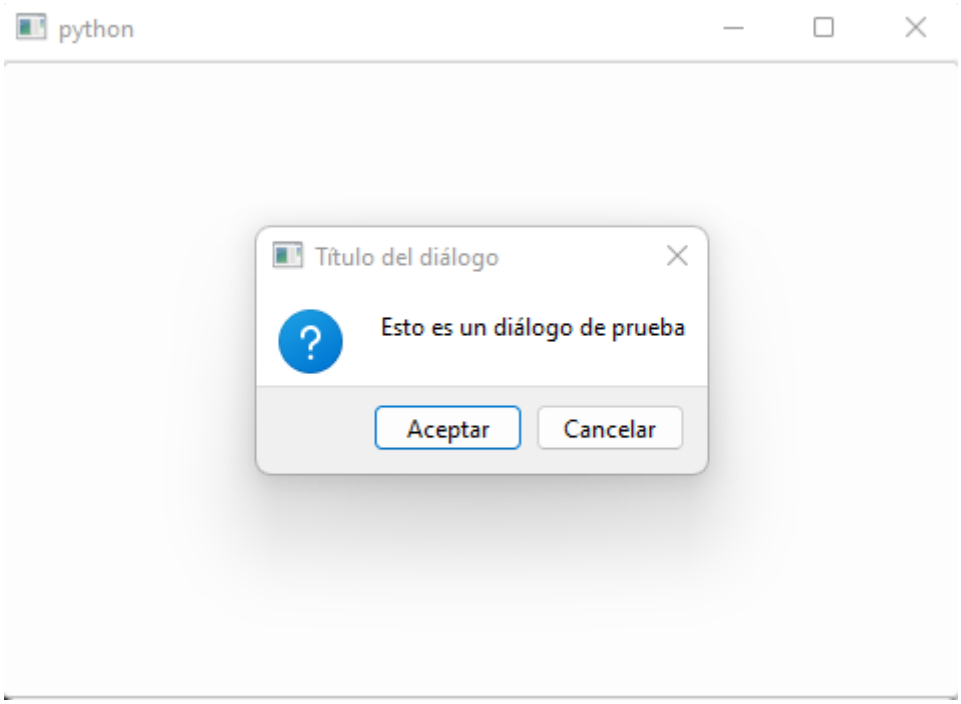
```
class Dialogo(QDialog):
    def __init__(self, parent=None):  # editado
        super().__init__(parent)      # editado

def boton_clicado(self):
    dialogo = Dialogo(self)  # editado
    respuesta = dialogo.exec_()
    if respuesta:
        print("Diálogo aceptado")
    else:
        print("Diálogo denegado")
```

Por cierto, los botones del diálogo se encuentran localizados, es decir que permiten traducciones. El problema es que hay que dar de alta un traductor y para dos botones no compensa hacerlo. Es más fácil cambiar el texto a mano:

```
# traducción en tiempo real de los botones
botones.button(QDialogButtonBox.Ok).setText("Aceptar")
botones.button(QDialogButtonBox.Cancel).setText("Cancelar")
```

En los recursos os dejo la documentación con la lista de botones que tenemos a nuestra disposición.

Diálogos de mensaje QMessageBox

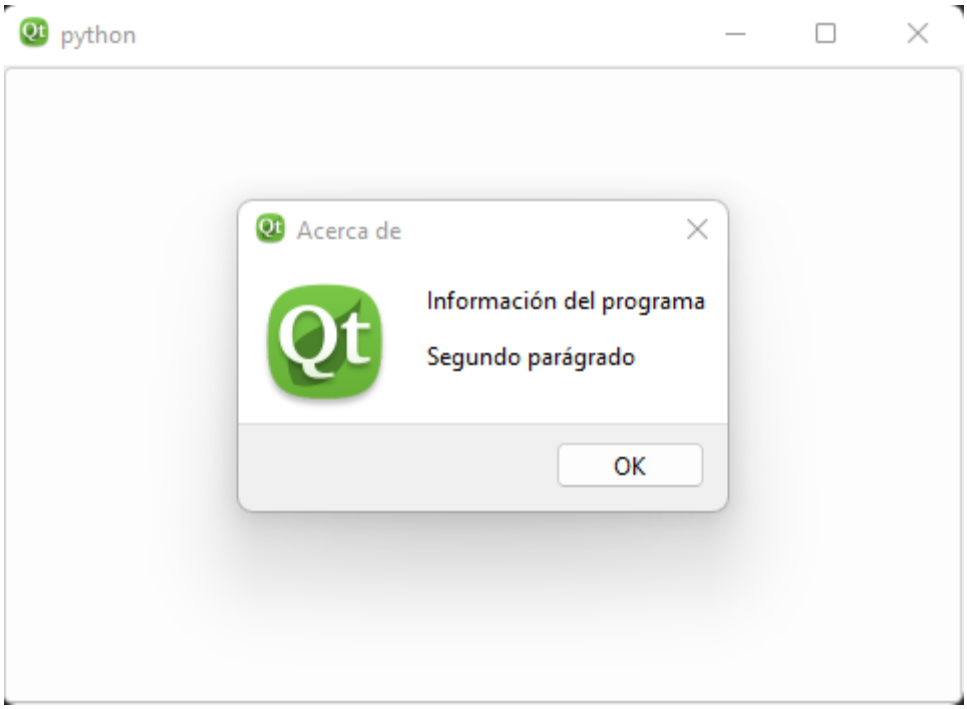


Si lo que deseamos en enviar un mensaje al usuario tenemos a nuestra disposición una clase llamada `QMessageBox` que simplifica la personalización de un `QDialog`. Básicamente podemos hacer todo lo de la anterior lección sin crear nuestra propia clase heredada y además podemos usar iconos predeterminados:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QMessageBox)
import sys


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

        boton = QPushButton("Mostrar diálogo")
        boton.clicked.connect(self.boton_clicado)
        self.setCentralWidget(boton)

    def boton_clicado(self):
        # creamos un diálogo de mensaje con un título y un texto
        dialogo = QMessageBox(self)
        dialogo.setWindowTitle("Título del diálogo")
        dialogo.setText("Esto es un diálogo de prueba")
        # añadimos unos botones y los traducimos
        dialogo.setStandardButtons(QMessageBox.Ok |
QMessageBox.Cancel)
        dialogo.button(QMessageBox.Ok).setText("Aceptar")
        dialogo.button(QMessageBox.Cancel).setText("Cancelar")
        # configuramos un icono
        dialogo.setIcon(QMessageBox.Question)

        # ejecutamos el diálogo y capturamos la respuesta
        respuesta = dialogo.exec_()
        # ahora debemos comprobar qué tipo de botón se ha clicado
        if respuesta == QMessageBox.Ok:
            print("Diálogo aceptado")
        else:
            print("Diálogo denegado")


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

Os dejo documentación sobre los botones e iconos disponibles en los diálogos de mensaje.

Diálogos predeterminados QMessageBox



Por suerte no necesitamos crear diálogos todo el tiempo, Qt incluye diálogos predeterminados para realiar diferentes tareas:

Mensaje de cuestión QMessageBox.question

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QMessageBox)
import sys

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

        boton = QPushButton("Mostrar diálogo")
        boton.clicked.connect(self.boton_clicado)
        self.setCentralWidget(boton)

    def boton_clicado(self):
        # creamos un diálogo de tipo cuestión
        dialogo = QMessageBox.question(
            self, "Diálogo de cuestión", "Esta es una pregunta de
prueba")

        # ahora podemos comprobar qué tipo de botón se devuelve
        print(dialogo)
        if dialogo == QMessageBox.Yes:
            print("Ha respondido sí")

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

Traducir los diálogos predeterminados no es algo trivial, en la siguiente lección os enseñaré como activar el traductor, por ahora veamos otros ejemplos.

Mensaje acerca de QMessageBox.about

Para mostrar información del programa o el autor:

```
def boton_clicado(self):
    dialogo = QMessageBox.about(
        self, "Acerca de", "<p>Información del programa</p><p>Segundo
parágrado</p>")
```

Este diálogo toma por defecto el icono de la ventana, vamos a añadir uno de ejemplo para verlo, usaremos la función para generar rutas absolutas que os enseñé:

```
self.setWindowIcon(QIcon(absPath("icon.png")))
```

Mensaje crítico QMessageBox.critical

Este diálogo reproduce un sonido de error mientras muestra la ventana:

```
def boton_clicado(self):
    dialogo = QMessageBox.critical(
        self, "Diálogo de error", "Ha ocurrido algo malo")
    print(dialogo)
```

Mensaje informativo

Para mostrar información genérica:

```
def boton_clicado(self):
    dialogo = QMessageBox.information(
        self, "Diálogo informativo", "Esto es un texto informativo")
```

Mensaje de aviso QMessageBox.warning

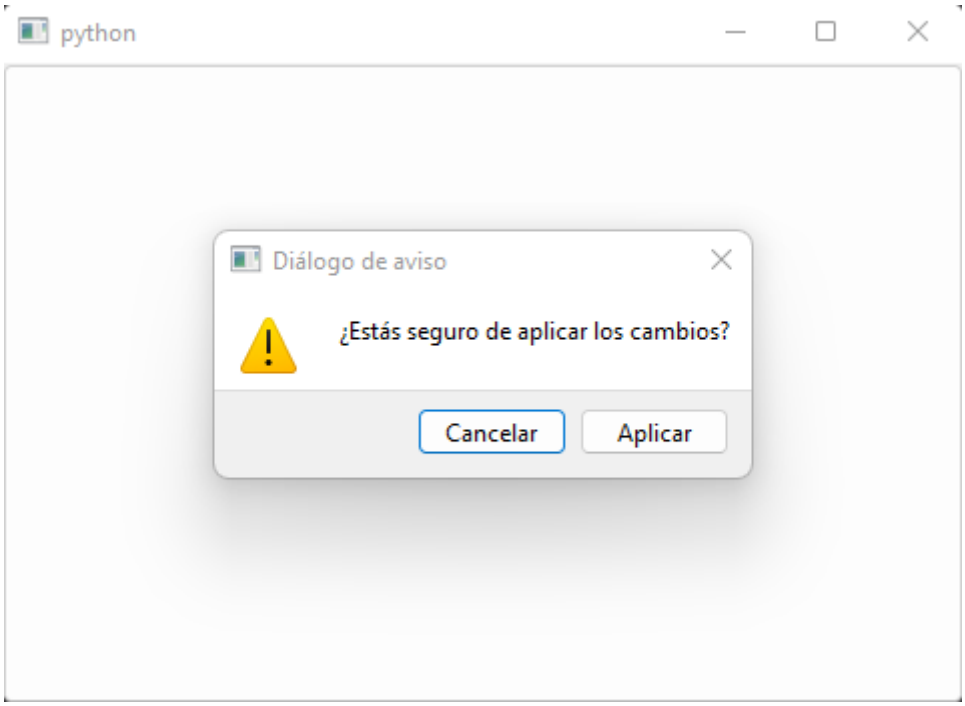
Para mostrar un aviso:

```
def boton_clicado(self):
    dialogo = QMessageBox.warning(
        self, "Diálogo de aviso", "Cuidado con este diálogo")
```

Si deseamos modificar los botones de un mensaje predeterminado podemos hacerlo:

```
def boton_clicado(self):
    dialogo = QMessageBox.warning(
        self, "Diálogo de aviso", "¿Estás seguro de aplicar los
cambios?",
        buttons=QMessageBox.Apply | QMessageBox.Cancel,
        defaultButton=QMessageBox.Cancel)

    if dialogo == QMessageBox.Apply:
        print("Aplicamos los cambios")
```

Activando las traducciones



Traducir los diálogos predeterminados a mano hace que se pierda su simplicidad, por eso es recomendable activar las traducciones. Esto implica que hay que distribuir los ficheros de traducción junto a los ejecutables, pero eso es algo que veremos más adelante.

Para activar la localización al idioma del sistema operativo debemos traducir la aplicación de la siguiente forma:

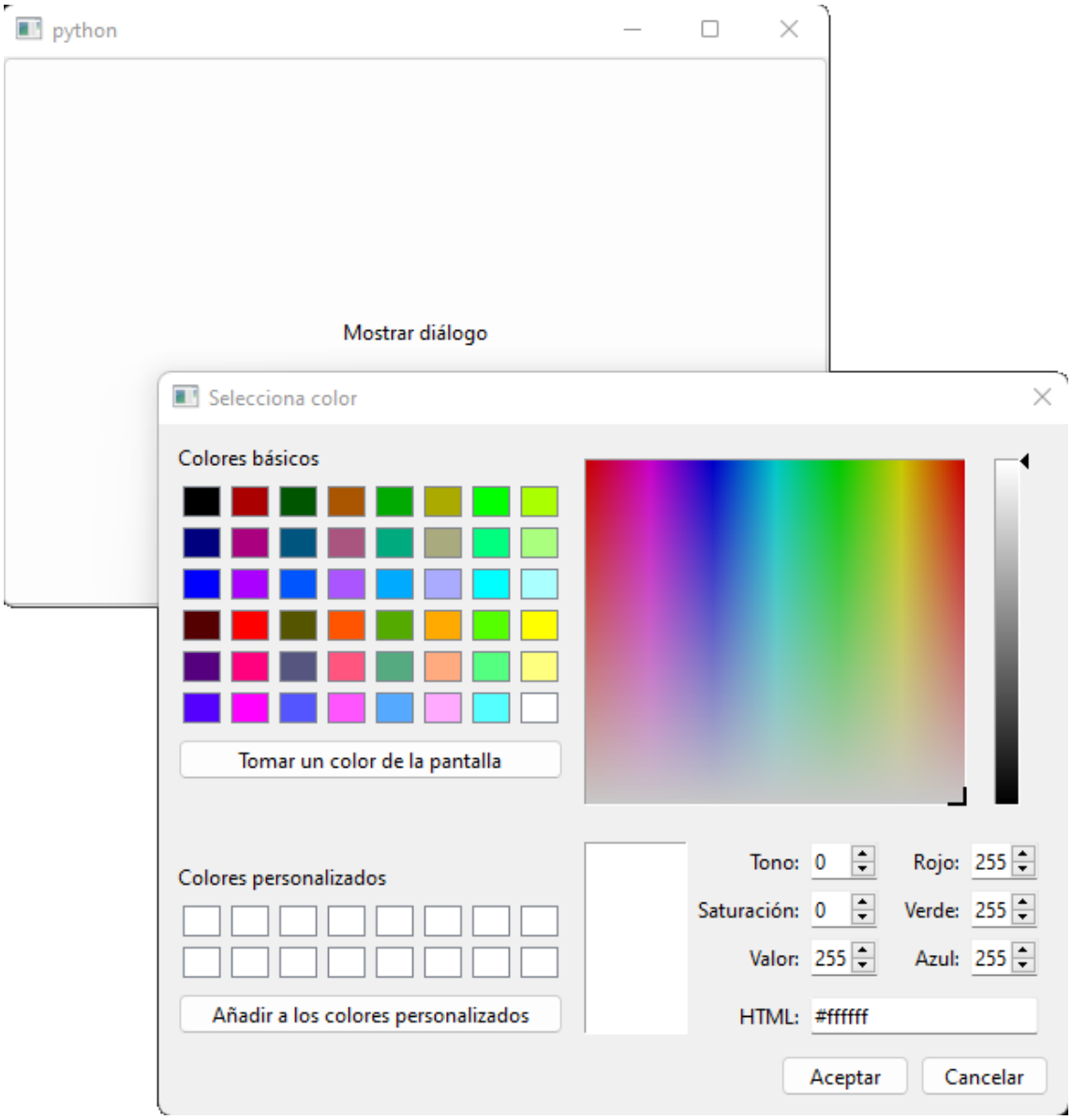
```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QMessageBox)
from PySide6.QtCore import QTranslator, QLibraryInfo  # nuevo
import sys


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

        boton = QPushButton("Mostrar diálogo")
        boton.clicked.connect(self.boton_clicado)
        self.setCentralWidget(boton)

    def boton_clicado(self):
        dialogo = QMessageBox.warning(
            self, "Diálogo de aviso", "¿Estás seguro de aplicar los
cambios?",
            buttons=QMessageBox.Apply | QMessageBox.Cancel,
            defaultButton=QMessageBox.Cancel)

        if dialogo == QMessageBox.Apply:
            print("Aplicamos los cambios")
```

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

    # envolvemos la aplicación con el traductor
    translator = QTranslator(app)
    # recuperamos el directorio de traducciones
    translations =
QLibraryInfo.location(QLibraryInfo.TranslationsPath)
    # cargamos la traducción en el traductor
    translator.load("qt_es", translations)
    # la aplicamos
    app.installTranslator(translator)

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

Podemos consultar los idiomas disponibles en la carpeta de traducciones:

```
print(translations)
cd ruta/
explorer .
```

Diálogos específicos



Vamos a acabar esta unidad viendo otros ejemplos de diálogos para usos específicos. Es importante tener en cuenta que es necesario activar las traducciones o los textos aparecerán en inglés.

Diálogos de fichero QFileDialog

Se utilizan para generar la ruta a un fichero usando el explorador, es decir, no afectan al fichero en sí y solo sirven para saber donde se encuentra un fichero, ya sea para abrirlo o para guardarlo:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QFileDialog)  # editado

def boton_clicado(self):
    fichero, _ = QFileDialog.getOpenFileName(self, "Abrir archivo",
".")
    print(fichero)
```

También se puede usar en el modo para guardar un fichero:

```
fichero, _ = QFileDialog.getSaveFileName(self, "Guardar archivo", ".")
```

Este modo es muy útil porque si el fichero ya existe te avisa de que se va a sobreescribir.

Diálogos de entrada de datos QInputDialog

Pensados para pedir un dato concreto al usuario:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QInputDialog)  # editado

def boton_clicado(self):
    dialogo = QInputDialog.getText(self, "Título", "Texto")
    dialogo = QInputDialog.getInt(self, "Título", "Entero")
    dialogo = QInputDialog.getDouble(self, "Título", "Decimal")
    dialogo = QInputDialog.getItem(
        self, "Título",  "Colores", ["Rojo", "Azul", "Blanco",
"Verde"])
```

Este diálogo devuelve una tupla, primero el valor y luego si se ha confirmado el diálogo. Esto sirve para saber si se cancela la captura de datos, por eso una forma de tratar la información es en dos variables:

```
color, confirmado = QInputDialog.getItem(
    self, "Título",  "Colores", ["Rojo", "Azul", "Blanco", "Verde"])

if confirmado:
    print(color)
```

Diálogos de fuente QFontDialog y color QColorDialog

Estos tienen el objetivo de seleccionar fuentes del sistema y colores.

Veamos como abrir una fuente para utilizarla en un botón:

```
from PySide6.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QFontDialog)  # new

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

        boton = QPushButton("Mostrar diálogo")
        boton.clicked.connect(self.boton_clicado)
        self.setCentralWidget(boton)

        self.boton = boton

    def boton_clicado(self):
        confirmado, fuente = QFontDialog.getFont(self)
        if confirmado:
            # fuente es un objeto QFont
            self.boton.setFont(fuente)
```

Y ahora un color para usarlo de fondo:

```
def boton_clicado(self):
    color = QColorDialog.getColor()
    if color.isValid():
        # color es un objeto QColor, name() devuelve su código
hexadecimal
        self.boton.setStyleSheet(f"background-color: {color.name()}")
```

Con esto hemos cubierto casi todo sobre los diálogos, si queréis profundizar os dejo la documentación oficial.