La ventana principal `QMainWindow`

Cualquier objeto o “widget” se puede convertir en una ventana. Incluso podríamos crear una ventana que fuera sólo un botón. (Archivo a03)

```
import sys
from PyQt6.QtWidgets import QApplication, QPushButton

app = QApplication(sys.argv)

ventana = QPushButton("Haz click aquí")
ventana.show()

app.exec()
```

Esto está bien, y nos puede ser útil para una ventana de aviso o para la típica ventana de “Aceptar”-”Cancelar”. Para ventanas más importantes usaremos mejor `QMainWindow`. (Archivo b01)

Ventana principal `QMainWindow`

Es un widget prediseñado que proporciona muchas funciones de ventana estándar que se usan habitualmente en las aplicaciones, incluidas barras de herramientas, menús, una barra de estado, widgets acoplables y más. Agregaremos, de momento, una `QMainWindow` vacía a nuestra aplicación.

```
import sys
from PyQt6.QtWidgets import QApplication, QMainWindow

app = QApplication(sys.argv)

ventana = QMainWindow()
ventana.show()

# Iniciar el bucle infinito que "escucha" los eventos.
app.exec()
```

Ya podemos agregar algo de contenido.

La mejor manera de personalizar la ventana es crear una subclase de QMainWindow y luego incluir la configuración de la ventana en el constructor (el bloque `__init__`). Esto permite que el comportamiento de la ventana sea autónomo.

Podemos agregar nuestra propia subclase de `QMainWindow` La llamaremos MainWindow.

```
import sys
```

```
from PyQt6.QtCore import QSize, Qt
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton


# Subclass QMainWindow to customize your application's main window
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("My App")
        boton = QPushButton("Press Me!")

        # Set the central widget of the Window.
        self.setCentralWidget(boton)


app = QApplication(sys.argv)

ventana = MainWindow()
ventana.show()

app.exec()
```

Para esta demostración estamos usando un `QPushButton`.

Los widgets principales de Qt siempre se importan desde el espacio de nombres `QtWidgets`, al igual que las clases `QMainWindow` y `QApplication`.

Cuando creamos una subclase de una clase Qt, siempre tenemos que llamar a la función `super().__init__()` para permitir que Qt configure el objeto. Sólo con eso hemos dotado a nuestra ventana de toda la funcionalidad habitual que tienen las ventanas de una interfaz gráfica.

En nuestro bloque `__init__` (en el del hijo) primero usamos `.setWindowTitle()` para cambiar el título de nuestra ventana principal.

Luego agregamos nuestro primer widget, un `QPushButton`, en el medio de la ventana (se coloca en el centro porque usamos el método `.setCentralWidget`). Este es uno de los widgets básicos disponibles en Qt. Al crear el botón, puede pasar el texto que desea que muestre el botón.

Veremos cómo agregar widgets diferentes a las ventanas en el tutorial de diseños.

Ahora, ejecutando el archivo, veremos nuestra ventana nuevamente, pero esta vez con el widget `QPushButton` en el medio. Al presionar el botón no se hará nada, lo solucionaremos a continuación.

> Una configuración alternativa es crear el botón antes de crear la ventana principal y en el momento de crearla, recibir el botón y colocarlo. (Archivo b02)
>
> ```
> import sys
> ```

```
from PyQt6.QtCore import QSize, Qt
from PyQt6.QtWidgets import QApplication, QMainWindow, QPusl


# Subclase QMainWindow para personalizar la ventana principa
class MainWindow(QMainWindow):
    def __init__(self, boton):
        super().__init__()

        self.setWindowTitle("Mi aplicación")

        # Colocar el botón como "Widget central".
        self.setCentralWidget(boton)


app = QApplication(sys.argv)

boton = QPushButton('Haz clic aquí')
window = MainWindow(boton)
window.show()

app.exec()
```