Conexión de señales y slots

En Qt, la comunicación entre diferentes partes de una aplicación se gestiona mediante un mecanismo llamado **señales y slots**.

Las señales son, esencialmente, **notificaciones** que los widgets emiten cuando ocurre un evento específico.

Ejemplo: Un botón emite una señal cuando un usuario hace clic en él.

Estas señales pueden ser **conectadas** a funciones o métodos, conocidos como **slots**, que se ejecutarán cuando se emita la señal. Esta conexión se establece usando el método correspondiente.

La información sobre el evento que provocó la señal, como el estado del botón, también se puede pasar como **datos** a los slots.

Beneficios de Signals y Slots:

- **Desacoplar componentes:** Los widgets emisores de señales no necesitan saber qué slots están conectados, promoviendo la modularidad y la reutilización de código.
- **Conectar múltiples slots:** Se pueden conectar múltiples slots a una sola señal, permitiendo que diferentes partes de la aplicación respondan al mismo evento.
- **Conectar widgets directamente:** Los slots también pueden pertenecer a otros widgets, permitiendo la comunicación directa entre ellos.

Tipos de Signals:

- **Signals predefinidos:** Qt proporciona una amplia gama de signals integrados en sus widgets que veremos a continuación.
- **Signals personalizados:** Los desarrolladores pueden crear sus propias señales para eventos específicos de la aplicación, extendiendo la funcionalidad del sistema de señales y slots.

Signals integrados

QPushButton Signals

Primera señal

Aportamos sólo el código de la clase MainWindow puesto que el resto de la aplicación se mantiene igual (Archivo c01):

```
class MainWindow(QMainWindow):
        def __init__(self):
                super().__init__()
                self.setWindowTitle("Mi aplicación")

                # Código del botón
                boton = QPushButton("Haz clic aquí")
                # Señal de click y llamada al método
                boton.clicked.connect(self.the_button_was_clicked)
```

```
                self.setCentralWidget(button)

        # Método que se lanza con el click
        def boton_pulsado(self):
                print("¡Señal recibida!")
```

Señal que envía datos.

Si convertimos el botón en un interruptor, como los de la luz, que se puede dejar activado, podemos enviar el estado del botón en un parámetro (Archivo c02):

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Mi aplicación")

        # Código del botón
        boton = QPushButton('Haz clic aquí')
        # Convertimos el botón en un interruptor que podemos dejar pulsado
        boton.setCheckable(True)
        # Señal de click y llamada al método
        boton.clicked.connect(self.boton_pulsado)
        boton.clicked.connect(self.interruptor_activado)
        self.setCentralWidget(boton)

    # Método que se lanza con el click
    def boton_pulsado(self):
        print("¡Señal recibida!")

    def interruptor_activado(self, checked):
        print("¿Activado?", checked)
```

Guardado del estado

Podemos asignar tantas funciones slot como queramos a nuestro botón y pueden responder a tipos distintos de señales.

Podemos guardar el cambio hecho en el estado del interruptor en una variable (Archivo c03):

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.interruptor_activado = False
        self.setWindowTitle("Mi aplicación")

        # Código del botón
        boton = QPushButton('Haz clic aquí')
        # Convertimos el botón en un interruptor que podemos dejar pulsado
```

```
        boton.setCheckable(True)
        # Inicializamos el estado del botón. Podría estar activado por defe
cto
        boton.setChecked(self.interruptor_activado)
        # Señal de click y llamada al método
        boton.clicked.connect(self.boton_pulsado)
        boton.clicked.connect(self.activa_interruptor)
        self.setCentralWidget(boton)

    # Método que se lanza con el click
    def boton_pulsado(self):
        print("¡Señal recibida!")

    def activa_interruptor(self, checked):
        self.interruptor_activado = checked
        print("¿Activado?", self.interruptor_activado)
```

Acceso al estado del botón

También puedo guardar el botón como un atributo de la ventana principal, así puedo tener acceso al estado del botón a través de `self` (Archivo c04):

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.interruptor_activado = False
        self.setWindowTitle("Mi aplicación")

        # Código del botón
        boton = QPushButton('Haz clic aquí')
        # Convertimos el botón en un interruptor que podemos dejar pulsado
        boton.setCheckable(True)
        # Inicializamos el estado del botón. Podría estar activado por defe
cto
        boton.setChecked(self.interruptor_activado)
        # Señal de click y llamada al método
        boton.clicked.connect(self.boton_pulsado)
        boton.clicked.connect(self.activa_interruptor)
        self.setCentralWidget(boton)
        self.boton = boton

    # Método que se lanza con el click
    def boton_pulsado(self):
        print("¡Señal recibida!")

    def activa_interruptor(self, checked):
        self.interruptor_activado = checked
        print("¿Activado?", self.interruptor_activado)
```

```
        # Imprimo accediendo al estado del interruptor desde self
        print("¿Activado (self)?", self.boton.isChecked())
```

Modificación de la ventana desde la ejecución de un slot.

(Archivo d01) Eliminamos la capacidad de "interruptor" del botón y nos centramos en el slot lanzado por el click.

En este código lo usamos para modificar el texto del botón, bloquearlo, e incluso cambiar el título de la ventana a la que pertenece.

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.interruptor_activado = False
        self.setWindowTitle("Mi aplicación")

        # Código del botón
        self.button = QPushButton('Haz clic aquí')

        # Señal de click y llamada al método
        self.button.clicked.connect(self.boton_pulsado)

        self.setCentralWidget(self.button)

    # Método que se lanza con el click
    def boton_pulsado(self):
        # Cambiamos el texto del botón
        self.button.setText("Ya me has pulsado.")
        # Inhabilitamos el botón. Ya no se puede volver a usar.
        self.button.setEnabled(False)
        # También cambiamos el nombre de la ventana.
        self.setWindowTitle("Aplicación con botón desactivado")
```

La ventana principal también tiene una señal que se activa cuando algo hace que su título cambie llamada `windowTitleChanged` que a su vez podría lanzar otro slot.

Podemos modificar un Widget desde otro.

(Archivo d02) La mayoría de los widgets Qt tienen slots disponibles, a las que puedes conectar cualquier señal que emita el mismo tipo que acepta. La documentación del widget tiene los slots para cada widget enumeradas en "Slots públicos". Por ejemplo, se puede consultar en este enlace los slots de QLabel. En este caso hemos usado `setText` :

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Mi aplicación")

        # Creamos un objeto Label que sirve para MOSTRAR texto
```

```
        self.label = QLabel()

        # Creamos un objeto Label que sirve para INTRODUCIR texto
        self.input = QLineEdit()

        # El objeto input genera señal al cambiar su texto
        # Lo recibe el label y replica el texto introducido
        self.input.textChanged.connect(self.label.setText)

        layout = QVBoxLayout()
        layout.addWidget(self.input)
        layout.addWidget(self.label)

        container = QWidget()
        container.setLayout(layout)

        self.setCentralWidget(container)
```