

## INSTRUMENTACION VIRTUAL

### LABEL

El widget Label es una instancia de la clase QLabel y es usado para desplegar mensajes e imágenes. Debido a que el widget Label simplemente despliega resultados y no recibe ninguna entrada, los Label son usados simplemente para desplegar información en la pantalla.

### METODOS

Los siguientes son los métodos proporcionados por la clase QLabel:

- **setText()**. Este método despliega texto en el widget Label.
- **setPixmap()**. Este método despliega un pixmap, es una instancia de la clase QPixmap, para el widget Label.
- **setNum()**. Este método despliega un valor entero o double al widget Label.
- **clear()**. Este método borra el texto del widget Label.

El texto por default que muestra el widget Label es TextLabel. Para poder cambiar el texto del Label se puede usar el método setText() o cuando se agrega el widget al formulario se puede cambiar en la ventana Editor de Propiedades en la propiedad text.

### LINE EDIT

El widget Line Edit es el más usado para introducir datos de una sola línea. El widget Line Edit es una instancia de la clase QLineEdit. Además de poder capturar lo que se introduzca también se puede editar. Además de introducir datos también se puede deshacer, rehacer, corta y pegar.

### METODOS

Los siguientes métodos son proporcionados por la clase QLineEdit:

- **setEchoMode()**. Establece el modo eco del widget Line Edit. Determina como el contenido del widget Line Edit será desplegada. Las opciones disponibles son:
  - **Normal**: Este es el modo por default y muestra los caracteres de la misma forma como se van introduciendo.
  - **NoEcho**: Esto apaga el eco del Line Edit, lo que significa que no desplegara nada.
  - **Password**: Esta opción es usada para introducir contraseñas y en su lugar desplegará asteriscos.
  - **PasswordEchoOnEdit**: Esta opción desplegara el texto actual mientras se esté editando el campo password al finalizar desplegara asteriscos.
- **maxLength()**. Este método es usado para especificar el máximo número de caracteres que podrá ser introducido en el widget Line Edit.
- **setText()**. Este método es usado para asignar el texto al widget Line Edit.
- **text()**. Este método toma el texto introducido en el widget Line Edit.

- **clear()**. Este método limpia o borra por completo el contenido del widget Line Edit.
- **setReadOnly()**. Cuando se establece el valor true a este método, el widget Line Edit se pondrá en el modo de solo lectura (no se podrá editar). Lo único que se podrá hacer es copiar el contenido del Line Edit.
- **isReadyOnly()**. Este método retornara verdadero si el Line Edit está en modo de solo lectura en caso contrario retornara Falso.
- **setEnabled()**. Cuando se pasa un valor booleano falso el Line Edit quedará deshabilitado y el usuario no podrá introducir algún valor al Line Edit pero si podrá introducir valor a través del método setText(). Por default el método setEnable() tiene un valor verdadero permitiendo introducir algún valor al Line Edit.
- **setFocus()**. Este método posicionara el cursor en el Line Edit especificado.

#### PUSH BUTTON.

El push button es una instancia de la clase QPushButton. Cuando colocamos un texto en el push button podemos crear un atajo colocando un ampersand (&) a lado izquierdo de la letra que queramos usar como atajo. Por ejemplo, si tenemos el texto “Aceptar” podemos poner un ampersand (&) antes de la “A” al hacer click en el push button o presionando la combinación Alt + A en ambos casos se emite una señal del tipo clicked(). Los métodos para desplegar texto o un icono en el push button son los siguientes:

- **setText()**. Este método es usado para asignar texto al push button.
- **setIcon()**. Este método es usado para asignar un icono al push button.

Vamos a crear una aplicación sencilla para entender los tres elementos descritos anteriormente. El siguiente ejemplo es una calculadora grafica sencilla la interfaz tendrá dos Line Edit para introducir los valores, cuatro push buttons para las operaciones y un label para desplegar el resultado.

Vamos a abrir el Designer y realizamos los siguientes pasos:

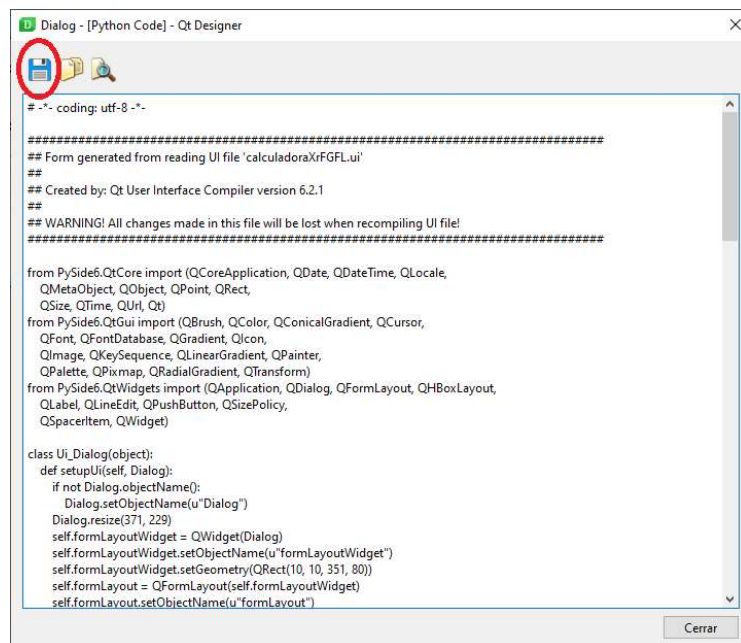
1. Creamos un dialogo sin botones
2. En la ventana que nos aparecerá arrastramos dos layouts un form layout y un horizontal layout.
3. Dentro del form layout agregamos un label y un line edit después debajo de esos dos widgets agregamos vertical spacer y debajo del espaciador arrastramos otro label y otro line edit.
4. En la propiedad text de los label modificamos su propiedad text y escribimos en el primer label “Numero 1:” y en el segundo label escribimos “Numero 2:”. Los line edit modificamos su propiedad objectName y escribimos “lineNum1” y “lineNum2” respectivamente.
5. Debajo del form layout colocamos el horizontal layout y agregamos cuatro push buttons uno a lado del otro.

6. En los cuatro push buttons vamos a modificar su propiedad text y escribimos “Sumar”, “Restar”, “Multiplicar” y “Dividir” respectivamente después modificamos su propiedad objectName y escribimos “btnSuma”, “btnResta”, “btnMul” y “btnDiv” respectivamente.
7. Por último, debajo del horizontal layout colaremos el label, en su propiedad text borramos su contenido y en su propiedad objectName escribimos “lblResult”.
8. Guardamos el archivo con el nombre calculadora.ui.

La interfaz en modo de diseño debe tener la siguiente forma:



9. Después de crear la interfaz convertimos el archivo “ui” en código Python. Primer vamos al menú Formulario/View Python Code...



Damos click en el icono de guardar (en la figura del disco) y guardamos el archivo Python en la ubicación deseada.

Lo siguiente es crear un nuevo proyecto en PyCharm y guardar el proyecto en la misma carpeta donde se tiene el archivo py y escribir el siguiente código.

```
import sys
from PySide6.QtWidgets import QApplication, QDialog
from calculadora import Ui_Dialog

class MiFormulario(QDialog, Ui_Dialog):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        self.btnSuma.clicked.connect(self.sumarNumeros)
        self.btnResta.clicked.connect(self.restarNumeros)
        self.btnMult.clicked.connect(self.multiplicarNumeros)
        self.btnDiv.clicked.connect(self.dividirNumeros)

    def sumarNumeros(self):

        a, b = self.tomarNumeros()
        suma = a + b
        self.lblResult.setText("La suma es: " + str(suma))

    def restarNumeros(self):
        a, b = self.tomarNumeros()

        resta = a - b
        self.lblResult.setText("La resta es: " + str(resta))

    def multiplicarNumeros(self):
        a, b = self.tomarNumeros()

        mult = a * b
        self.lblResult.setText("La multiplicacion es: " +
str(mult))

    def dividirNumeros(self):
        a, b = self.tomarNumeros()

        div = a / b
        cadena = "La division es: %.2f" % div
        self.lblResult.setText(cadena)
```

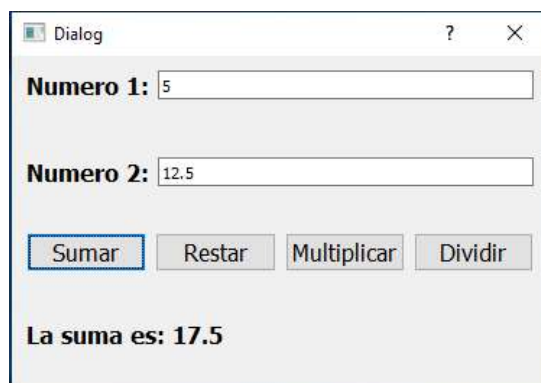
```
def tomarNumeros(self):  
    if len(self.lineNum1.text()) != 0:  
        a = eval(self.lineNum1.text())  
    else:  
        a = 0  
  
    if len(self.lineNum2.text()) != 0:  
        b = eval(self.lineNum2.text())  
    else:  
        b = 0  
  
    return a, b  
  
if __name__ == "__main__":  
    app = QApplication(sys.argv)  
    w = MiFormulario()  
    w.show()  
    sys.exit(app.exec())
```

Vamos a entender el código anterior ya que nos servirá como base para los siguientes programas.

1. Primero importamos los módulos necesarios en este caso la clase QtWidgets que es la clase base de todas las interfaces de usuario en PySide6.
2. Creamos una clase llamada MiFormulario que hereda de QDialog.
3. Creamos el constructor por default para QDialog y cargamos la interfaz gráfica.
4. El manejo de eventos en PySide6 es a través de un mecanismo de señales/slots. Una señal es un evento y un slot es un método que es ejecutado cuando ocurre una señal. Por ejemplo, cuando se da click en cualquiera de los cuatro push buttons un evento clicked será nuestra señal que a través del método connect() se enlazara con su slot correspondiente en este caso a los métodos "sumarNumeros", "restarNumeros", "multiplicarNumeros" o "dividirNumeros". Un lazo manejador de eventos se estará ejecutando continuamente hasta que un método exit() sea llamado o la ventana principal sea destruida.
5. Crearemos un objeto aplicación con el nombre app a través del método QApplication(). Cada aplicación PySide6 debe crear un objeto de aplicación sys.argv el cual contiene una lista de argumentos desde la línea de comandos, y se pasa al método mientras se crea el objeto de aplicación. El parámetro sys.argv ayuda en el paso y control de los atributos de inicio de un script.
6. Una instancia de la clase MiFormulario es creado con el nombre w.
7. El método show() desplegará la ventana (widget) en la pantalla.

8. Los métodos “sumarNumeros”, “restarNumeros”, “multiplicarNumeros” y “dividirNumeros” realizan las operaciones de sumar, restar, multiplicar y dividir los números que son capturados en los line edit. El método “tomarNumeros” es un método que nos servirá para saber si un numero valido es introducido en los line edit.
9. El método sys.exit() nos asegura una salida limpia, liberando los recursos de memoria al cerrar la ventana.

La salida del programa se muestra a continuación.



Dialog

Numero 1: 5

Numero 2: 12.5

Sumar Restar Multiplicar Dividir

La suma es: 17.5