Ximena Céspedes Quesada

Tutorial interfaz grafica para una calculadora sencilla con PYQT6 en Python.

En el siguiente tutorial, se creara una interfaz gráfica para una calculadora sencilla, mediante la biblioteca "PYQT6" en Python 3. Seguidamente, se explicara paso a paso como realizar el código:

1. **Importaciones:**

   En primer lugar, en la terminal de comandos de Windows, instala el paquete "pip install PyQt6". Una vez instalado, importa varios elementos de este módulo utilizando " PyQt6.QtWidgets", la cual nos ayudara a crear una interfaz utilizando Qt, este nos ayudara también a crear los botones y las cajas de entrada de la interfaz. También, es importante importar el módulo "sys" el cual nos va a ayudar con el manejo de argumentos de línea de comandos entre otras cosas. Dichas importaciones se observan en la figura 1.

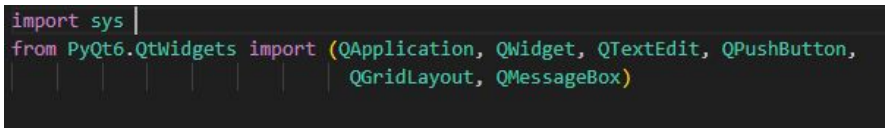

Figura 1: Importaciones.

2. **Creación de la clase:**

   Ahora bien, para representar a la calculadora, se define la clase "Calculator", la cual va a adquirir todas las propiedades de la clase "QWidget", la cual se utiliza para crear ventanas y widgets en la aplicación gráfica. Seguidamente se llama el método "$\_\_init\_\_$", el cual se conoce como el método constructor de la clase y este se va a utilizar para inicializar la calculadora y poder configurar la interfaz.

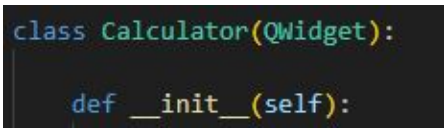

Figura 2: Definición de la clase.

3. **Inicialización de la clase:**

   En tercer lugar, se inicializara la clase, esto se hace mediante el método "$super().\_init\_()$", se utiliza esta para asegurarse de que se inicialice correctamente la parte de clase "QWidget", entonces así inicializa la parte de la clase base, configurando los atributos y comportamientos heredados y seguido de inicializar esta clase se llama a la función "inicializar" para realizar la inicialización de la subclase. Esto se puede observar en la figura 3

```
def __init__(self):
    super().__init__()
    self.inicializar()
```

Figura 3: Inicialización de la clase.

4. **Creación de la función “inicializar”:**

   Como se muestra en la figura 4, se configura el título de la ventana, cuáles son las dimensiones que esta va a tener y el color del fondo de la ventana. Seguidamente, para lograr almacenar los datos numéricos introducidos por el usuario, se inicializa una variable llamada “equation” como una cadena vacía. Seguidamente, se llamará a la función “$create\_widgets$”(se explicara más adelante) y por último se mostrara la ventana mediante “show”

```
def inicializar(self):
    self.setWindowTitle("Calculadora")
    self.setGeometry(0, 0, 250, 350)
    self.setStyleSheet("background-color: black;")
    self.equation = ""
    self.create_widgets()
    self.show()
```

Figura 4: Inicializar subclase.

5. **Creación de widgets:**

   Se define la función “$create\_widgets$”, la cual será la encargada de crear los elementos de la interfaz, tales como la caja de resultados y los botones. Esta función se llamará en la función inicializar, la cual se llama en el constructor de la clase para así poder crear los widgets al momento que se inicia la calculadora. Por lo tanto, el código que se debe tener hasta el momento se observa en la figura 5.

```
import sys
from PyQt6.QtWidgets import (QApplication, QWidget, QTextEdit, QPushButton,
                             QGridLayout, QMessageBox)


class Calculator(QWidget):

    def __init__(self):
        super().__init__()
        self.inicializar()

    def inicializar(self):
        self.setWindowTitle("Calculadora")
        self.setGeometry(0, 0, 250, 350)
        self.setStyleSheet("background-color: black;")
        self.equation = ""
        self.create_widgets()
        self.show()

    def create_widgets(self):
```

Figura 5: Primera parte del código.

Seguidamente, en la misma función definida anteriormente se realiza la creación de la caja de resultados. Para esto es necesario crear un campo de texto mediante “QTextEdit()”.seguidamente se deshabilita la edición de la pantalla y se configura el tamaño y el estilo de la misma, como color de fondo, color de fuente y tamaño de la misma, como se observa en la figura 6.

```
def create_widgets(self):
    self.pantalla = QTextEdit()
    self.pantalla.setDisabled(True)
    self.pantalla.setFixedHeight(40)
    self.pantalla.setStyleSheet("background-color: white; color: black; font-size: 16px;")
```

Figura 6: Creación de la caja de resultados.

En tercer lugar, en la misma función se crea una cuadricula principal, esto con el fin de organizar los botones y la pantalla en el espacio que deseamos, como se observa en la figura 7, se coloca la pantalla a la cuadricula en la fila 0 y que ocupe 4 columnas.

```
self.main_grid = QGridLayout()
self.main_grid.addWidget(self.pantalla, 0, 0, 1, 4)  # Pantalla en la fila 0, ocupando 4 columnas
```

Figura 7: Creación de la cuadricula y posicionamiento de la pantalla.

6. **Creación de los botones:**

Ahora bien, en la misma función anterior, se definen una lista de etiquetas para los botones. Por lo que, en esta lista están todos los números y símbolos que ocupara nuestra calculadora. Como se observa en la figura 8.

```
botones = [
    "C", "%", "√", "x",
    "7", "8", "9", "+",
    "4", "5", "6", "-",
    "1", "2", "3", "/",
    "0", ".", "="
]
```

Figura 8: Lista de botones

Seguidamente, se establecen dos variables, “row” y “col”, con valores iniciales de 0 y 1 respectivamente, estos valores iniciales se utilizan para poder rastrear la fila y la columna en la cuadricula de nuestra interfaz, esto mientras se van añadiendo los botones a sus respectivos espacios. Primero “‘row=1” debido a que los botones se van a empezar a colocar en la segunda fila, ya que la primera fila está ocupada por el campo de entrada y “col=0” significa que los botones se van a empezar a colocar desde la primera columna. Estas variables se van a ir actualizando a medida que el bucle vaya iterando y colocando los botones sobre la cuadricula, por lo que estas variables van a cambiar para lograr que los botones se coloquen en las posiciones adecuadas.

Ahora, como se observa en la figura 9, se crea un bucle el cual se va a encargar de configurar los botones de la calculadora, así como las funciones de estos y el estilo. Por lo tanto, mediante un bucle for se va a

iterar a través de cada elemento de la lista creada anteriormente, seguidamente dentro de este bucle se crea un botón, utilizando el comando “QPushButton”, el cual contiene la etiqueta “texto”..

```
for texto in botones:
    boton = QPushButton(texto)
```

Figura 9: Creación del bucle para configurar los botones

Seguido de esto se crean 4 diferentes condicionales, para los 4 diferentes tipos de botones que tenemos, ya que se tienen los botones que representan los números, otros que representan los operadores, los que tienen una función especial y por último el botón de resultado.

Como se observa en la figura 10, primero se crea el condicional el cual va a verificar si el “texto” del botón está en lista de botones especiales, luego para estos botones se configura el color de fondo, el tamaño de la fuente, el borde del botón, el color de la fuente, entre otros. Para estos se especificó que se quiere que tengan fondo gris con texto blanco. Seguido de esto se establece la acción que va a tener los botones, esto mediante “clicked.connect”, mediante este método se va a poder manejar los eventos de clic en los botones. Por lo tanto, cuando se hace clic en el botón “C”, se llama a la función “self.boton_limpiar”, y cuando se hace clic en “%” o “√”, se llama a funciones específicas “self.boton_especial_presionado’ con los valores correspondientes (“%” o “√”).

```
if texto in ["C", "%", "√"]:
    # Cambia el color de fondo y el color de texto para "C", "%", y "√"
    boton.setStyleSheet("background-color: #83838B; border:1px solid #000; font-size:20px; border-radius:8px; color: #fff;")
    if texto == "C":
        boton.clicked.connect(self.boton_limpiar)
    elif texto == "%":
        boton.clicked.connect(lambda: self.boton_especial_presionado("%"))
    elif texto == "√":
        boton.clicked.connect(lambda: self.boton_especial_presionado("√"))
```

Figura 10: Creación botones.

De forma similar, se realiza para los otros botones, como se observa en la figura 11, se tiene que para los botones que representan un operador se configuro el estilo de estos y se estableció un fondo naranja y texto blanco, y que se asoció a la función “self.boton_presionado”, seguidamente se creó para el botón de resultado, el cual tiene el mismo estilo que los botones anteriores y este se asoció a la función “self.calcular_resultado” y por ultimo para los botones numéricos, el cual se estilizo con un fondo gris oscuro y texto blanco, y se asoció a la función “self.boton_presionado”

```
if texto in ["x", "+", "-", ".", "/"]:
    boton.setStyleSheet("background-color: #b5520b; border:1px solid #000; font-size:20px; border-radius:8px; color: #fff;")
    boton.clicked.connect(self.boton_operador_presionado)

if texto in ["="]:
    boton.setStyleSheet("background-color: #b5520b; border:1px solid #000; font-size:20px; border-radius:8px; color: #fff;")
    boton.clicked.connect(self.calcular_resultado)

if texto in ["7", "8", "9", "4", "5", "6", "1", "2", "3", "0"]:
    boton.setStyleSheet("background-color: #2e2a27; border:1px solid #000; font-size:20px; border-radius:8px; color: #fff;")
    boton.clicked.connect(self.boton_numerico_presionado)
```

Figura 11: Creación botones.

Por último, para terminar con la configuración de los botones, mediante un condicional se logra colocar los botones en la cuadricula de la interfaz, y especificar el ancho de estos, para esto primero se verifica si el texto del botón es igual a 0, esto porque el botón 0 es más ancho que todos los demás, por lo que, si el botón es el 0, se agrega el botón a la cuadricula mediante “addWidget”, luego se le asigna la posición (row, col) en la cuadrícula y se le da un ancho de 2 columnas. Esto asegura que el botón ”0.ºcupe el espacio de dos columnas en la cuadrícula, mientras que otros botones ocupan solo una columna, después se utiliza un else en caso de que no sea 0, y se dice que si no es cero entonces que lo agregue a la interfaz pero con un ancho de una columna, por ultimo después de agregar los botones, se actualizan las variable “col”, entonces decimos que si el botón es 0 le sumamos 2 a la variable y de lo contrario solo se suma 1, esto nos va a ayudar para avanzar en las columnas del grid de la interfaz, seguido, se verifica si esta variables es mayor a 3, esto para asegurarnos que los botones contenga siempre 4 columnas, por lo que si col es mayor a 3 tiene que reiniciarse a 0 para volver a empezar en la siguiente fila. Esto lo podemos observar en la figura 12

```
        if texto == "0":
            self.main_grid.addWidget(boton, row, col,1,2)
        else:
            self.main_grid.addWidget(boton,row,col,1,1)

        col += 2 if texto == "0" else 1

        if col > 3:
            col = 0
            row += 1

    self.setLayout(self.main_grid)
```

Figura 12: Creación botones.

7. **Funcionamiento de los botones:**

   Para que los botones funcionen al momento que se llaman al hacerle clic, se definen varias funciones.

   Primeramente, se define la función “botón_especial_presionado”, la cual se va a llamar al momento de darle clic a un botón especial, y este va a mostrar el valor que le corresponde al botón que fue presionado en la caja de resultados. En la figura 13, se muestra el código empleado para la creación de la función.

```
def boton_especial_presionado(self, value):
    if value == "%":
        value = "/100"
    elif value == "√":
        value = "**0.5"

    self.equation += value
    self.pantalla.insertPlainText(value)
```

Figura 13: Creación de la función “botón_especial_presionado”

Luego se crea la función “botón_presionado” la cual será la encargada de hacer que se muestren los botones de operadores y números en pantalla, dicha función se muestra en la figura 14

```
def boton_presionado(self):
    sender = self.sender()
    texto = sender.text()
    self.equation += texto
    self.pantalla.insertPlainText(texto)
```

Figura 14: Creación de la función “botón_presionado”

Seguidamente, como se observa en la figura 15, se define la función “botón_limpiar”, la cual se llama en el momento que se hace clic en el botón “C” y esta borra los valores que se tienen en la caja de resultados.

```
def boton_limpiar(self):
    self.equation = ""
    self.pantalla.clear()
```

Figura 15: Creación de la función “botón_limpiar”

Por último, se define la función “calcular_resultado”, la cual nos va a ayudar a evaluar la expresión matemática al momento de hacer clic en el botón “=”, en esta misma función, se toma en cuenta el manejo de errores, previniendo que el usuario pueda ingresar una expresión errónea. Se puede observar en la figura 16.

```
def calcular_resultado(self):

    self.equation = self.equation.replace("x","*")

    try:
        self.result = eval(self.equation)
        self.equation = str(self.result)
        self.pantalla.clear()
        self.pantalla.insertPlainText(self.equation)
    except ZeroDivisionError:
        QMessageBox.critical(self, "Error", "No se puede dividir entre cero")
        self.equation = ""
        self.pantalla.clear()
    except Exception as e:
        QMessageBox.critical(self, "Error", "Operación inválida: " + str(e))
        self.equation = ""
        self.pantalla.clear()
```

Figura 16: Creación de la función “calculate”

8. **Creación del bucle principal de la aplicación:**

Como se observa en la figura 17, se crea una instancia de la aplicación con “ QApplication” llamada “app”, seguido de esto se crea una instancia de la clase “Calculator” y por ultimo utilizando “sys.exit(app.exec())” se ejecuta la aplicación, este comando inicia el bucle principal.

```python
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Calculator()
    sys.exit(app.exec())
```

Figura 17: Creación del bucle principal

9. **Resultado Final:**

   Por ultimo, en la figura 18, se observa el resultado final de la interfaz de la calculadora.

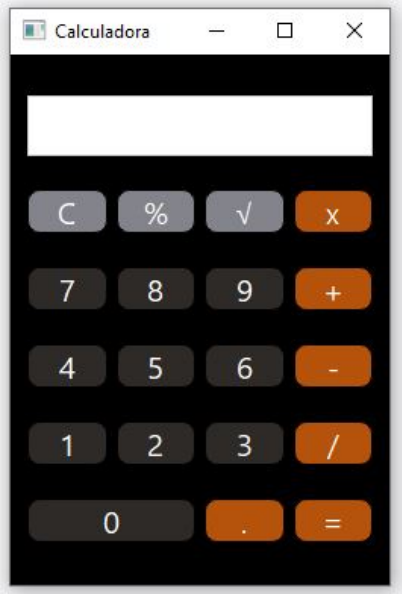

Figura 18: Resultado final