Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Programación Avanzada 2MV7

Practica 4

Autor:
Barrios Mendez Jose Alberto
Boleta: 2022640111

Profesor:
Cruz Mora Jose Luis

Ing. Mecatrónica

28 de marzo de 2024

Índice

1. Objetivo.

Crear aplicaciones con diferentes graficas de usuarii (GUI) utilizando controles basicos

2. Introduccion.

En esta presente practica, continuaremos con el aprendizaje de python, de forma mas especifica abordaremos las interfaces graficas, con ayuda de las librerias Pyqt6 y su diseñador de interfaces graficas, nos va a facilitar el crear la interfaz grafica, ya que de otra forma tendriamos que programar cada boton y que este corresponda dentro del lugar deseado. Con la ayuda de pyqt6,daremos solucion al programa que se plantea, el cual es diseñar una calculadora con las operaciones basicas. En este reporte abordaremos la forma en darle solucion con la libreria anteriormente mencionada

3. Desarrollo.

3.1. Creacion de la interfaz

Con ayuda del "Qt Disigner", el cual esta incluido dentro de las herramientas o "tools"de pyqt6, podemos abrir una GUI, la cual nos brinda una pantalla con varias opciones, para empezar a DISEÑAR nuestra calculadora, es importante recalcar que en este paso unicamente nos vamos a centrar en el diseño de la interfaz, la implementacion de las opciones que se muestran o la configuracion de los botones se realiza posteriormente.

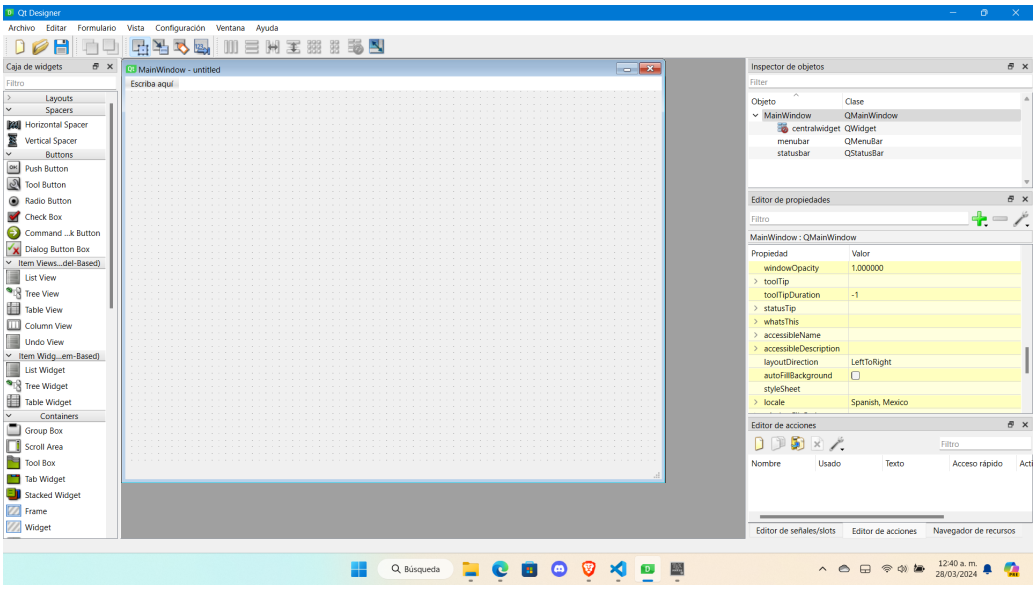

Figura 1: Interfaz de Qt Designer

Aqui vamos a empezar por el diseño de nuestra calculadora, para ello vamos utilizar 2 tipos de herramientas.

1. PushButtom: Esto nos va a servir como un boton de toda la vida, al presionarlo realizara alguna accion, en nuestro caso solo agregara numero o calculara las operaciones.

2. QLabel: Esto nos va a servir para poder mostrar los numeros que queremos cuando presionemos algun, numero o seleccionemos alguna operacion, es decir la funcionalidad es mostrar el texto en un recuadro

Una vez definios los tipos de herramientas a usar, continuaremos con el diseño de la calculadora para esto, yo elegi el siguiente diseño.

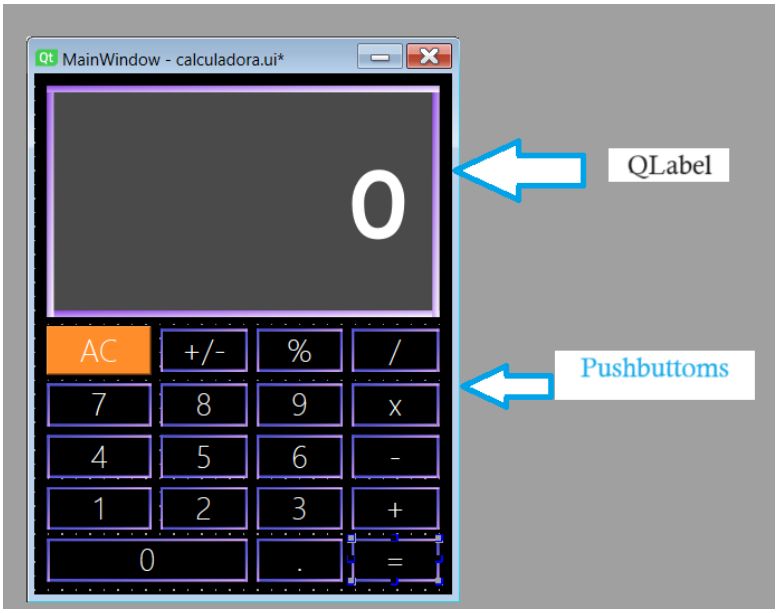

Figura 2: Diseño de la interfaz para la calculadora

3.2. Compilacion

Una vez realizado este paso, procedemos a guardar nuestro archivo, el cual se guarda con la extension .ui, para poder ejecutarlo como archivo python, es necesario realizar lo siguiente. Primero debemos usar una plantolla, la cual es vista en clase, con esta plantilla conseguiremos heredar el archivo .ui, todas las caracteristicas y atributos, pero con la diferencia que ahora le podremos agregar el backend, necesario para lograr que nuestra calculadora funcione de manera correcta. Antes de empezar a programar las funciones, es necesario crear un segundo archivo con extension python, esto es por lo siguiente.
Cuando nosotros obtenemos el primer archivo .py, este contiene toda la interfaz, es decir, es como si hubieramos diseñado la interfaz en este archivo, por lo cual es un archivo poco practico para implementar todo el backend, de tal forma que los archivos nos quedaran de la siguiente forma.

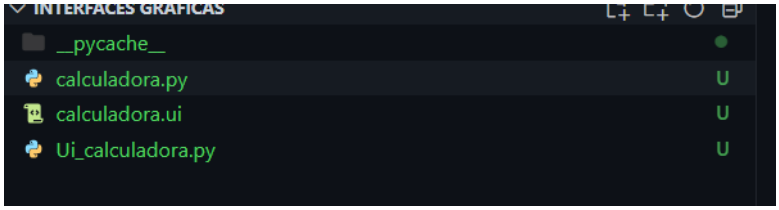

Figura 3: Archivos

3.3. Implementacion de cada funcion

Para esta parte, viene lo realmente complejo, ya que aqui es donde va a ocurrir la magia, en esta parte debemos definir que es lo que va a realizar cada boton, para esto, se implemento de la siguiente forma. Dentro de la clase principal "MainWindow"vamos a declarar el constructor, el cual contenera lo siguiente.

```
class MainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self, *parent, **flags) -> None:
        super().__init__(*parent, **flags)
        self.setupUi(self)

        self.btn0.clicked.connect(self.setNumero)
        self.btn1.clicked.connect(self.setNumero)
        self.btn2.clicked.connect(self.setNumero)
        self.btn3.clicked.connect(self.setNumero)
        self.btn4.clicked.connect(self.setNumero)
        self.btn5.clicked.connect(self.setNumero)
        self.btn6.clicked.connect(self.setNumero)
        self.btn7.clicked.connect(self.setNumero)
        self.btn8.clicked.connect(self.setNumero)
        self.btn9.clicked.connect(self.setNumero)
        self.btnMas.clicked.connect(lambda: self.agregarOperacion("+"))
        self.btnMenos.clicked.connect(lambda: self.agregarOperacion("-"))
        self.btnPor.clicked.connect(lambda: self.agregarOperacion("*"))
        self.btnDivi.clicked.connect(lambda: self.agregarOperacion("/"))
        self.btnIgual.clicked.connect(self.calcular)

        self.btnPunto.clicked.connect(self.setPunto)

        self.btnAC.clicked.connect(self.setCero)
```

Figura 4: Constructor de la clase principal

Lo que hace es conectar las señales (eventos) emitidas por los botones de la interfaz gráfica con las funciones correspondientes que manejan las acciones del usuario.
'self.btn0.clicked.connect(self.setNumero)' Conecta la señal çlicked"del botón "btn0çon la función "setNumero", lo que significa que cuando se haga clic en el botón "btn0", se llamará a la función "setNumero". Lo mismo ocurre para los botones btn1 a btn9, que están conectados a la función setNumero. Esto significa que cuando cualquiera de estos botones numéricos es presionado, se agrega el número correspondiente a la pantalla de la calculadora.
self.btnMas.clicked.connect(lambda: self.agregarOperacion(-")): Conecta la señal clicked del botón btnMas con una función lambda que llama a agregarOperacion con el argumento +. Esto significa que cuando se haga clic en el botón btnMas, se llamará a agregarOperacion con el operador +. Lo mismo ocurre para los botones de operación btnMenos, btnPor y btnDivi, que están conectados a funciones lambda que llaman a agregarOperacion con los operadores -, * y /. self.btnIgual.clicked.connect(self.calcular): Conecta la señal clicked del botón btnIgual con la función calcular

```python
def keyPressEvent(self, key: QKeyEvent) -> None:
    super().keyPressEvent(key)
    if key.key() >= QtCore.Qt.Key.Key_0 and key.key() <= QtCore.Qt.Key.Key_9:
        self.setNumber(key.key() - QtCore.Qt.Key.Key_0)
    elif key.key() == QtCore.Qt.Key.Key_Period:
        self.setPunto()
    elif key.key() == QtCore.Qt.Key.Key_Plus:
        self.agregarOperacion("+")
    elif key.key() == QtCore.Qt.Key.Key_Minus:
        self.agregarOperacion("-")
    elif key.key() == QtCore.Qt.Key.Key_Asterisk:
        self.agregarOperacion("*")
    elif key.key() == QtCore.Qt.Key.Key_Slash:
        self.agregarOperacion("/")
    elif key.key() == QtCore.Qt.Key.Key_Return or key.key() == QtCore.Qt.Key.Key_Enter:
        self.calcular(self.etiqueta.text())
```

Figura 5: Metodo para manejar los eventos del teclado

Este método keyPressEvent es responsable de manejar los eventos de teclado en la ventana de la calculadora

El metodo verifica que tecla es la que fue presionada y realiza la accion correspondiente:

- Si la tecla presionada es un número (0 a 9), llama a la función self.setNumber() con el número correspondiente.
- Si la tecla presionada es el punto (.), llama a la función self.setPunto() para agregar un punto decimal.
- Si la tecla presionada es la tecla de suma (+), resta (-), multiplicación (*) o división (/), llama a la función self.agregarOperacion() con el operador correspondiente.
- Si la tecla presionada es la tecla de enter , llama a la función self.calcular() con la expresión actual en la etiqueta de la calculadora

Ahora pasemos con los siguiente metodos

```python
def setCero(self):
    self.etiqueta.setText("0")

def setPunto(self):
    cadena = self.etiqueta.text()
    if cadena.find(".") == -1:
        cadena = cadena + "."
        self.etiqueta.setText(cadena)
```

Figura 6: Metodos para poner 0 y para poner punto

Estos metodos son triviales ya que hacen lo siguiente:

- SetCero: Unicamente borra los datos ingresados, o visto de otra forma, vuelve al estado inicial de la calculadora.
- SetPunto: metodo para agregar el punto decimal a un numero ingresado, verifica que no se agregue mas de uno, con el fin de que el codigo funcione correctamente

```python
def setNumero(self):
        cadena = self.etiqueta.text() + self.sender().text()
        if cadena.isdigit() or (cadena.startswith("-") and cadena[1:].isdigit()):
            flotante = float(cadena)
            if flotante.is_integer():
                entero = int(flotante)
                cadena = str(entero)
            else:
                cadena = str(flotante)
        self.etiqueta.setText(cadena)
```

Figura 7: Metodo para agregar digitos al QLabel

Este método, llamado setNumero, esta diseñado para manejar el evento de click, en cualquier boton de la interfaz

- cadena = self.etiqueta.text() + self.sender().text() obtiene el texto actual de la etiqueta de la calculadora y lo concatena Luego, se verifica si la cadena resultante después de la concatenación cumple con alguna de las siguientes condiciones:
- Es una cadena de dígitos (números enteros).
- Comienza con un signo menos (-) y el resto de la cadena son dígitos (números enteros).
- Contiene un punto (.) y el resto de la cadena son dígitos o puntos (números en formato de punto flotante). Si alguna de estas condiciones se cumple, se convierte la cadena a un número en formato de punto flotante (float(cadena)). Luego se verifica si el número es un enter (flotante.isinteger()). Si es un entero, se convierte a entero y se convierte nuevamente a cadena (str(entero)). Si no es un entero, se deja como está.
- Finalmente, se establece el texto de la etiqueta de la calculadora con la cadena resultante

Ahora vamos a revisar el metodo para calcular las operaciones.

```python
81  def calcular(self, operacion):
82
83      operacion = self.etiqueta.text()
84      if '+' in operacion:
85          operandos = operacion.split('+')
86          resultado = float(operandos[0])
87          for op in operandos[1:]:
88              resultado += float(op)
89      elif '-' in operacion:
90          operandos = operacion.split('-')
91          resultado = float(operandos[0])
92          for op in operandos[1:]:
93              resultado -= float(op)
```

Figura 8: Metodo para calcular las operaciones

Este método se encarga de realizar la evaluación de la expresión matemática presente en la etiqueta de la calculadora y mostrar el resultado en la misma etiqueta.

- **Pasos:**

1. Se obtiene la expresión matemática actualmente mostrada en la etiqueta de la calculadora mediante la asignación operacion = self.etiqueta.text().
2. Se verifica qué operador está presente en la expresión (+, −, ∗, /) utilizando condicionales `if`, `elif` y `else`.
3. Dependiendo del operador encontrado, se divide la expresión en una lista de operandos mediante el método `split()`. Por ejemplo, si la expresión es "5+3", se dividirá en ["5","3"].
4. Se inicializa la variable `resultado` con el primer operando convertido a un número de punto flotante mediante la expresión `float(operandos[0])`.
5. Se itera sobre los operandos restantes y se realiza la operación correspondiente con cada uno de ellos.
6. Si la operación implica una división y el divisor es cero, se establece el resultado como ."Error". De lo contrario, se realiza la división normalmente.
7. Finalmente, se establece el resultado calculado como el texto de la etiqueta de la calculadora mediante la expresión `self.etiqueta.setText(str(resultado))`.

4. Resultados.

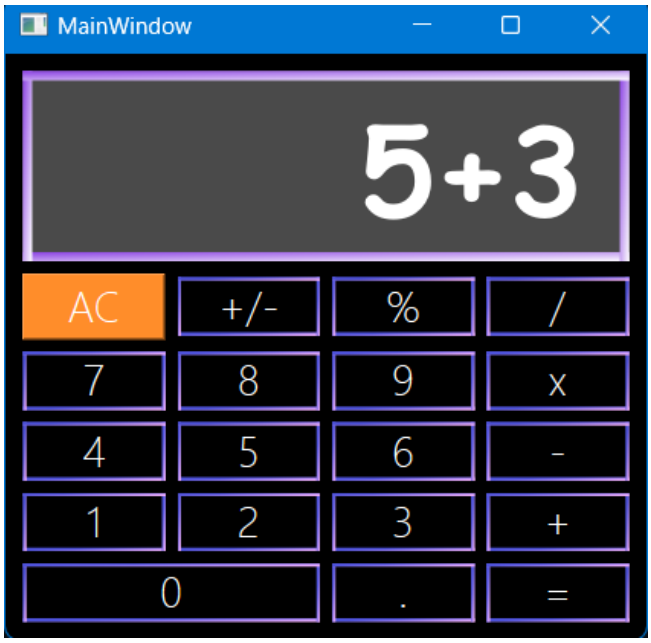

Figura 9

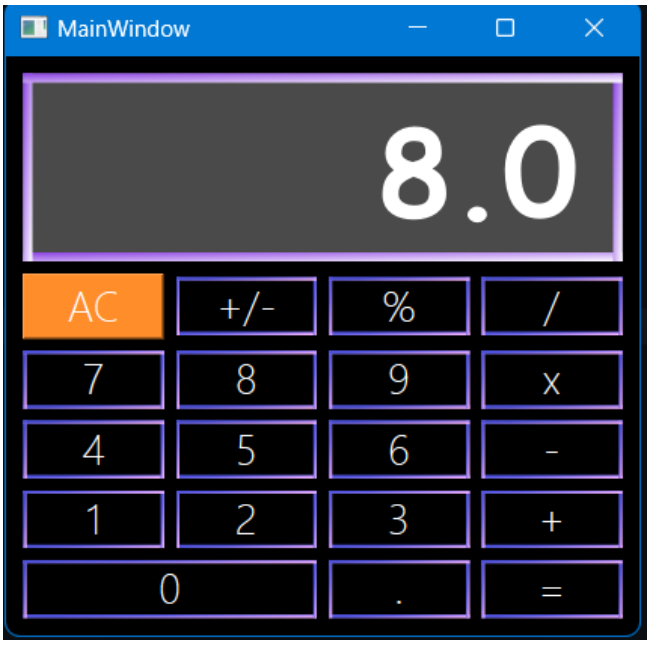

Figura 10

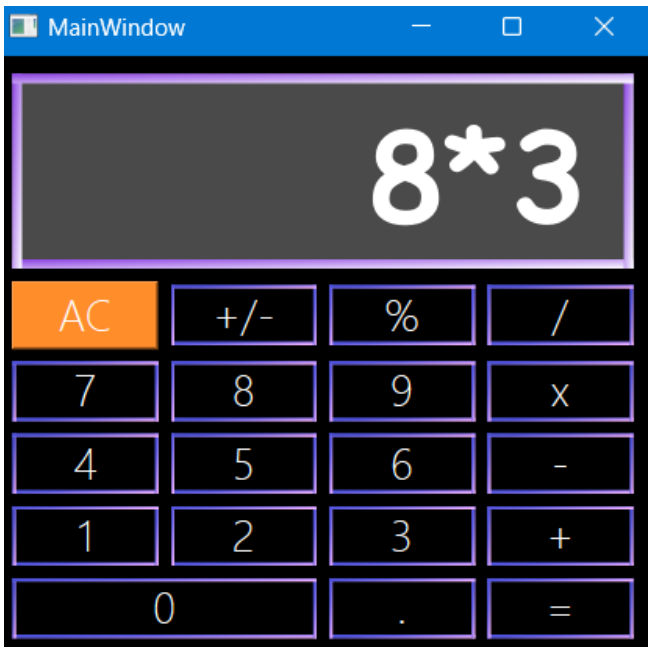

Figura 11

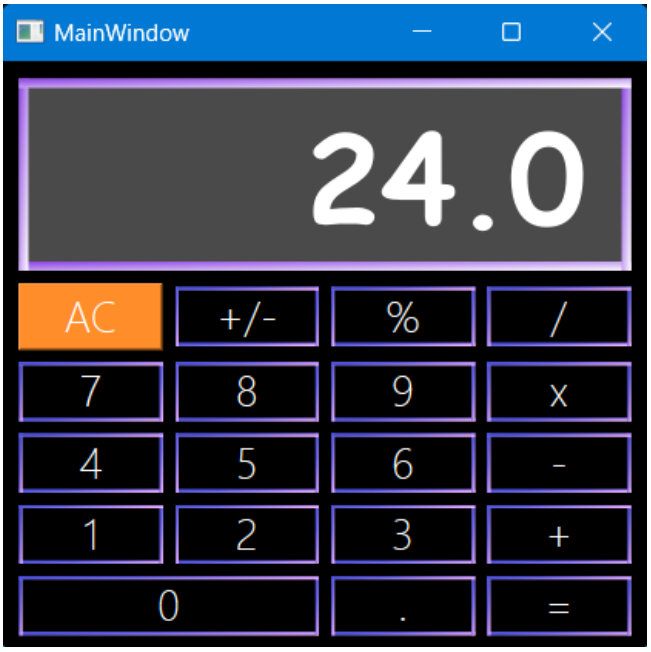

Figura 12

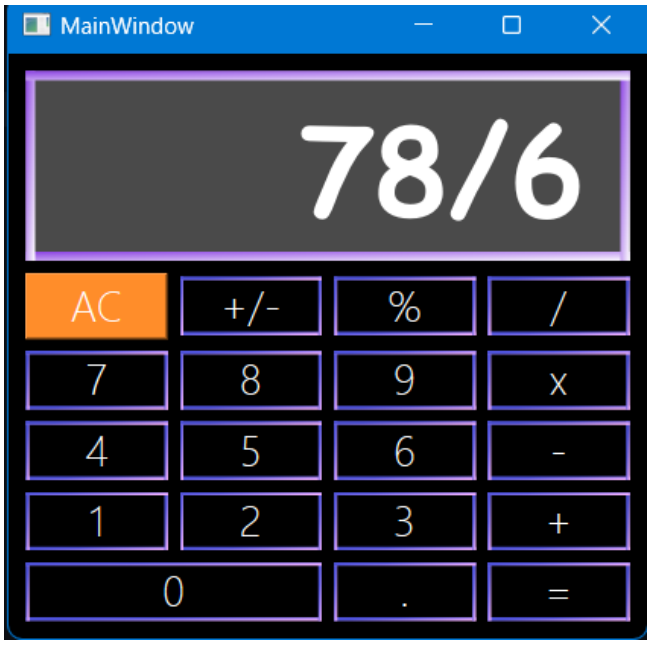

Figura 13

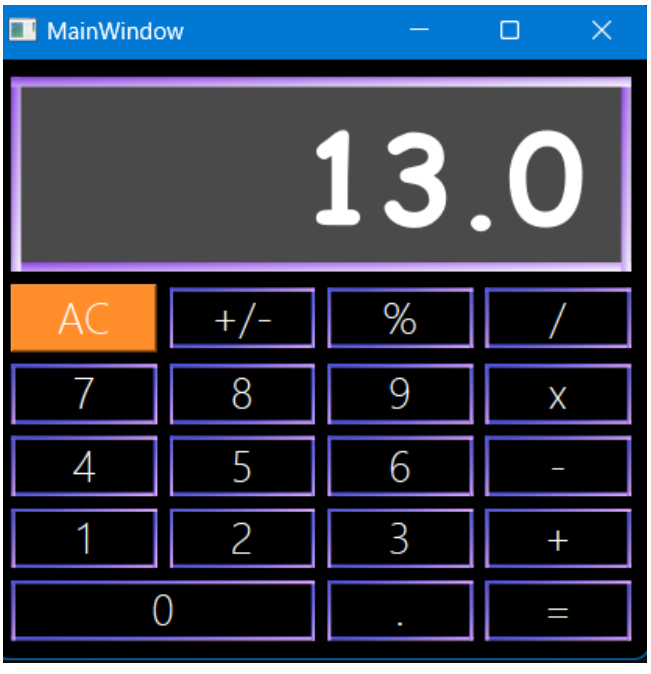

Figura 14

5. Conclusiones.

Esta practica fue un gran reto para mi, considerando que debemos tener claro los conceptos de programacion orientada a objetos, cuyos ejercios estuvimos trabajando en practicas pasadas, donde estas practicas me sirviero para poder comprender la implementacion de los metodos para poder lograr el funcionamiento de la calculadora,cabe mencionar que la idea es similar a las interfaces de matlab, las cuales tengo un poco mas de experiencia manejando, pero no dejando de lado la importancia de este lenjuagje para futuros poyectos