

# **Universidad Autónoma de Tamaulipas**

## **Facultad de Ingeniería Tampico**



**ASIGNATURA**  
**Programación de Interfaces y Puertos**  
**6to. Semestre – Grupo “G”**  
**2025 -1**

**TRABAJO**  
**Programas en Clase y Ejercicios**

**UNIDAD**  
**1 - MODELOS DE INTERACCIÓN**  
**COMPUTACIONAL**

**Docente: Dr. García Ruiz Alejandro H.**

<b>Integrante del Equipo</b>	<b>Nivel de Participación</b>
Vega Ruiz Adela	<b>33.33%</b>
Muñoz Perales Luis Gonzalo	<b>33.33%</b>
Hernández Juárez José Ángel	<b>33.33%</b>
<b>Total:</b>	<b>100%</b>

## Índice

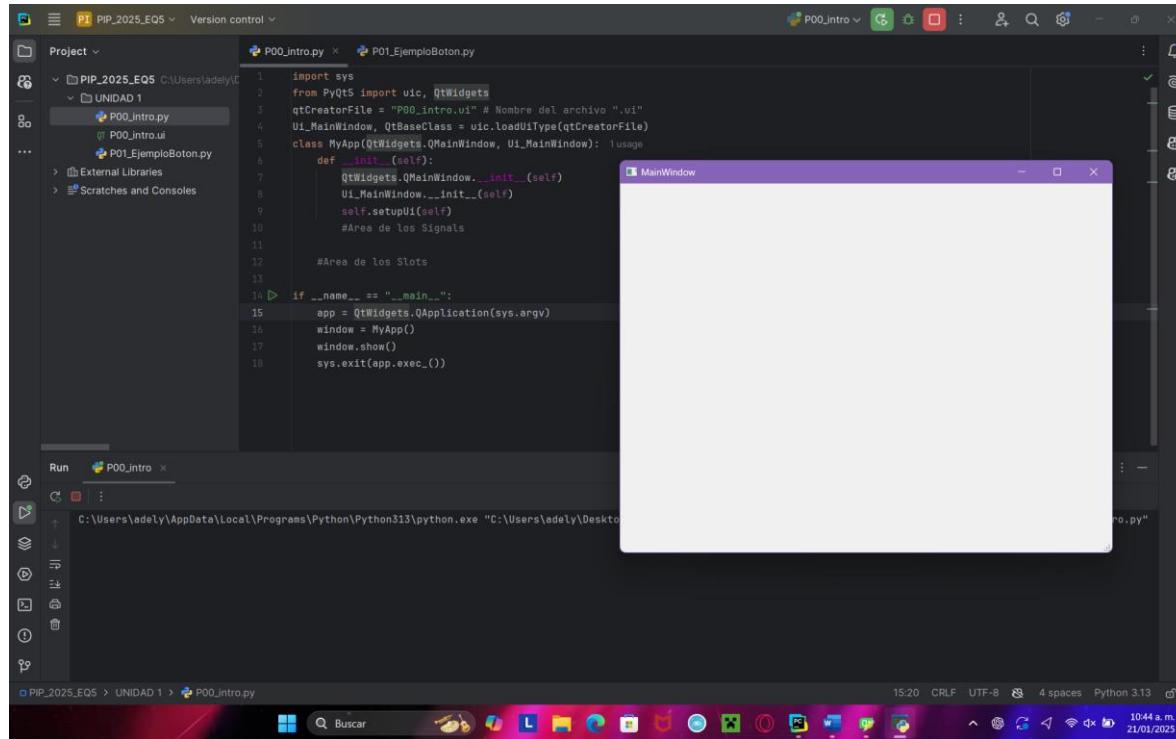
<i>Repository(s)</i> .....	2
<i>C1. P00_Intro</i> .....	2
<i>C2. P01_EjemploBoton</i> .....	3
<i>C3. P02_EjemploMensajeEmergenteBasico</i> .....	6
<i>C4. P03_EjemploLineEdit</i> .....	8
<i>C5. P04_SumaDosNumeros</i> .....	11
<i>C6. P05_SumaDosNumeros_V2</i> .....	14
<i>C7. P06_PromedioNumeros</i> .....	16
<i>C8. P07_PromedioNumeros-Load</i> .....	21
<i>C9. P08_PromedioNumeros-Load_V2</i> .....	33
<i>C10. E01_IMC</i> .....	41
<i>C11. E02_MetrosPies</i> .....	47
<i>C12. E03_PuntoMedio_2</i> .....	53
<i>C13. E04_AreaTriangulo</i> .....	57
<i>C14. E05_AreaRectangulo</i> .....	61
<i>C15. E06_IVA</i> .....	65
<i>C16. E07_Promedio_5</i> .....	70
<i>C17. E08_Velocidad</i> .....	78
<i>C18. E09_Voltaje_Ley_OHM</i> .....	84
<i>C19. E10_PromedioNumeros-File_check</i> .....	90
<i>C20. E11_PromedioNumeros-Union de proms</i> .....	97
<i>C21. E12_PromedioNumeros-UnaCarga</i> .....	99



## Repository(s)

Actividad	Repository
Trabajos en clase (C1.P00-C9.P08)	<a href="https://github.com/Adely201810/PIP_2025_EQ5.git">https://github.com/Adely201810/PIP_2025_EQ5.git</a>
Ejercicios (C10.E01-C21.E12)	<a href="https://GitHub.com/perales14/PIP_2025_1_eq5">https://GitHub.com/perales14/PIP_2025_1_eq5</a>

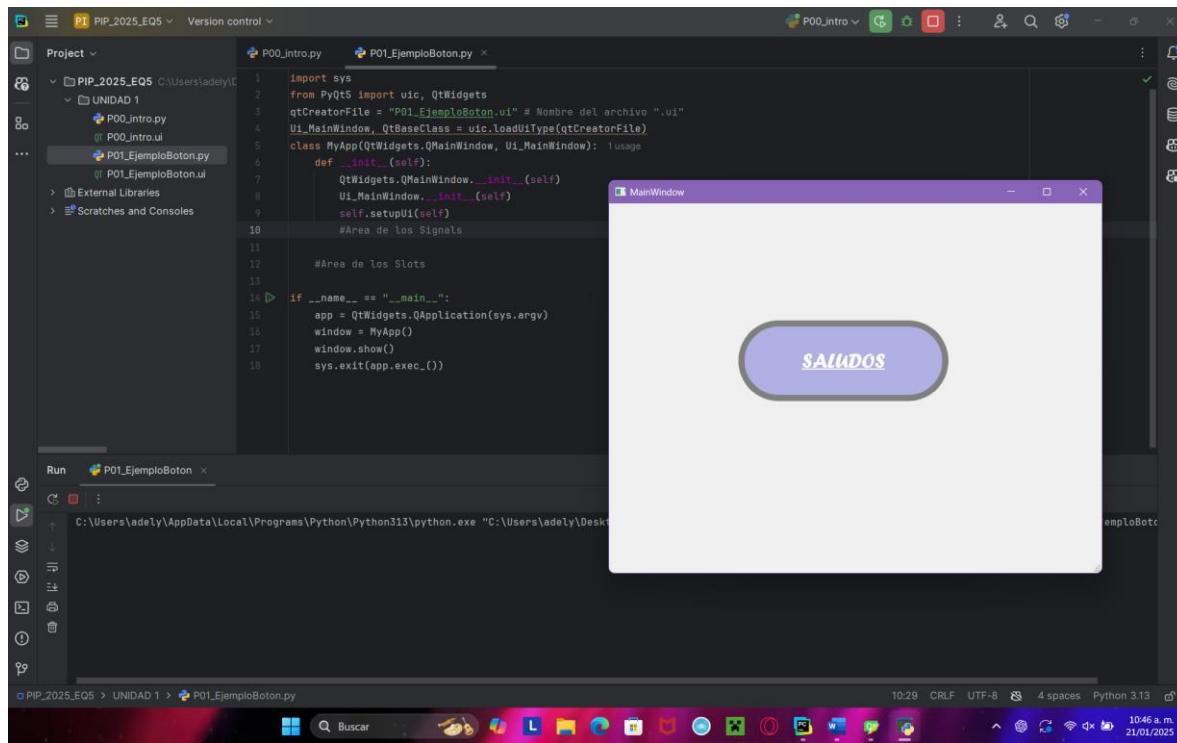
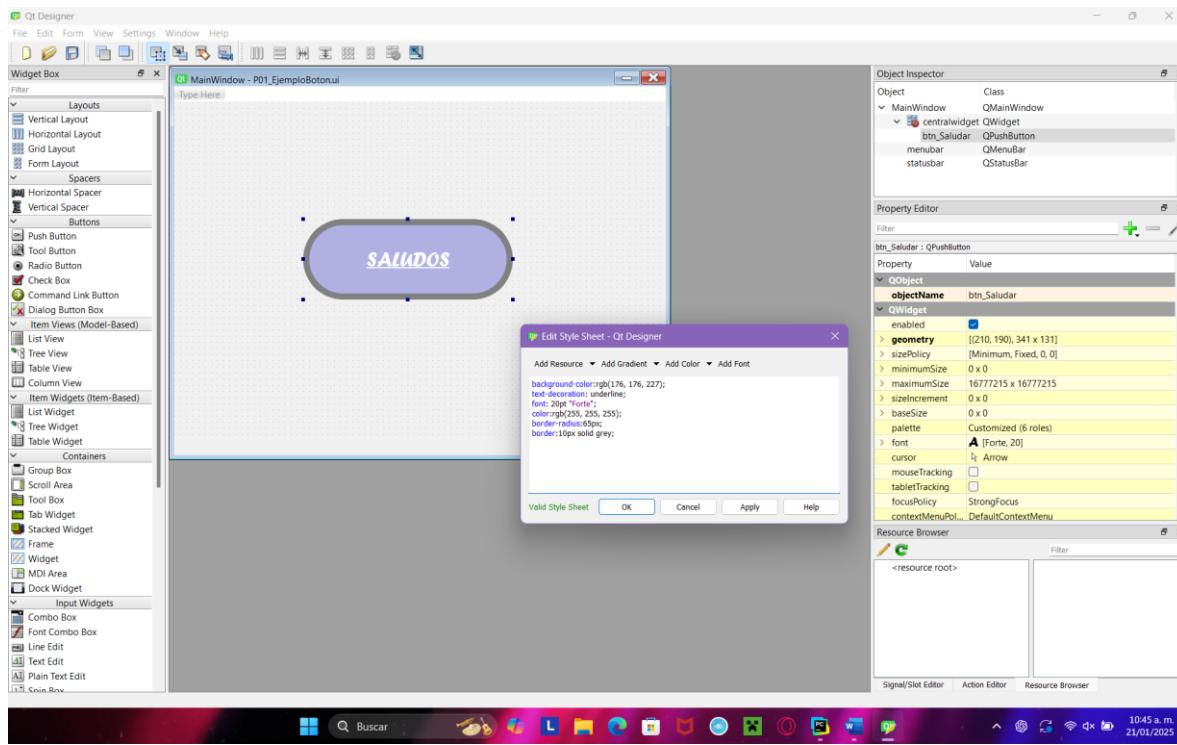
## C1. P00\_Intro



```
import sys
from PyQt5 import uic, QtWidgets
qtCreatorFile = "P00_intro.ui" # Nombre del archivo ".ui"
Ui_MainWindow, tBaseClass = uic.loadUiType(qtCreatorFile)
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        #Área de los Signals
        #Área de los Slots
    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```



## C2. P01\_EjemploBoton





The screenshot shows the PyCharm IDE interface. In the top navigation bar, it says "PIP\_2025\_EQ5" and "Version control". The project tree on the left shows "PIP\_2025\_EQ5" and "UNIDAD 1" which contains "P00\_intro.py", "P00\_intro.ui", and "P01\_EjemploBoton.py". The "Run" tab is selected, showing the command "C:\Users\adely\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\adely\Desktop\6° semestre\Programacion de Interfaces y Puertos\PIP\_2025\_EQ5\UNIDAD 1\P01\_EjemploBoton.py"". The main editor window displays Python code for a PyQt5 application. The code includes imports for sys and QtWidgets, a UI file named "P01\_EjemploBoton.ui", and a class "MyApp" that loads the UI and sets up signals. A specific error is highlighted at line 19: "self.btn\_saludar.clicked.connect(self.Saludar)". The error message in the terminal below the editor states: "AttributeError: 'MyApp' object has no attribute 'btn\_saludar'. Did you mean: 'btn\_Saludar'?". The status bar at the bottom right shows the time as 10:47 a.m. and the date as 21/01/2025.

The screenshot shows the PyCharm IDE interface again, but this time the application is running. The terminal output shows the printed message "Hola! es un gusto, ¿Comó estás?". The main window titled "MainWindow" is displayed, featuring a single button with the text "SALUDOS" in white on a purple background. The status bar at the bottom right shows the time as 10:48 a.m. and the date as 21/01/2025.



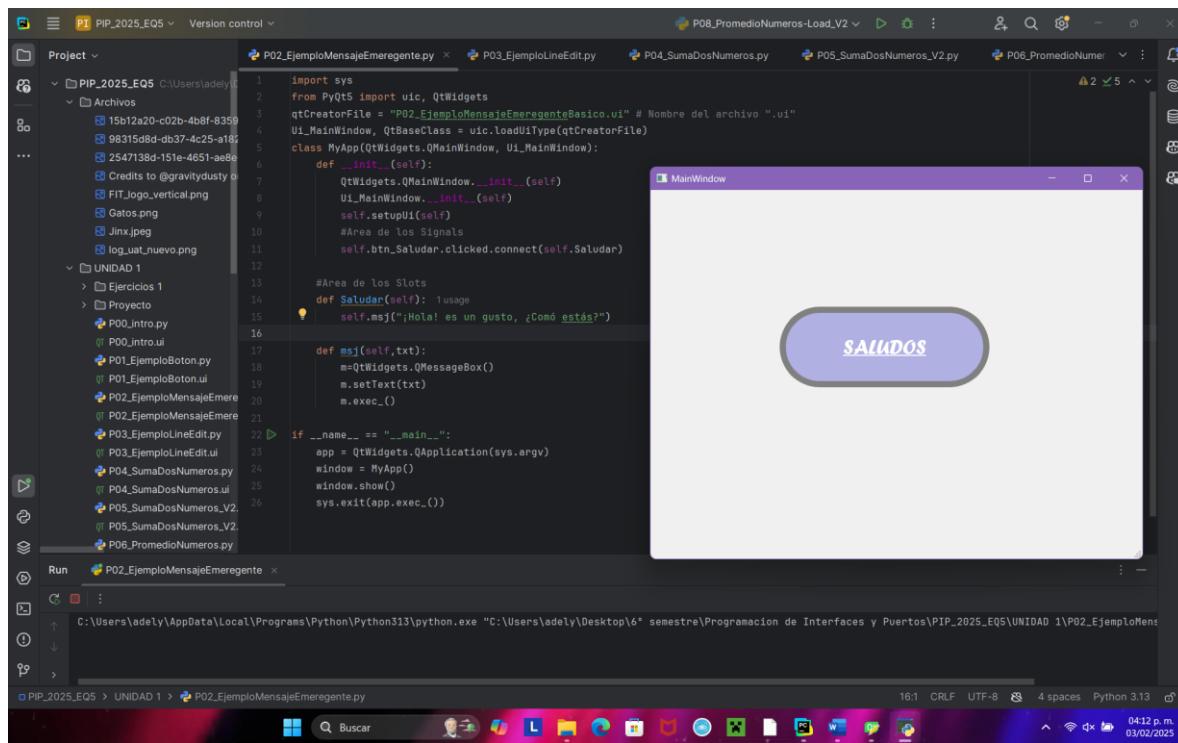
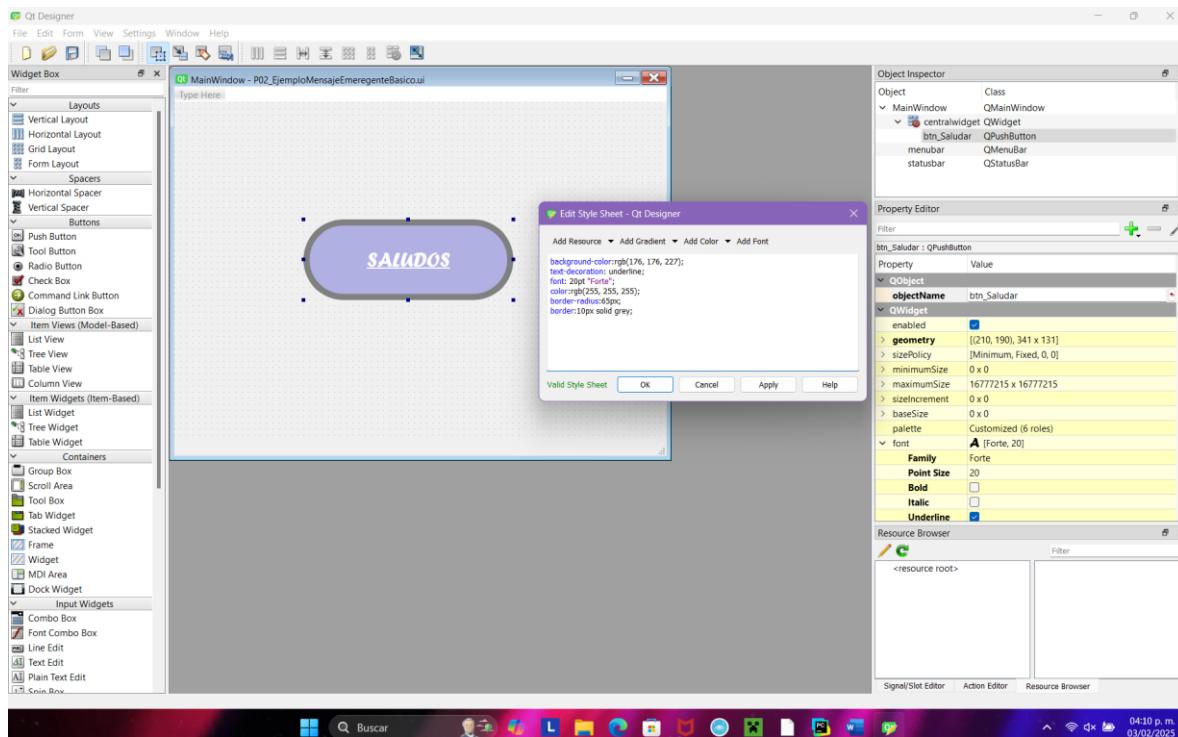
The screenshot shows the PyCharm IDE interface. On the left, the project structure for 'PIP\_2025\_EQ5' is visible, containing files like 'P00\_intro.py', 'P00\_intro.ui', and 'P01\_EjemploBoton.py'. The main editor window displays the code for 'P01\_EjemploBoton.py':

```
1 import sys
2 from PyQt5 import uic, QtWidgets
3 qtCreatorFile = "P01_EjemploBoton.ui" # Nombre del archivo ".ui"
4 ui_MainWindow, qtBaseClass = uic.loadUiType(qtCreatorFile)
5 class MyApp(QtWidgets.QMainWindow, ui_MainWindow):
6     def __init__(self):
7         QtWidgets.QMainWindow.__init__(self)
8         self.setupUi(self)
9         #Área de los Signals
10        self.btn_Saludar.clicked.connect(self.Saludar)
11
12        #Área de los Slots
13    def Saludar(self):
14        print("¡Hola! es un gusto, ¿Comó estás?")
15
16    if __name__ == "__main__":
17        app = QtWidgets.QApplication(sys.argv)
18        window = MyApp()
19        window.show()
20        sys.exit(app.exec_())
```

The bottom run tab shows the output: 'C:\Users\adely\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\adely\Desktop\6. Python\PIP\_2025\_EQ5\UNIDAD 1\P01\_EjemploBoton.py"' followed by several lines of text starting with '¡Hola! es un gusto, ¿Comó estás?'. To the right, a screenshot of the application window titled 'MainWindow' is shown, featuring a purple rounded rectangle button with the text 'SALUDOS'.



### C3. P02\_EjemploMensajeEmergenteBasico



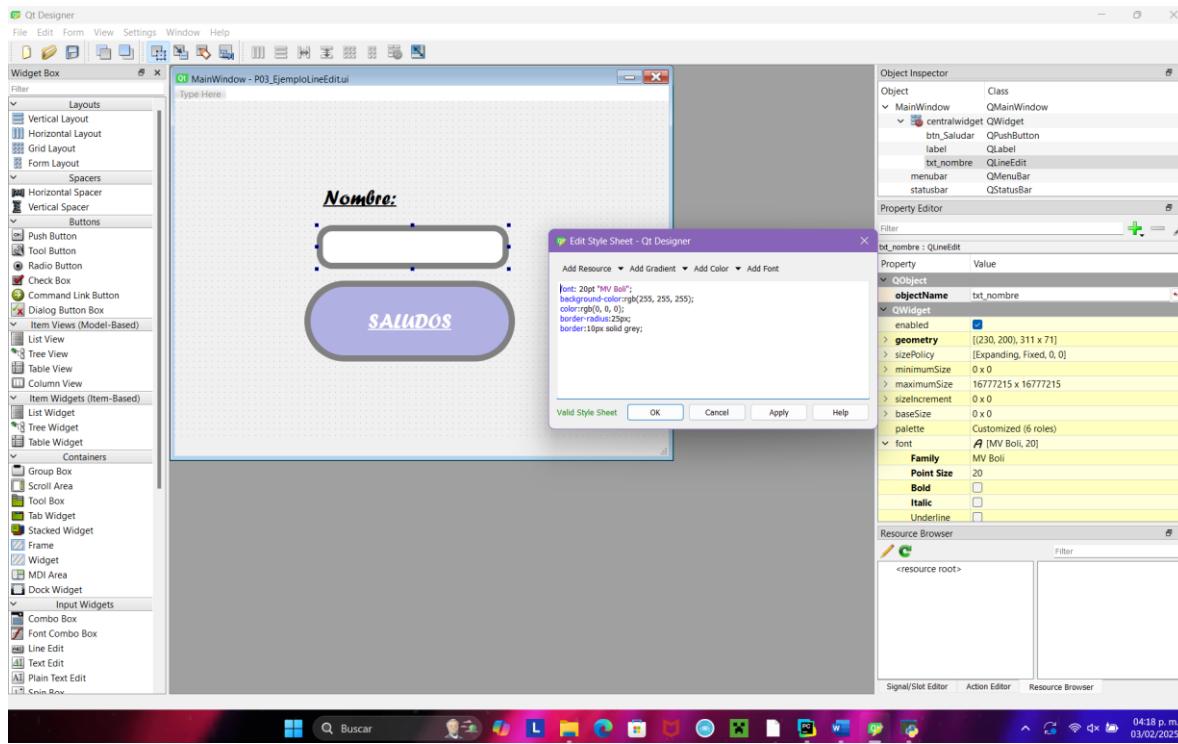
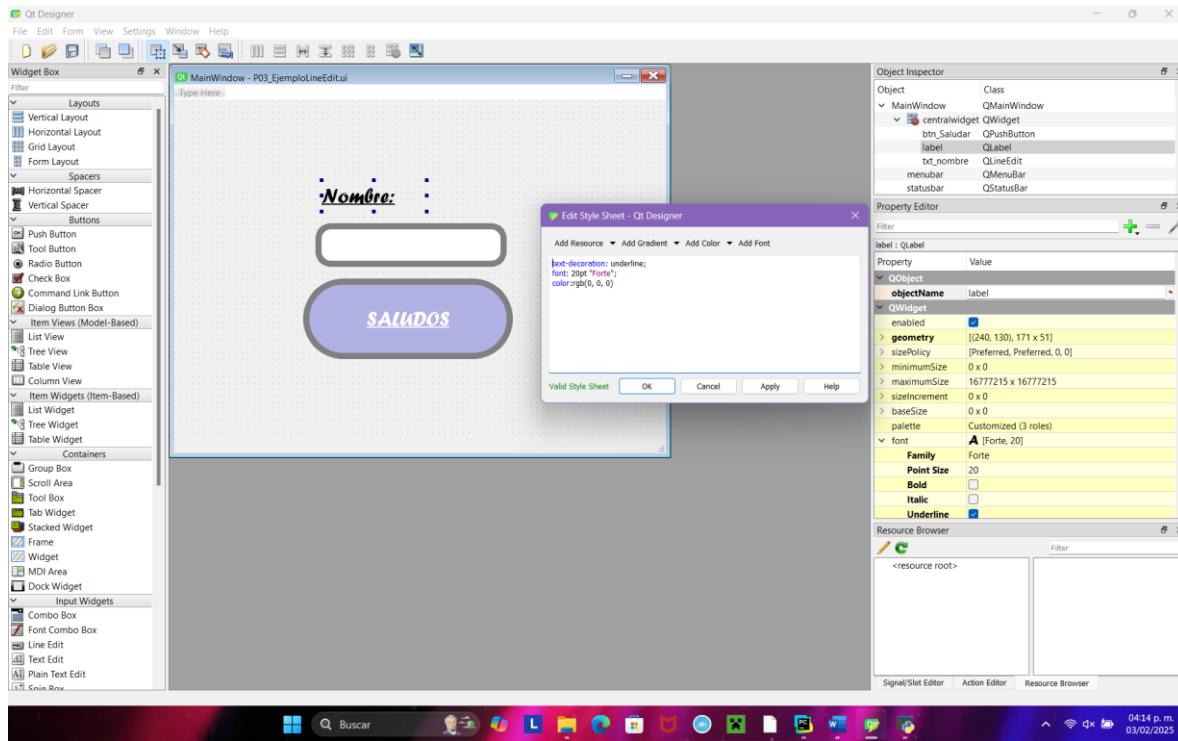


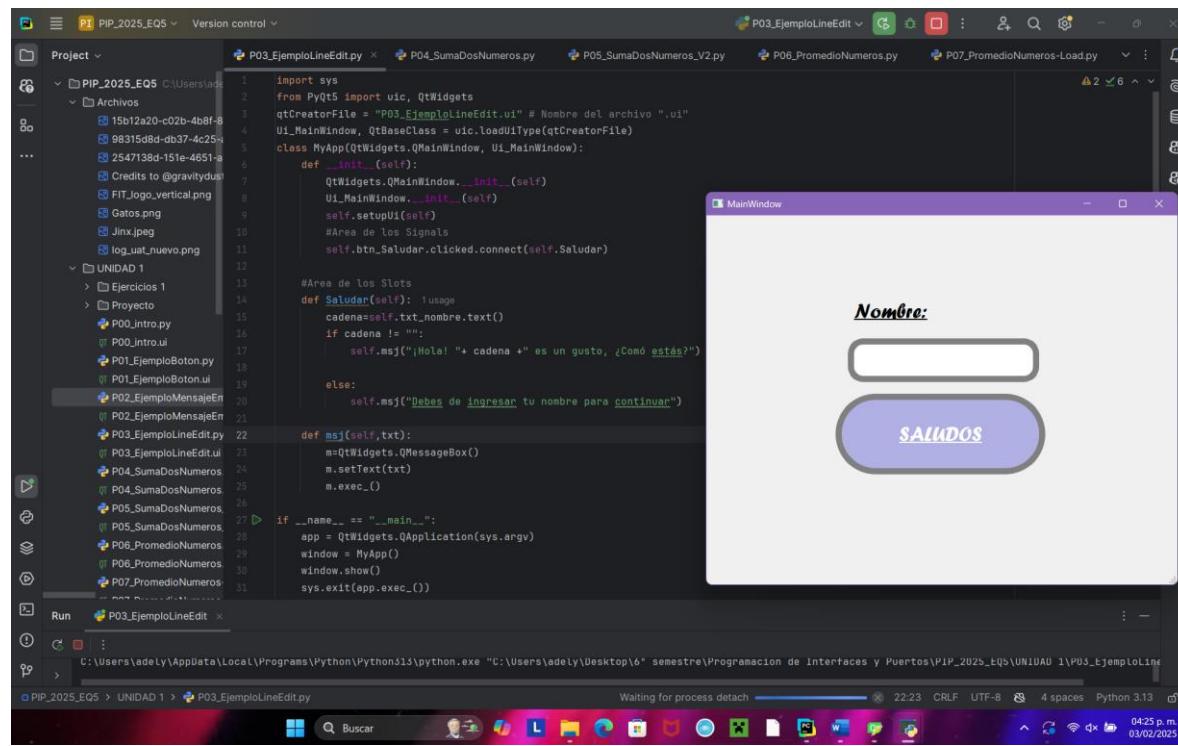
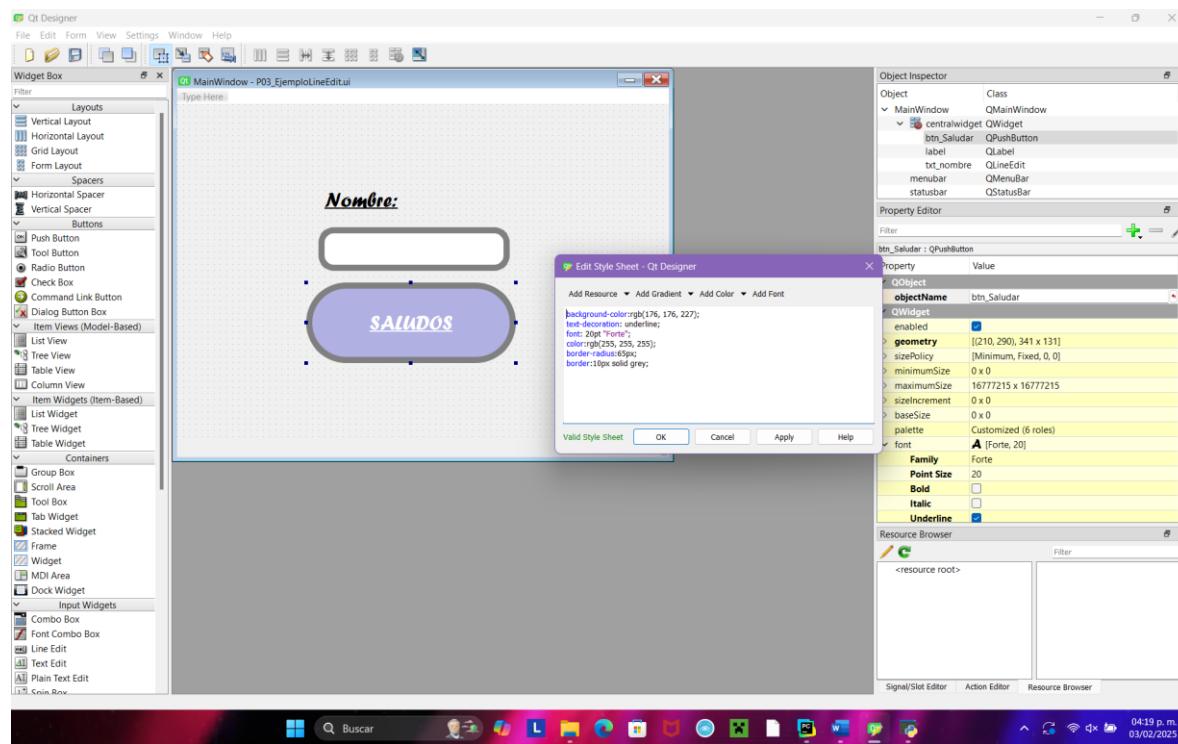
```
import sys
from PyQt5 import uic, QtWidgets
qtCreatorFile = "P02_EjemploMensajeEmergenteBasico.ui" # Nombre del archivo ".ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        #Área de los Signals
        self.btn_Saludar.clicked.connect(self.Saludar)

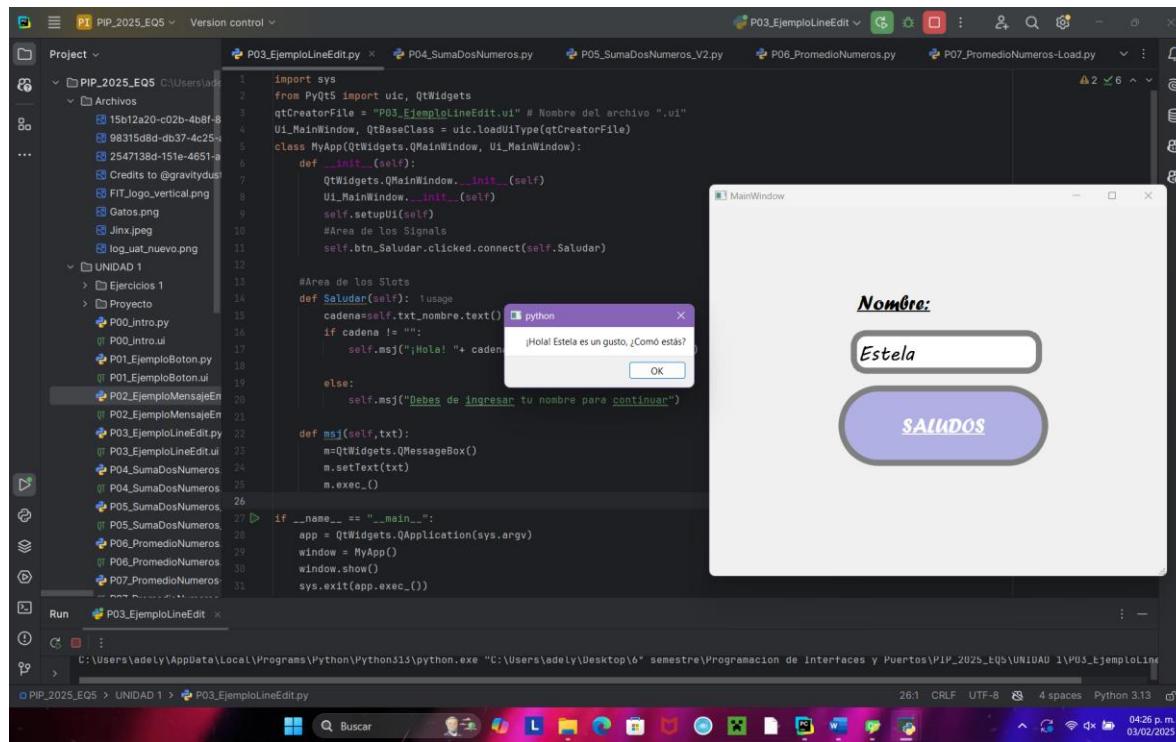
    #Área de los Slots
    def Saludar(self):
        self.msg("¡Hola! es un gusto, ¿Comó estás?")
def msg(self,txt):
    m=QtWidgets.QMessageBox()
    m.setText(txt)
    m.exec_()
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```



## C4. P03\_EjemploLineEdit



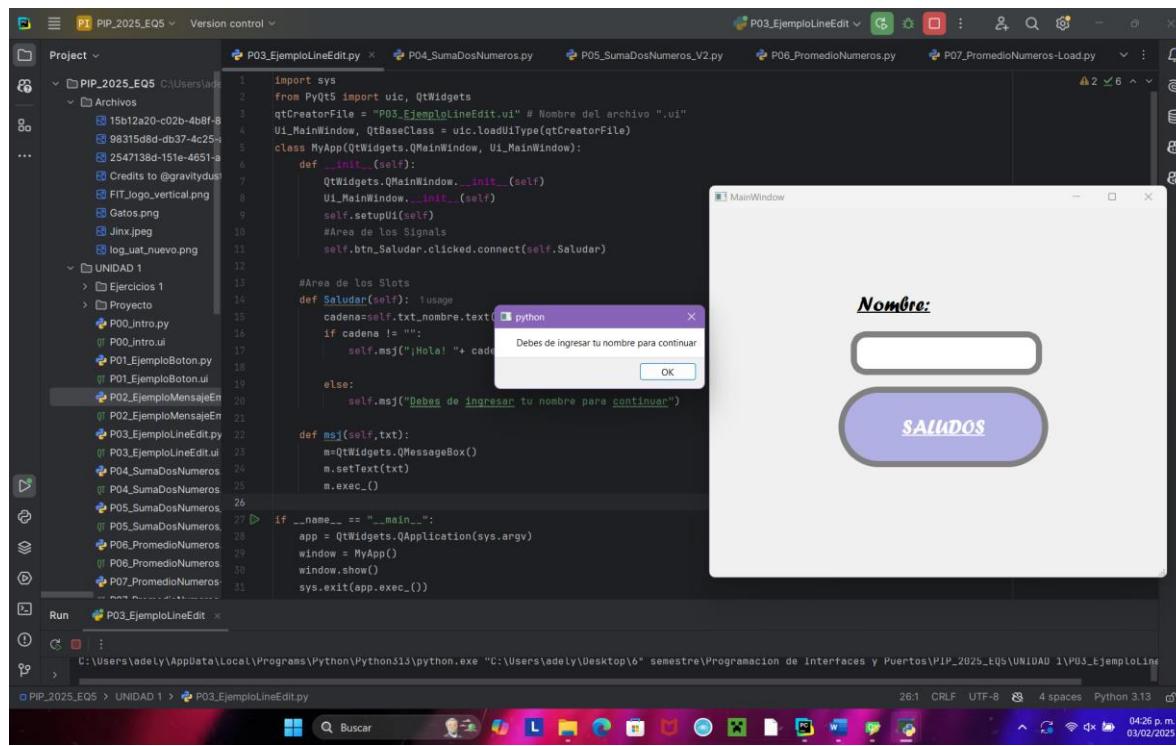




Project PIP\_2025\_EQ5

```
1 import sys
2 from PyQt5 import uic, QtWidgets
3 qtCreatorFile = "P03_EjemploLineEdit.ui" # Nombre del archivo ".ui"
4 UI_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
5 class MyApp(QtWidgets.QMainWindow, UI_MainWindow):
6     def __init__(self):
7         QtWidgets.QMainWindow.__init__(self)
8         self.setupUi(self)
9         #Área de los Signals
10        self.btn_Saludar.clicked.connect(self.Saludar)
11
12        #Área de los Slots
13    def Saludar(self): Usage
14        cadena=self.txt_nombre.text()
15        if cadena != "":
16            self.msj("¡Hola! Estela es un gusto, ¿Cómo estás?")
17        else:
18            self.msj("Debes de ingresar tu nombre para continuar")
19
20    def msj(self,txt):
21        m=QtWidgets.QMessageBox()
22        m.setText(txt)
23        m.exec_()
24
25    if __name__ == "__main__":
26        app = QtWidgets.QApplication(sys.argv)
27        window = MyApp()
28        window.show()
29        sys.exit(app.exec_())
30
```

Nombre:  
Estela  
SALUDOS



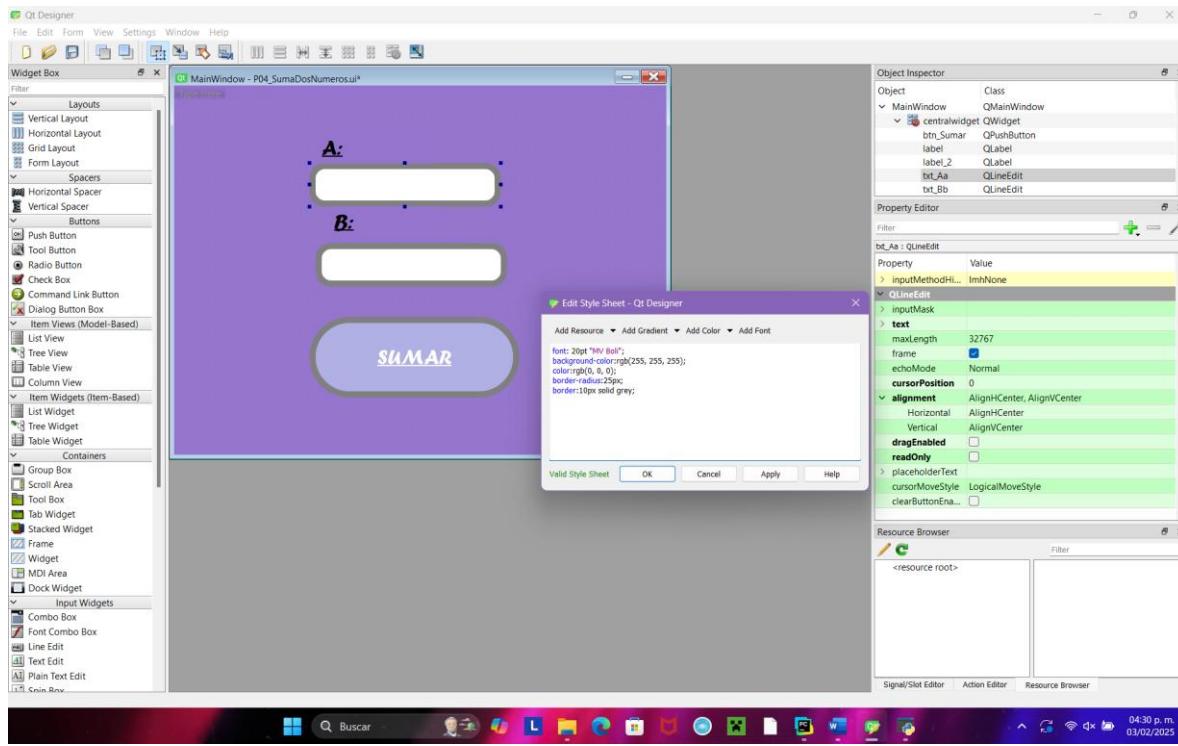
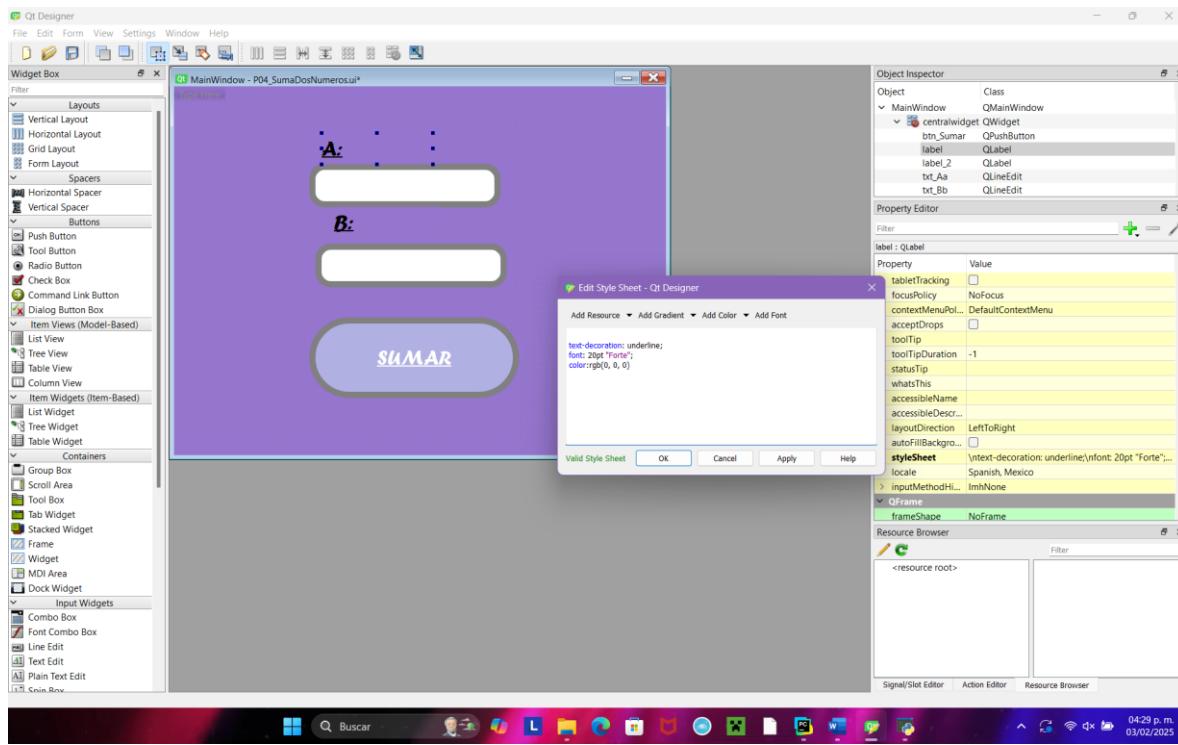
Project PIP\_2025\_EQ5

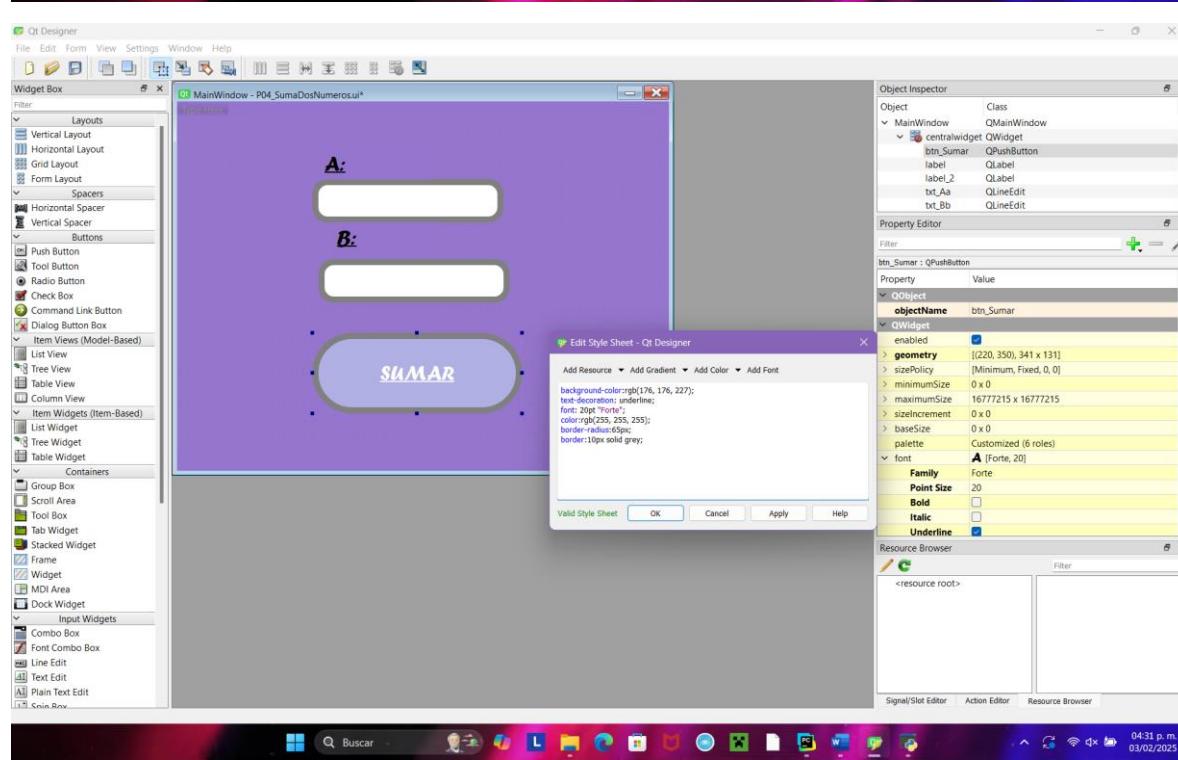
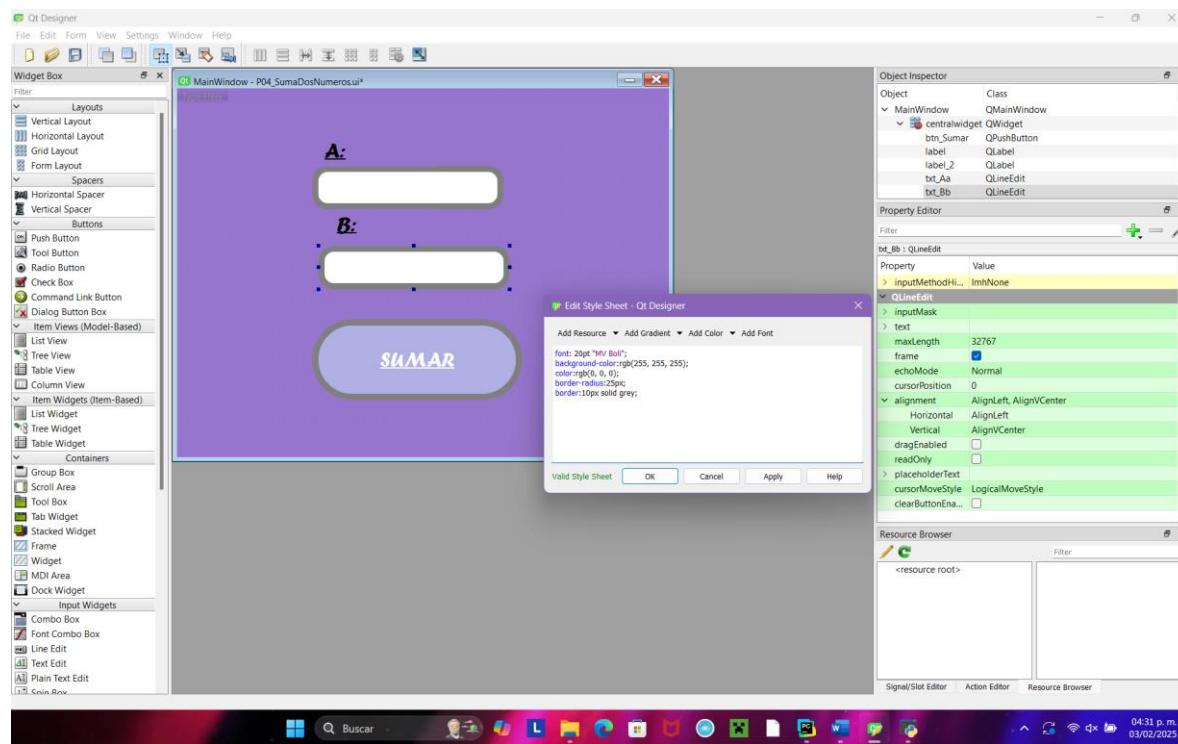
```
1 import sys
2 from PyQt5 import uic, QtWidgets
3 qtCreatorFile = "P03_EjemploLineEdit.ui" # Nombre del archivo ".ui"
4 UI_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
5 class MyApp(QtWidgets.QMainWindow, UI_MainWindow):
6     def __init__(self):
7         QtWidgets.QMainWindow.__init__(self)
8         self.setupUi(self)
9         #Área de los Signals
10        self.btn_Saludar.clicked.connect(self.Saludar)
11
12        #Área de los Slots
13    def Saludar(self): Usage
14        cadena=self.txt_nombre.text()
15        if cadena != "":
16            self.msj("¡Hola! " + cadena)
17        else:
18            self.msj("Debes de ingresar tu nombre para continuar")
19
20    def msj(self,txt):
21        m=QtWidgets.QMessageBox()
22        m.setText(txt)
23        m.exec_()
24
25    if __name__ == "__main__":
26        app = QtWidgets.QApplication(sys.argv)
27        window = MyApp()
28        window.show()
29        sys.exit(app.exec_())
30
```

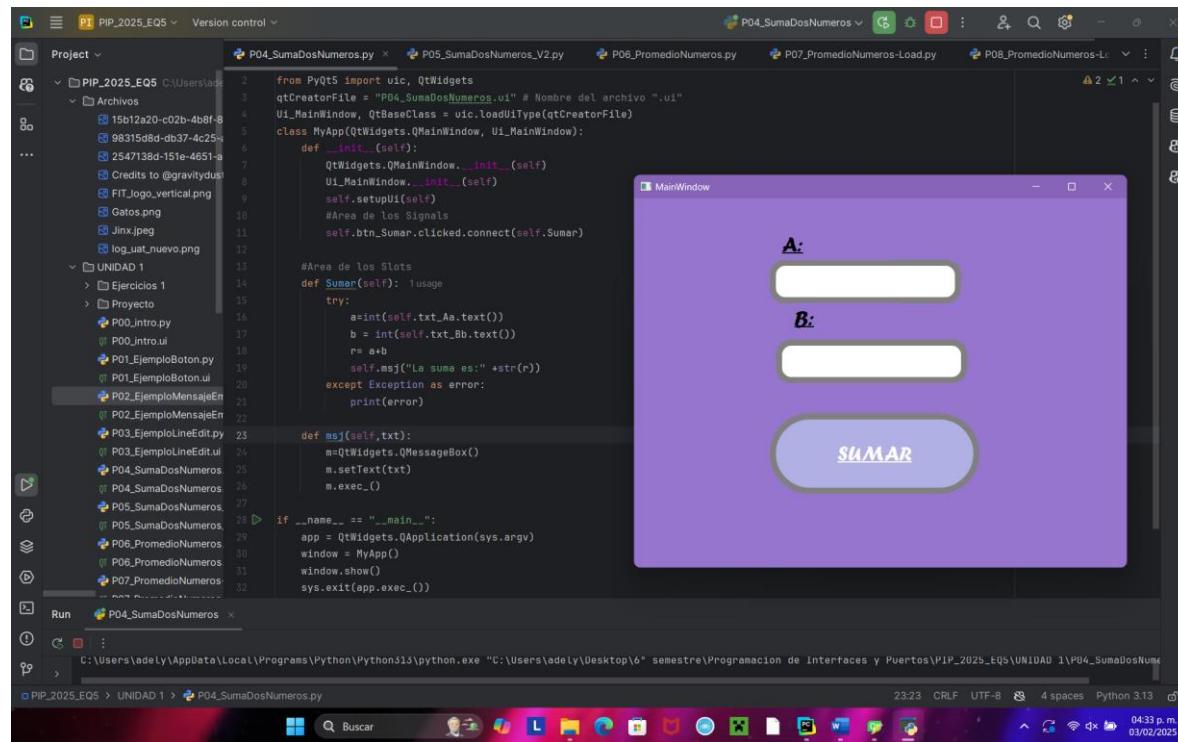
Nombre:  
SALUDOS



## C5. P04\_SumaDosNumeros





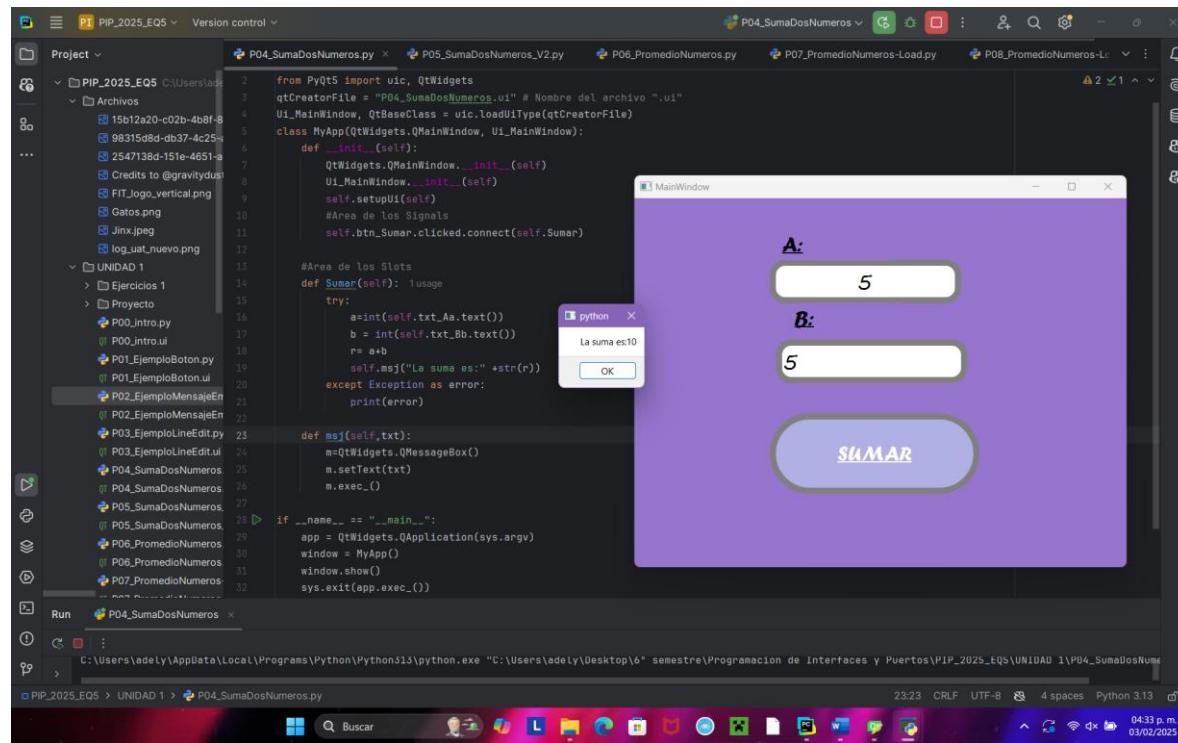


```
from PyQt5 import uic, QtWidgets
qtCreatorFile = "P04_SumaDosNumeros.ui" # Nombre del archivo ".ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        #Area de los Signals
        self.btn_Sumar.clicked.connect(self.Sumar)

    #Area de los Slots
    def Sumar(self): usage
        try:
            a=int(self.txt_Aa.text())
            b = int(self.txt_Bb.text())
            r= a+b
            self.msj("La suma es:" +str(r))
        except Exception as error:
            print(error)

    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```



```
from PyQt5 import uic, QtWidgets
qtCreatorFile = "P04_SumaDosNumeros.ui" # Nombre del archivo ".ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        #Area de los Signals
        self.btn_Sumar.clicked.connect(self.Sumar)

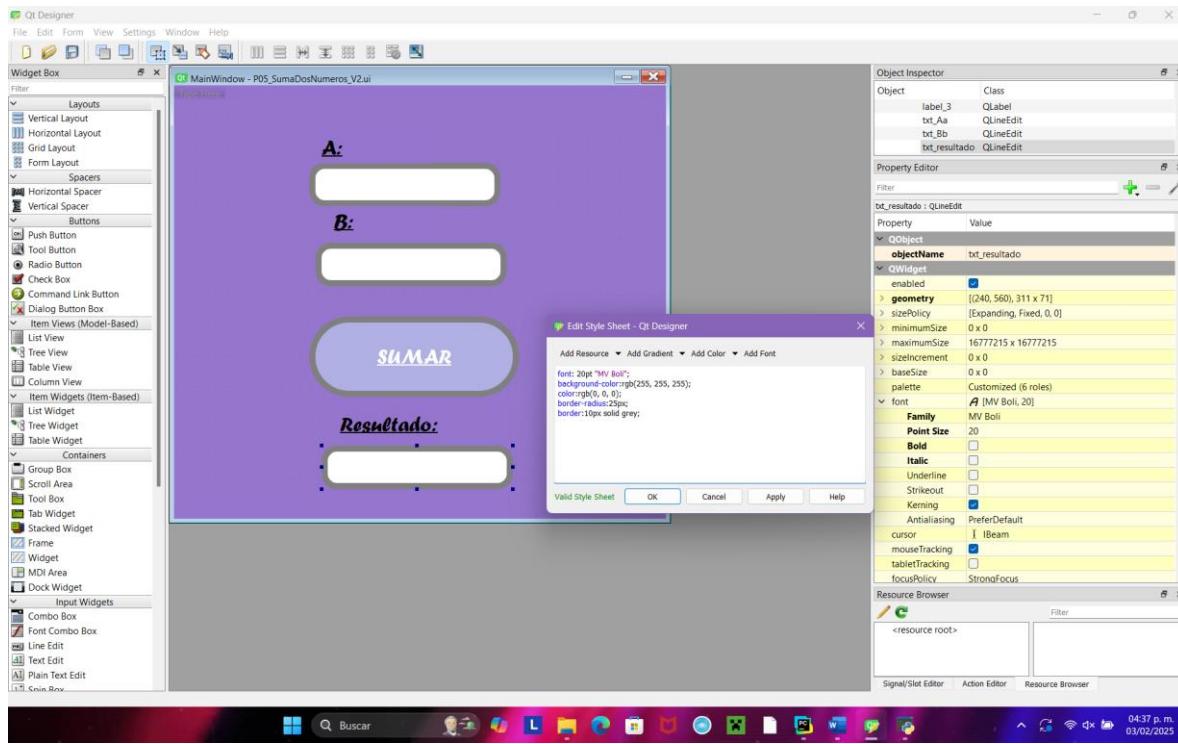
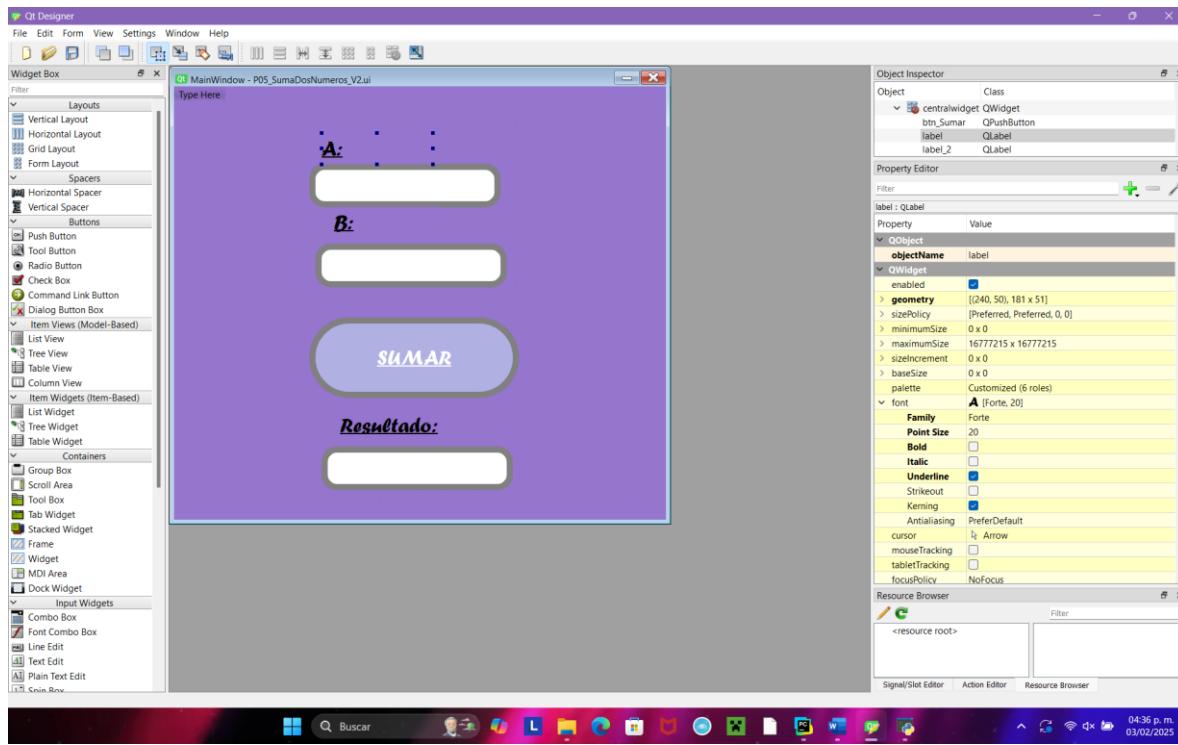
    #Area de los Slots
    def Sumar(self): usage
        try:
            a=int(self.txt_Aa.text())
            b = int(self.txt_Bb.text())
            r= a+b
            self.msj("La suma es:" +str(r))
        except Exception as error:
            print(error)

    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```



## C6. P05\_SumaDosNumeros\_V2





The screenshot shows the PyCharm IDE interface with the project **PIP\_2025\_EQ5** open. The code editor displays **P05\_SumaDosNumeros\_V2.py**, which contains Python code for a simple addition application. The application window titled **MainWindow** is shown, featuring two input fields labeled **A:** and **B:**, a **SUMAR** button, and a result field labeled **Resultado:**. The result field displays the value **15**.

```
> import ...
qtCreatorFile = "P05_SumaDosNumeros_V2.ui" # Nombre del archivo ".ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        #Área de los Signals
        self.btn_Sumar.clicked.connect(self.Sumar)

    #Área de los Slots
    def Sumar(self): lusage
        try:
            a=int(self.txt_Aa.text())
            b = int(self.txt_Bb.text())
            r= a+b
            #self.msj("La suma es:" +str(r))
            self.txt_resultado.setText(str(r))
        except Exception as error:
            print(error)

    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```

The screenshot shows the PyCharm IDE interface with the project **PIP\_2025\_EQ5** open. The code editor displays **P05\_SumaDosNumeros\_V2.py**, which contains Python code for a simple addition application. The application window titled **MainWindow** is shown, featuring two input fields labeled **A:** and **B:**, a **SUMAR** button, and a result field labeled **Resultado:**. The result field displays the value **15**.

```
> import ...
qtCreatorFile = "P05_SumaDosNumeros_V2.ui" # Nombre del archivo ".ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        #Área de los Signals
        self.btn_Sumar.clicked.connect(self.Sumar)

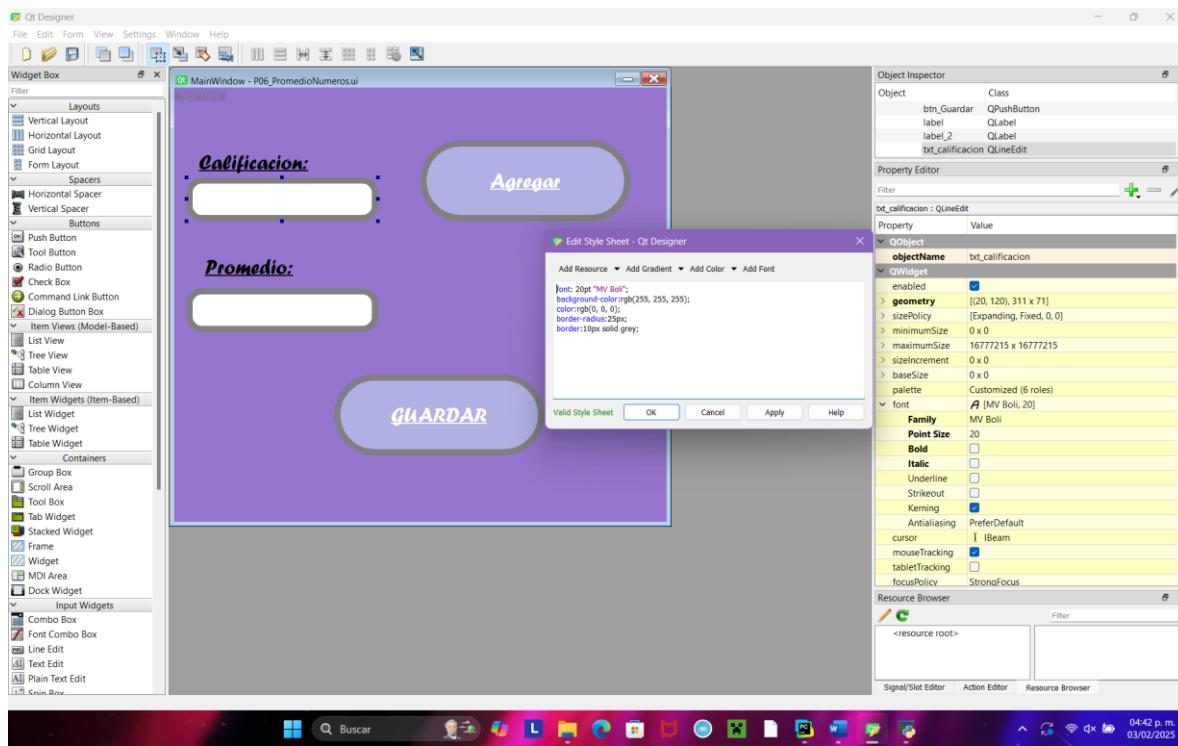
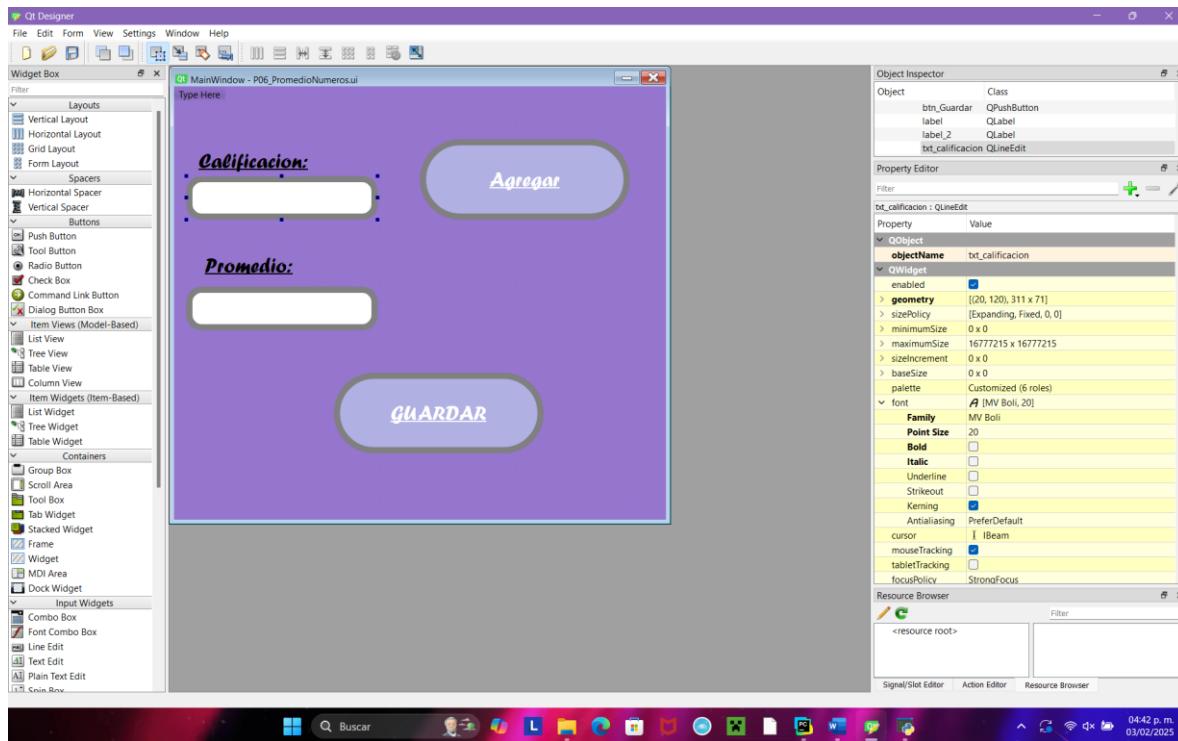
    #Área de los Slots
    def Sumar(self): lusage
        try:
            a=int(self.txt_Aa.text())
            b = int(self.txt_Bb.text())
            r= a+b
            #self.msj("La suma es:" +str(r))
            self.txt_resultado.setText(str(r))
        except Exception as error:
            print(error)

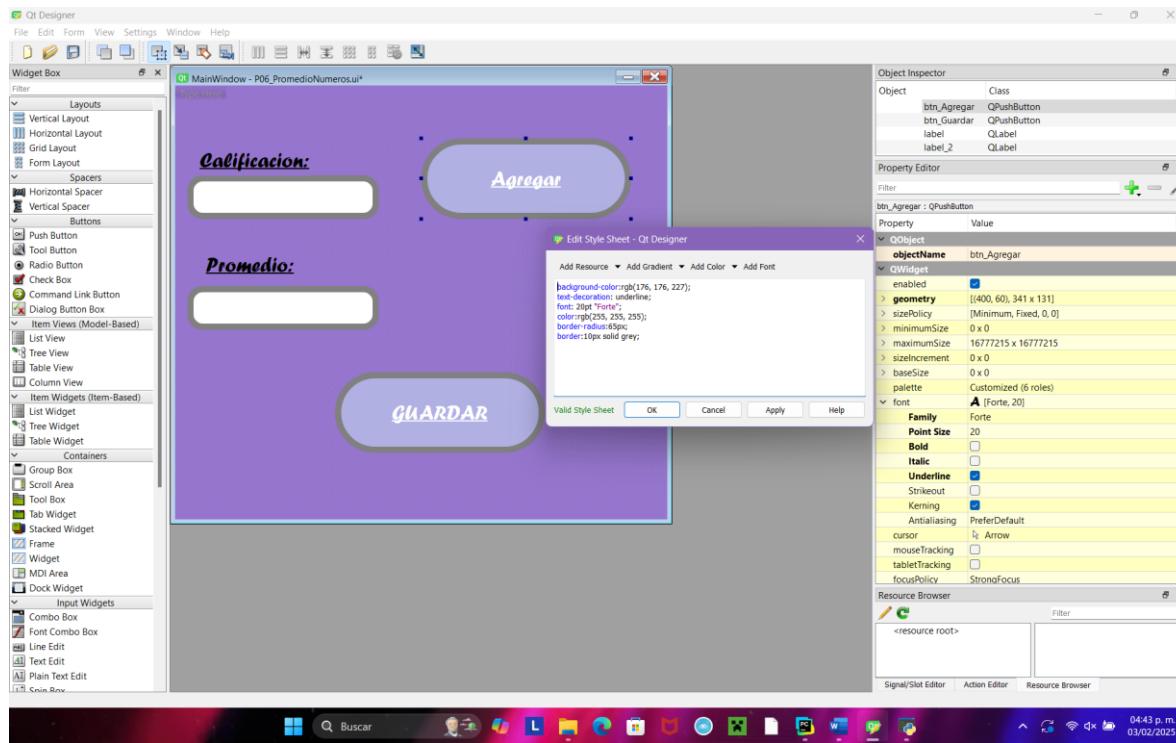
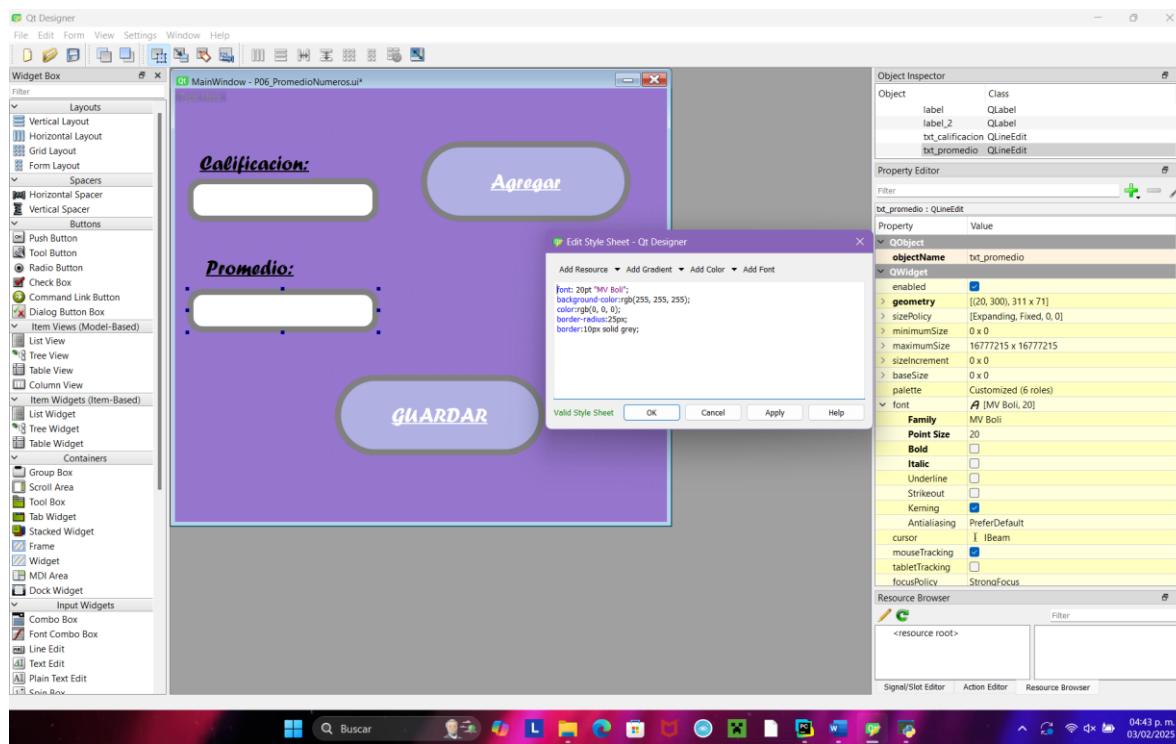
    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

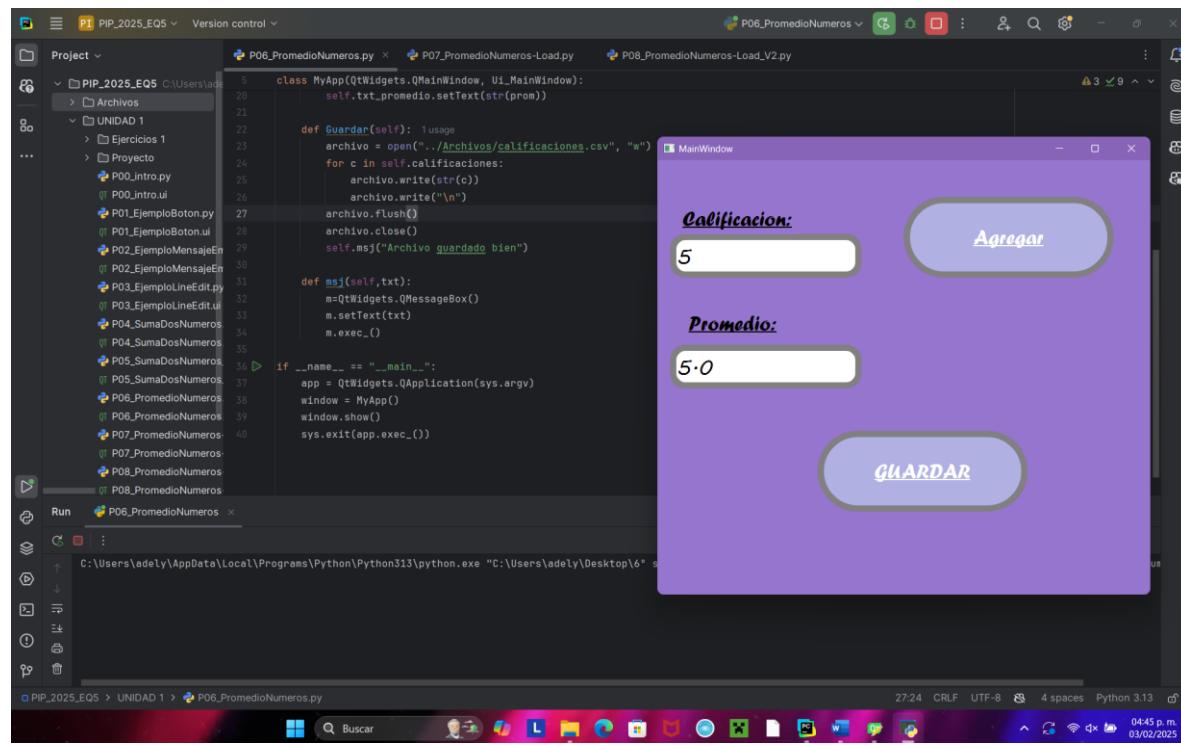
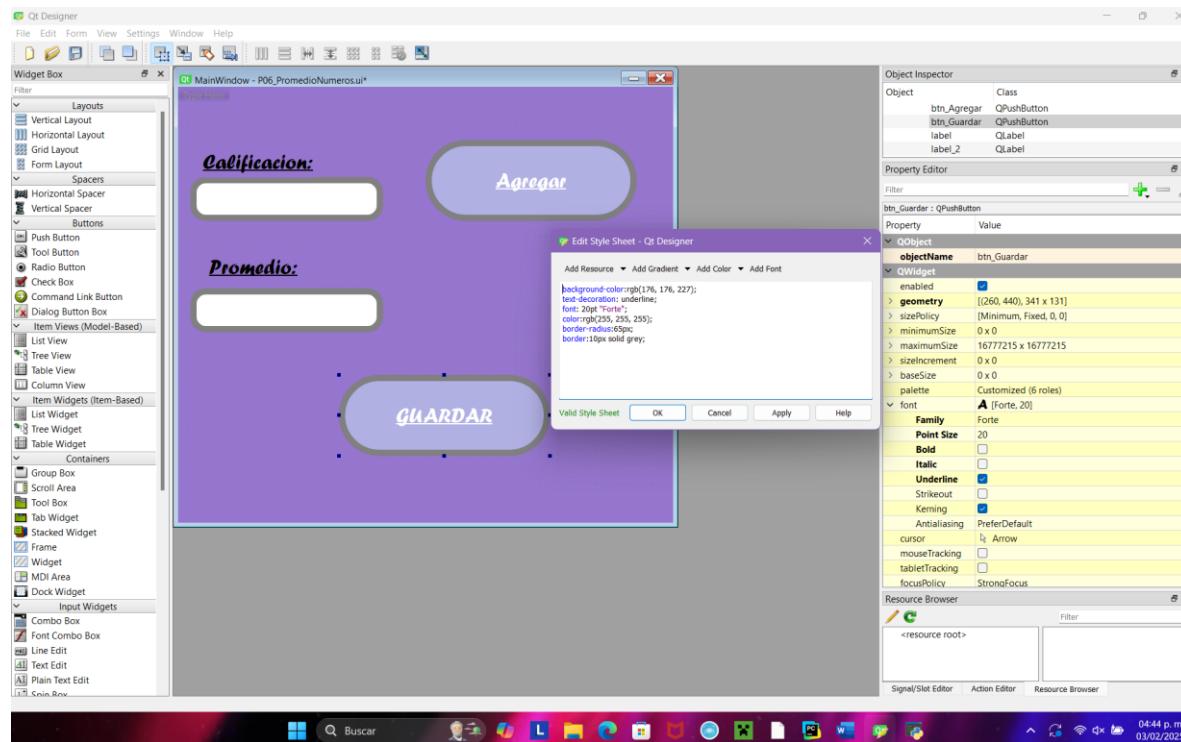
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```



## C7. P06\_PromedioNumeros







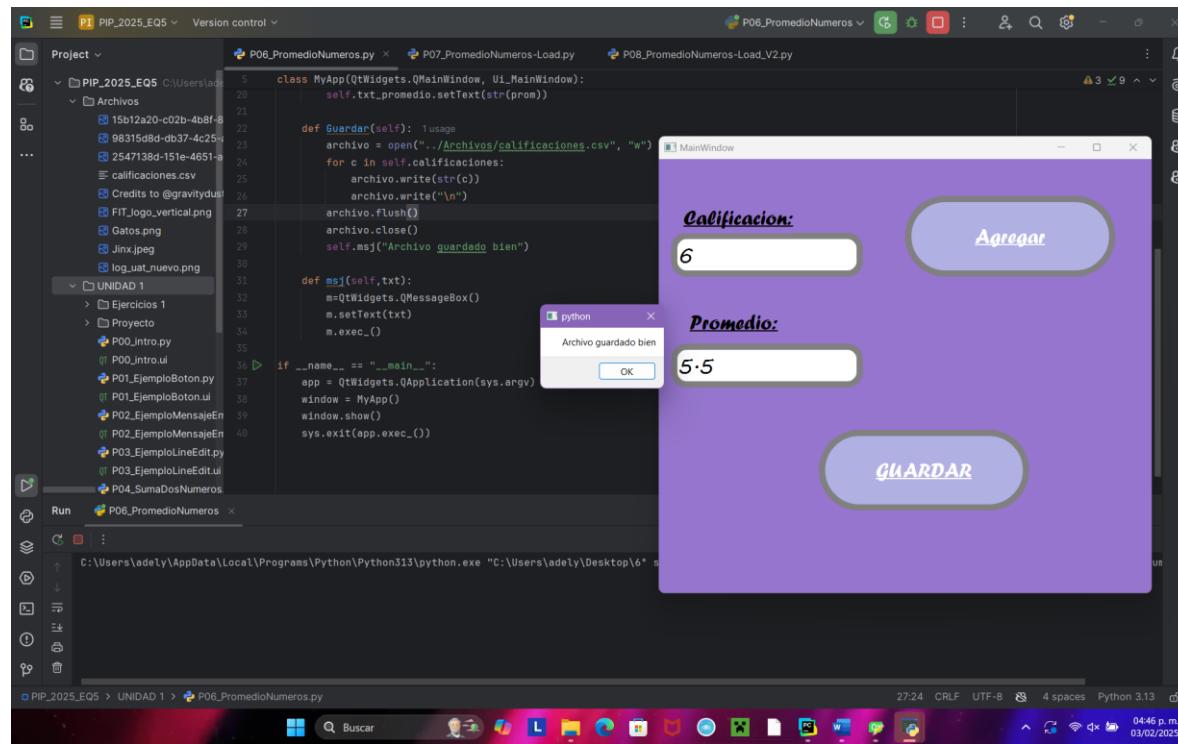


The screenshot shows the PyCharm IDE interface. On the left is the project tree for 'PIP\_2025\_EQ5' containing files like P06\_PromedioNumeros.py, P07\_PromedioNumeros-Load.py, and P08\_PromedioNumeros-Load\_V2.py. The main editor shows the code for P06\_PromedioNumeros.py, which includes a 'Guardar' function that writes to a CSV file and an 'Agregar' button. A message box from the application is visible in the foreground, displaying 'Archivo guardado bien'. The application window titled 'MainWindow' has fields for 'Calificación:' (6) and 'Promedio:' (5.5), with 'Agregar' and 'GUARDAR' buttons.

```
5     class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
6         self.txt_promedio.setText(str(prom))
7
8         def Guardar(self):
9             usage
10            archivo = open("./Archivos/calificaciones.csv", "w")
11            for c in self.calificaciones:
12                archivo.write(str(c))
13                archivo.write("\n")
14            archivo.flush()
15            archivo.close()
16            self.msj("Archivo guardado bien")
17
18            def msj(self,txt):
19                m=QtWidgets.QMessageBox()
20                m.setText(txt)
21                m.exec_()
22
23            if __name__ == "__main__":
24                app = QtWidgets.QApplication(sys.argv)
25                window = MyApp()
26                window.show()
27                sys.exit(app.exec_())
28
29
30
31
32
33
34
35
36
37
38
39
40
```

The screenshot shows the PyCharm IDE interface. On the left is the project tree for 'PIP\_2025\_EQ5' containing files like P06\_PromedioNumeros.py, P07\_PromedioNumeros-Load.py, and P08\_PromedioNumeros-Load\_V2.py. The main editor shows the same Python code as the previous screenshot. A message box from the application is visible in the foreground, displaying 'Archivo guardado bien'. The application window titled 'MainWindow' has fields for 'Calificación:' (6) and 'Promedio:' (5.5), with 'Agregar' and 'GUARDAR' buttons.

```
5     class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
6         self.txt_promedio.setText(str(prom))
7
8         def Guardar(self):
9             usage
10            archivo = open("./Archivos/calificaciones.csv", "w")
11            for c in self.calificaciones:
12                archivo.write(str(c))
13                archivo.write("\n")
14            archivo.flush()
15            archivo.close()
16            self.msj("Archivo guardado bien")
17
18            def msj(self,txt):
19                m=QtWidgets.QMessageBox()
20                m.setText(txt)
21                m.exec_()
22
23            if __name__ == "__main__":
24                app = QtWidgets.QApplication(sys.argv)
25                window = MyApp()
26                window.show()
27                sys.exit(app.exec_())
28
29
30
31
32
33
34
35
36
37
38
39
40
```

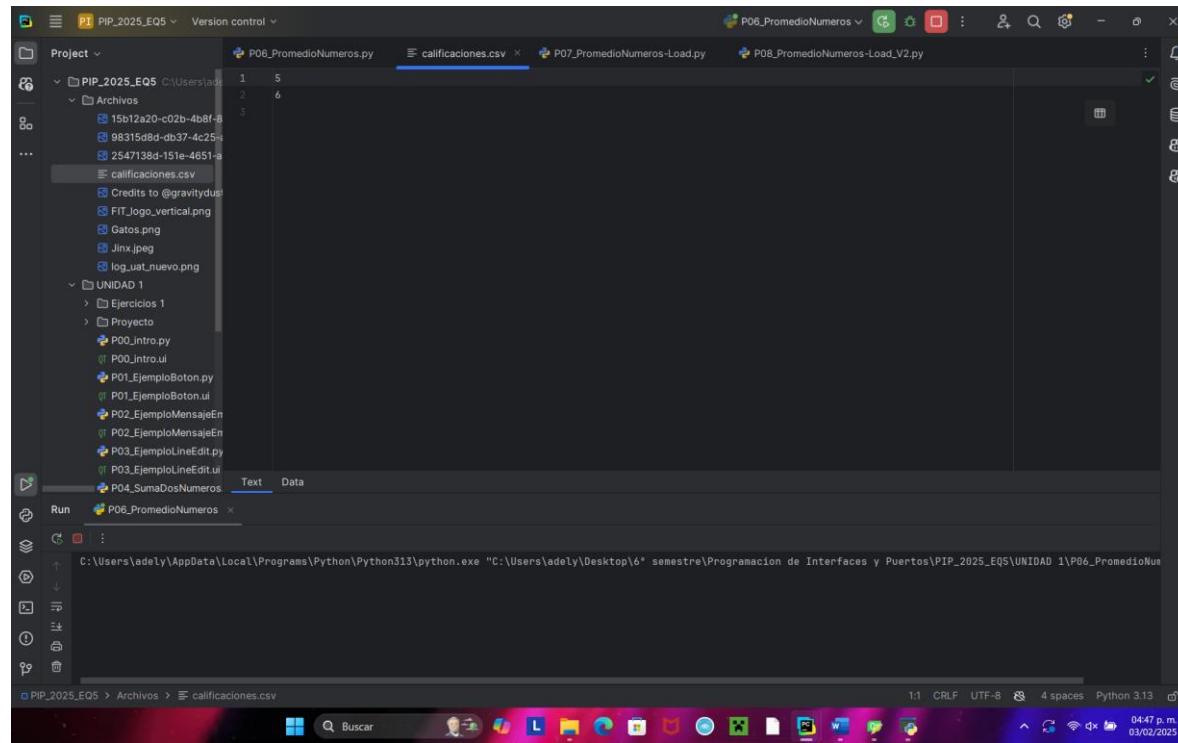


```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        self.txt_promedio.setText(str(prom))
        self.txt_calificacion.setText(str(calificacion))

    def Guardar(self):
        archivo = open("./Archivos/calificaciones.csv", "w")
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo guardado bien")

    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

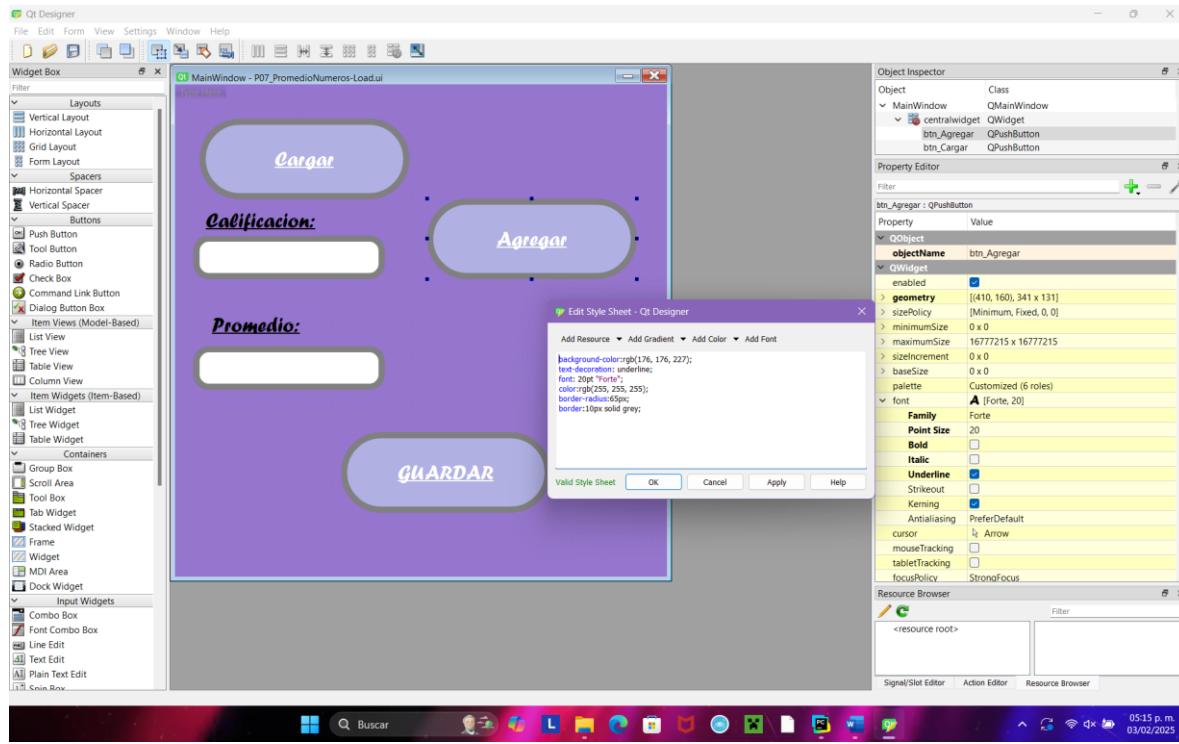
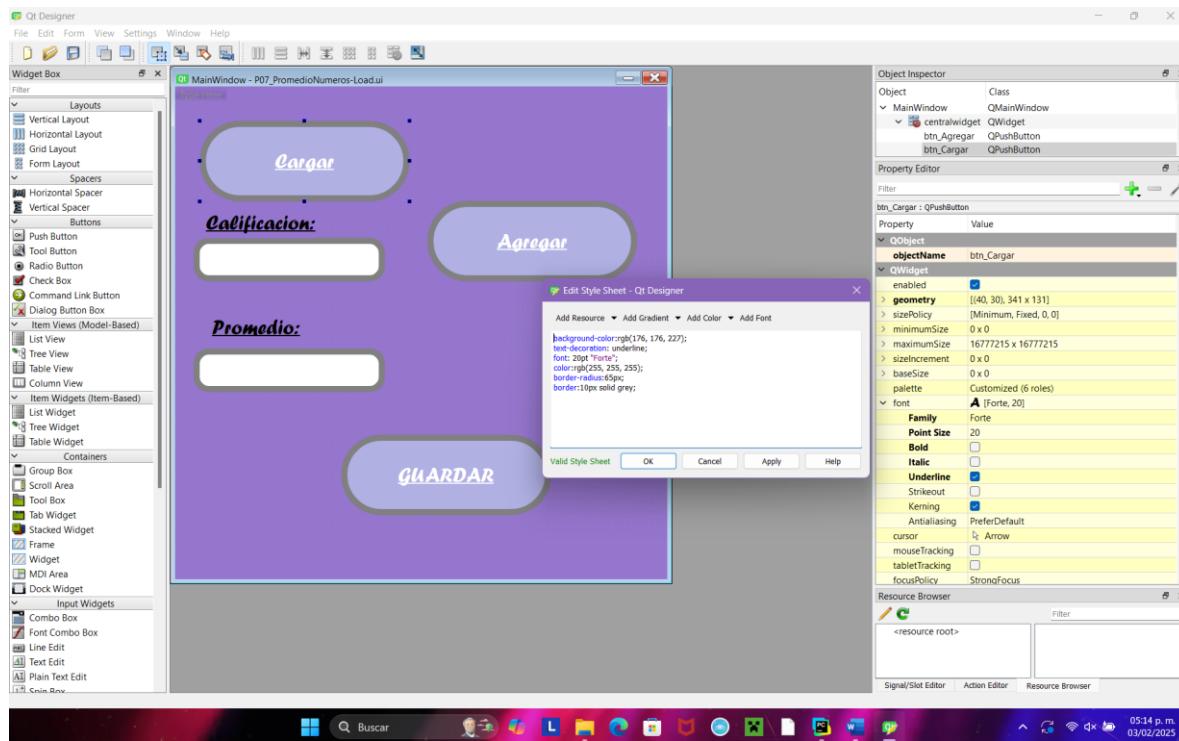
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```

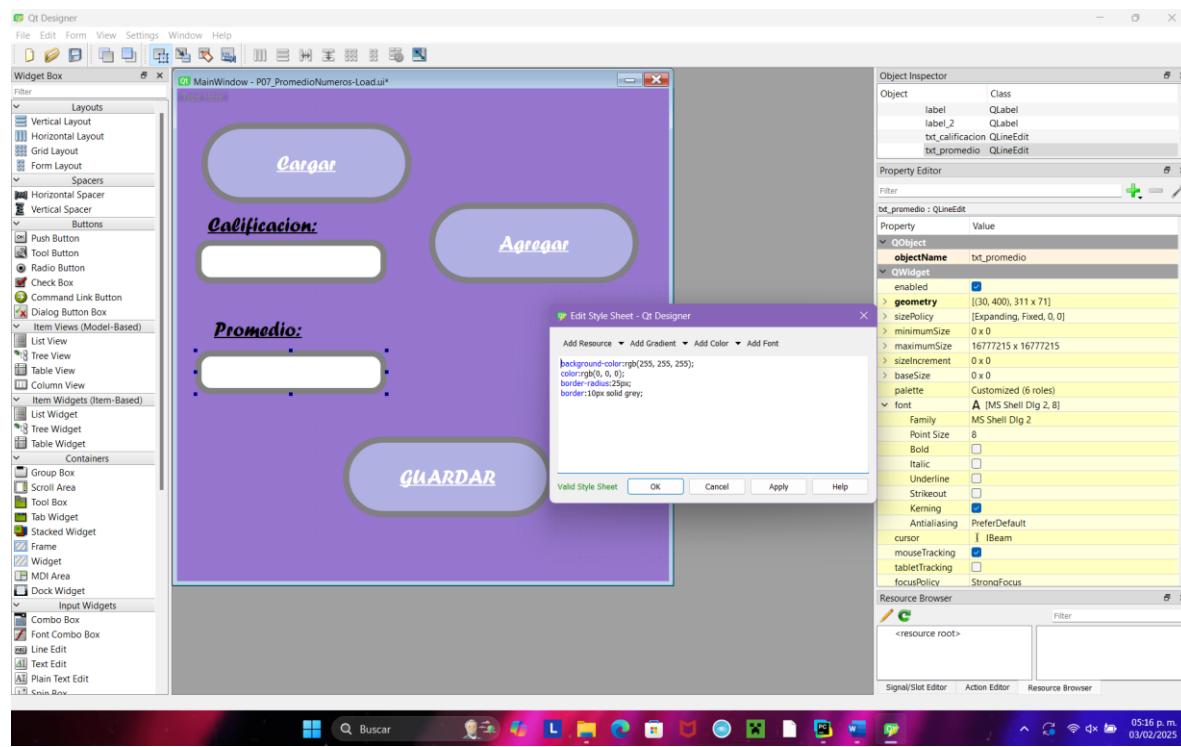
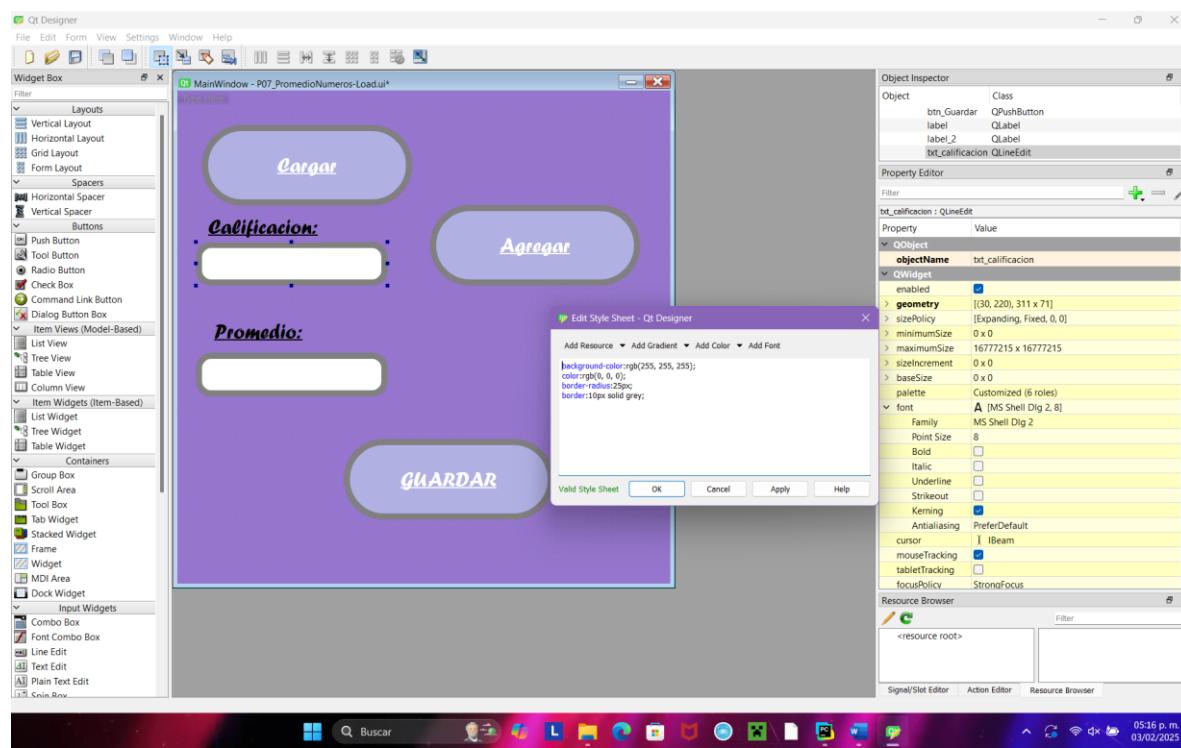


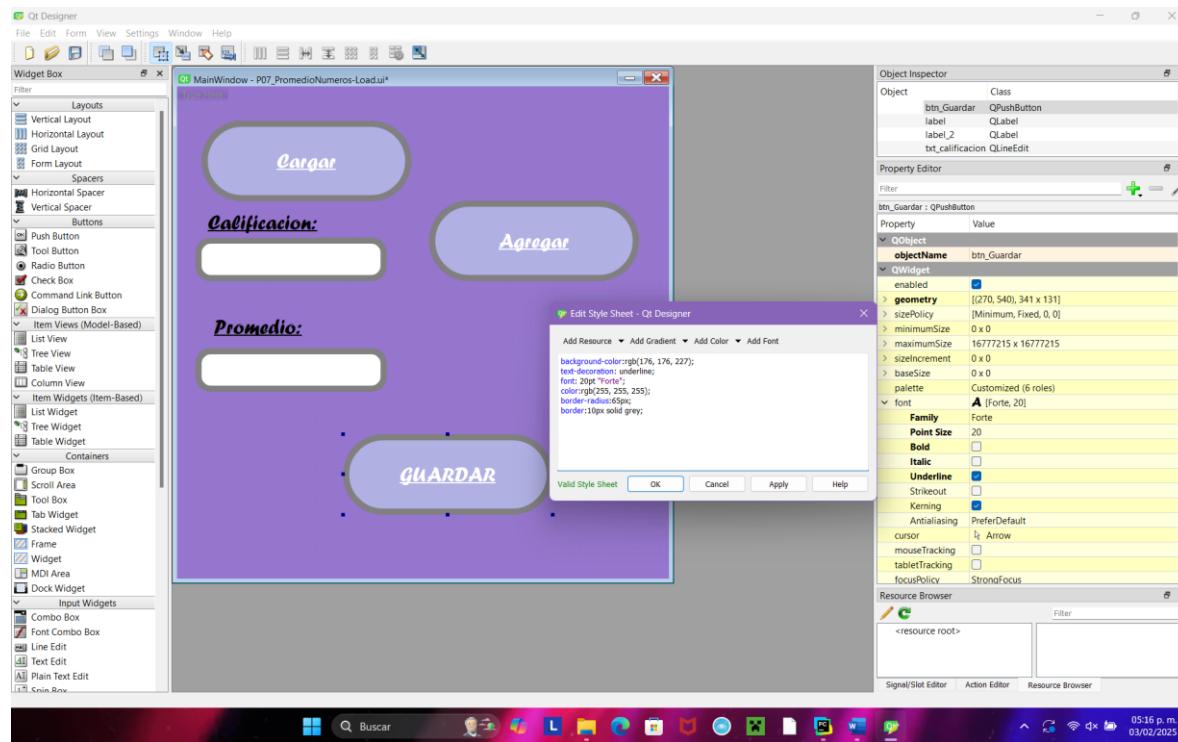
```
1,5
2,6
```



## C8. P07\_PromedioNumeros-Load

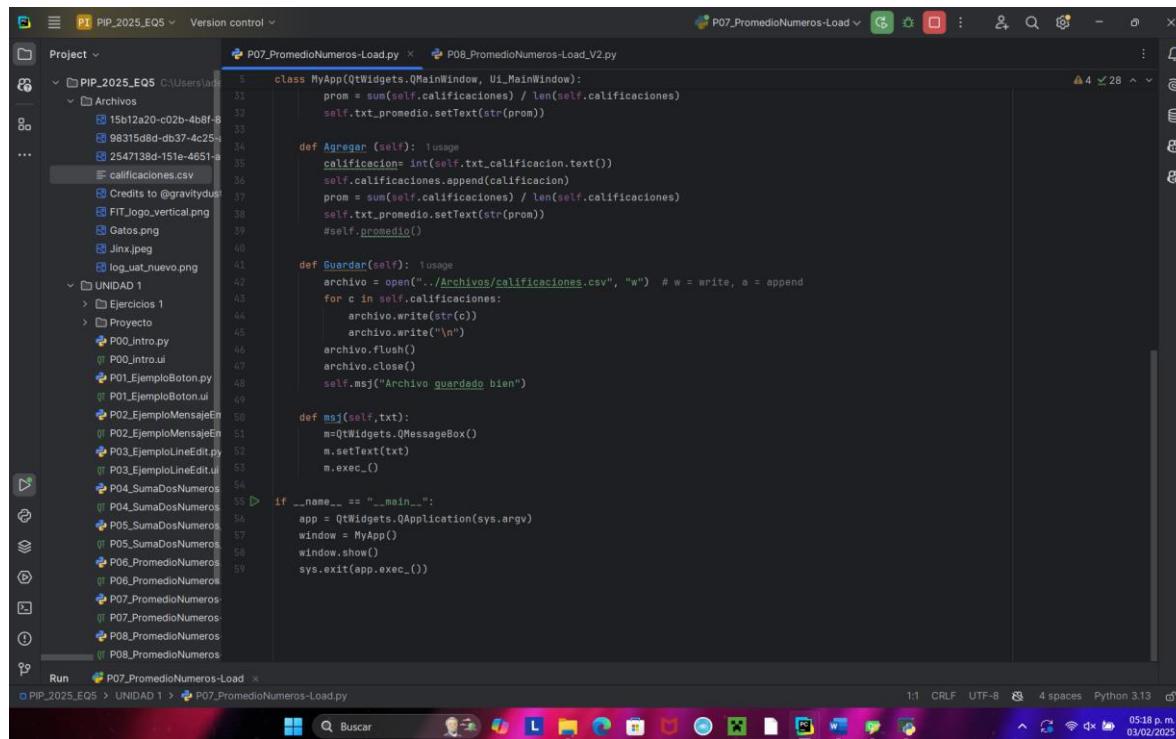






```
Project PIP_2025_EQ5 Version control
P07_PromedioNumeros-Load.py P08_PromedioNumeros-Load_V2.py

1 import sys
2 from PyQt5 import uic, QtWidgets
3
4 qtCreatorFile = "P07_PromedioNumeros-Load.ui" # Nombre del archivo ".ui"
5 Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
6
7 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
8     def __init__(self):
9         QtWidgets.QMainWindow.__init__(self)
10        self.setupUi(self)
11        #Área de los Signals
12        self.btn_Cargar.clicked.connect(self.Cargar)
13        self.btn_Agregar.clicked.connect(self.Agregar)
14        self.btn_Guardar.clicked.connect(self.Guardar)
15        self.calificaciones = []
16
17        #Área de los Slots
18        def Cargar (self): #usage
19            #Tarea como compruebo si el archivo existe
20            #ejercicio 10
21            # tarea ej 11 en lugar de sobre escribir, concatenar
22            #tarea ej 12 asegurarse de que solo se pueda cargar hasta antes de
23            #agregar la grmera calificación enables y/o código
24            archivo = open("./Archivos/calificaciones.csv")
25            contenido = archivo.readlines()
26            print(contenido)
27            datos = [int(x) for x in contenido]
28            print(datos)
29            self.calificaciones = datos
30
31        def promedio(self):
32            prom = sum(self.calificaciones) / len(self.calificaciones)
33            self.txt_promedio.setText(str(prom))
34
35        def Agregar (self): #usage
36            calificacion= int(self.txt_calificacion.text())
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
```



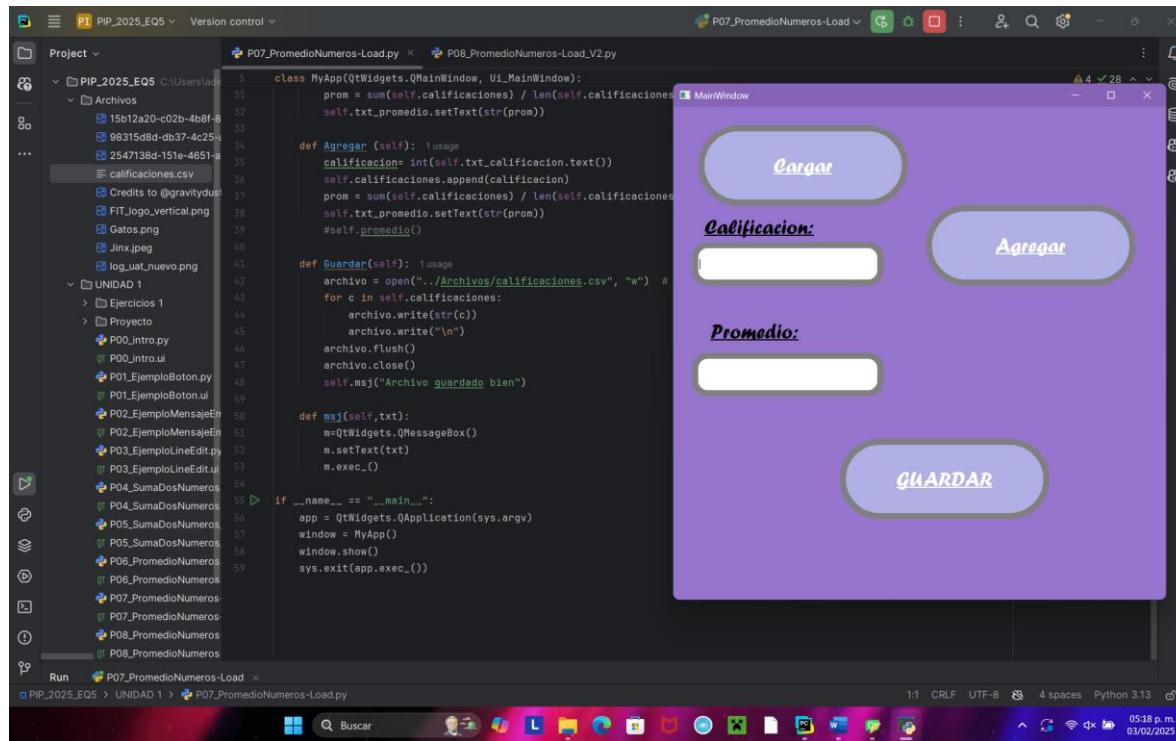
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        self.txt_promedio.setText("0.0")
        self.txt_calificacion.setText("0.0")

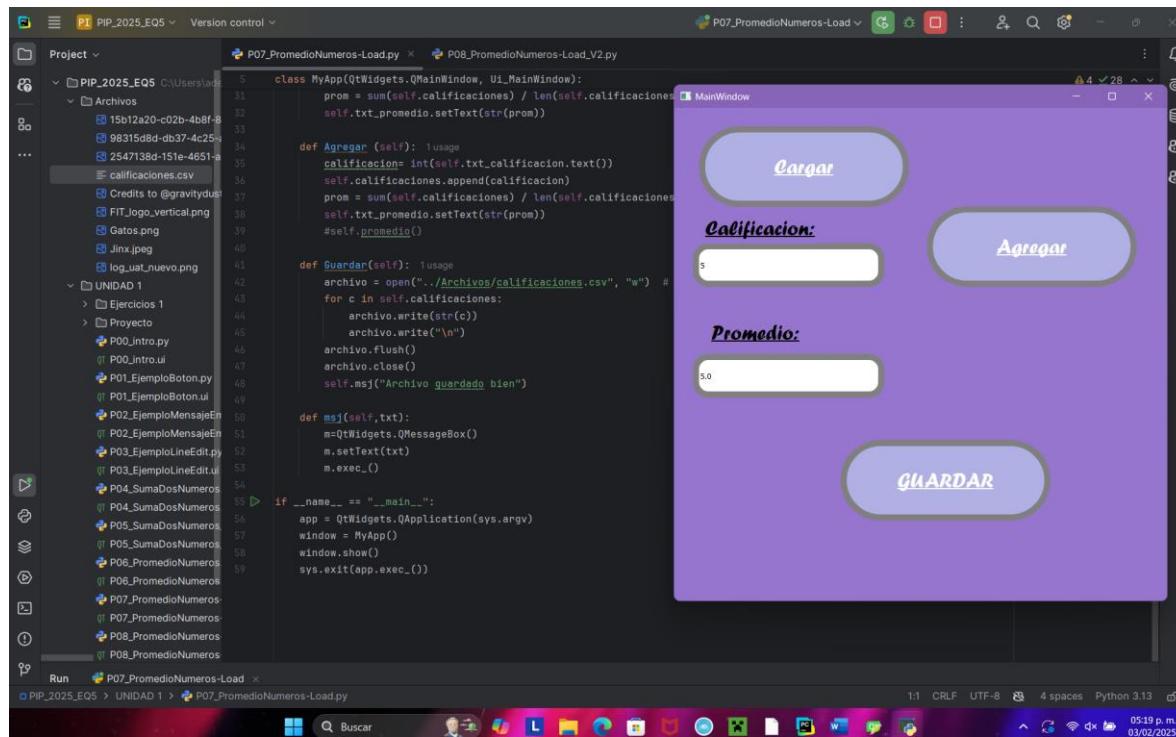
    def Agregar(self):
        calificacion = int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))
        self.lbl_promedio.setText(str(prom))

    def Guardar(self):
        archivo = open("./Archivos/calificaciones.csv", "w") # w = write, a = append
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo guardado bien")

    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```





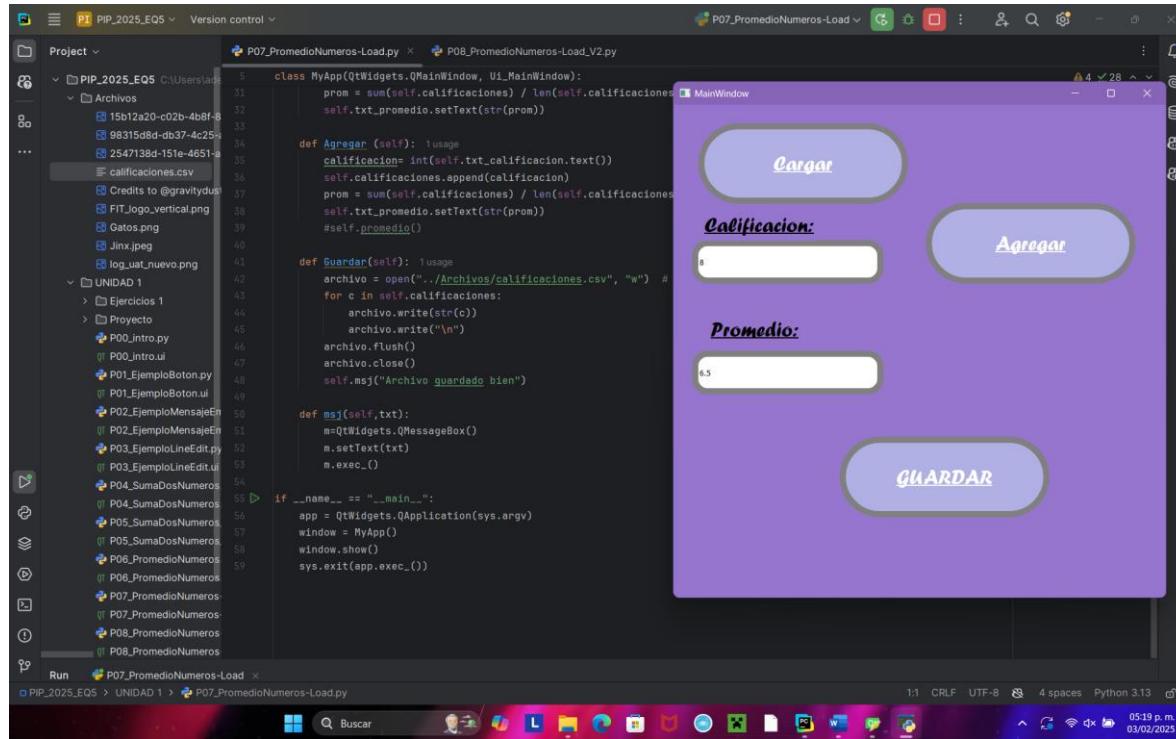
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    prom = sum(self.calificaciones) / len(self.calificaciones)
    self.txt_promedio.setText(str(prom))

    def Agregar(self):  # usage
        calificacion= int(self.txt_calificación.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))
        self.promedio()

    def Guardar(self):  # usage
        archivo = open("./Archivos/calificaciones.csv", "w")  #
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo guardado bien")

    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    prom = sum(self.calificaciones) / len(self.calificaciones)
    self.txt_promedio.setText(str(prom))

    def Agregar(self):  # usage
        calificacion= int(self.txt_calificación.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))
        self.promedio()

    def Guardar(self):  # usage
        archivo = open("./Archivos/calificaciones.csv", "w")  #
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo guardado bien")

    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```



The screenshot shows the PyCharm IDE interface. On the left, the project structure for 'PIP\_2025\_EQ5' is visible, including files like 'calificaciones.csv', 'P07\_PromedioNumeros-Load.py', and 'P08\_PromedioNumeros-Load2.py'. The main editor window displays the Python code for 'P07\_PromedioNumeros-Load.py'. A message box in the center says 'Archivo guardado bien'. To the right, a screenshot of a Windows application window titled 'MainWindow' is shown. The window has four rounded rectangular buttons: 'Cargar' (Load), 'Calificación:' (Grade:), 'Agregar' (Add), and 'Promedio:' (Average:). Below these buttons is a text input field containing '6.5'. At the bottom right of the window is a button labeled 'GUARDAR' (Save).

This screenshot is identical to the one above, showing the PyCharm IDE with the same project structure and code in 'P07\_PromedioNumeros-Load.py'. The message box still says 'Archivo guardado bien'. The application window 'MainWindow' is also present, showing the same state with the grade '6.5' entered and the 'GUARDAR' button at the bottom right.



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    prom = sum(self.calificaciones) / len(self.calificaciones)
    self.txt_promedio.setText(str(prom))

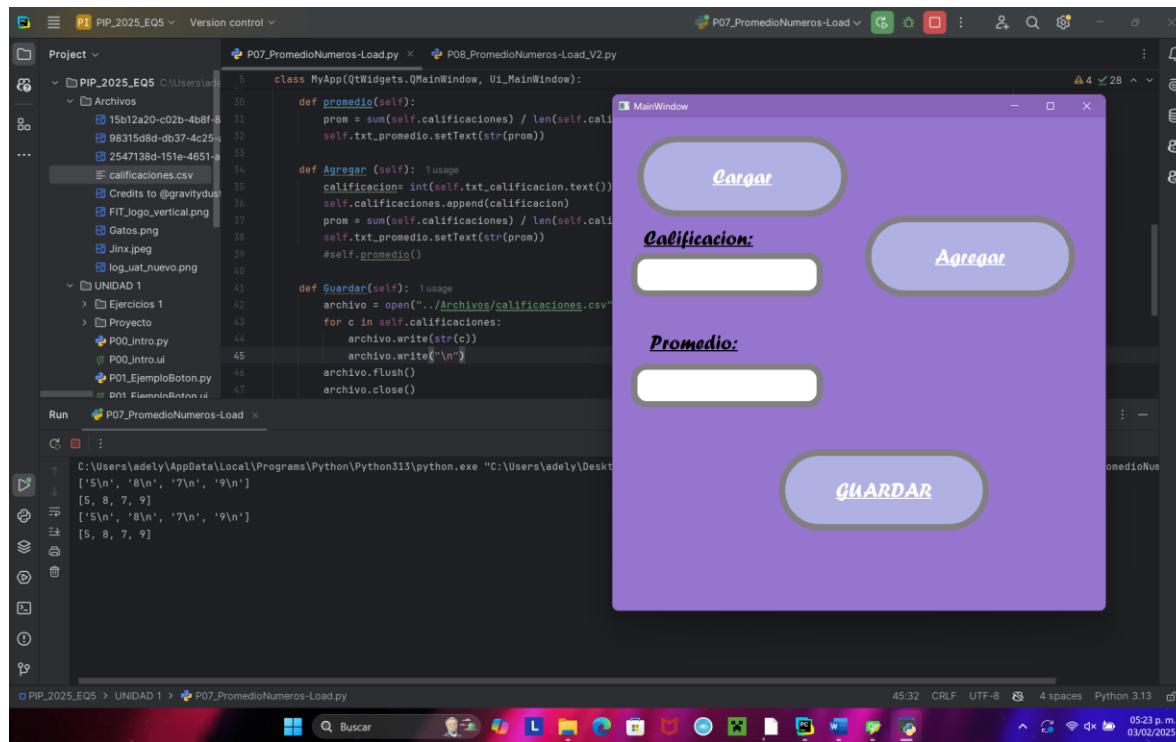
    def Agregar(self):  # usage
        calificacion= int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))
        self.promedio()

    def Guardar(self):  # usage
        archivo = open("../Archivos/calificaciones.csv", "w") #
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo guardado bien")
```

```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    prom = sum(self.calificaciones) / len(self.calificaciones)
    self.txt_promedio.setText(str(prom))

    def Agregar(self):  # usage
        calificacion= int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))
        self.promedio()

    def Guardar(self):  # usage
        archivo = open("../Archivos/calificaciones.csv", "w") #
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo guardado bien")
```

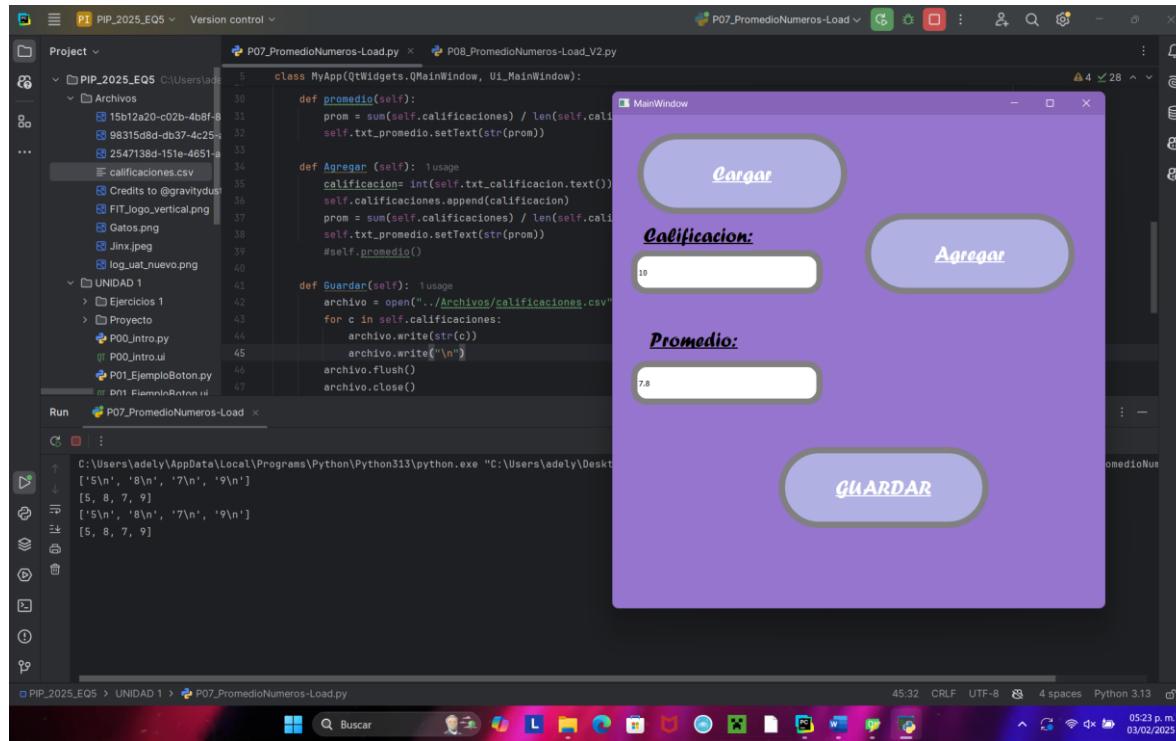


```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

    def promedio(self):
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))

    def Agregar(self):
        calificacion = int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))
        self.promedio()

    def Guardar(self):
        archivo = open("./Archivos/calificaciones.csv", "w")
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
```

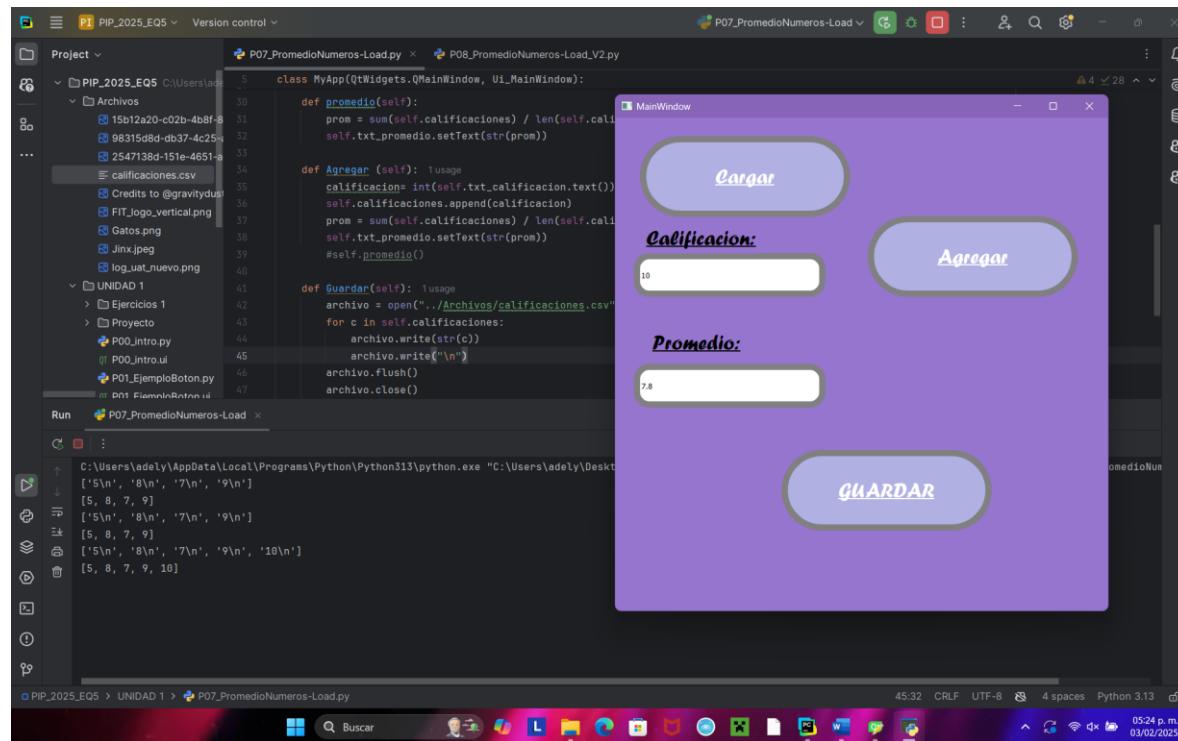
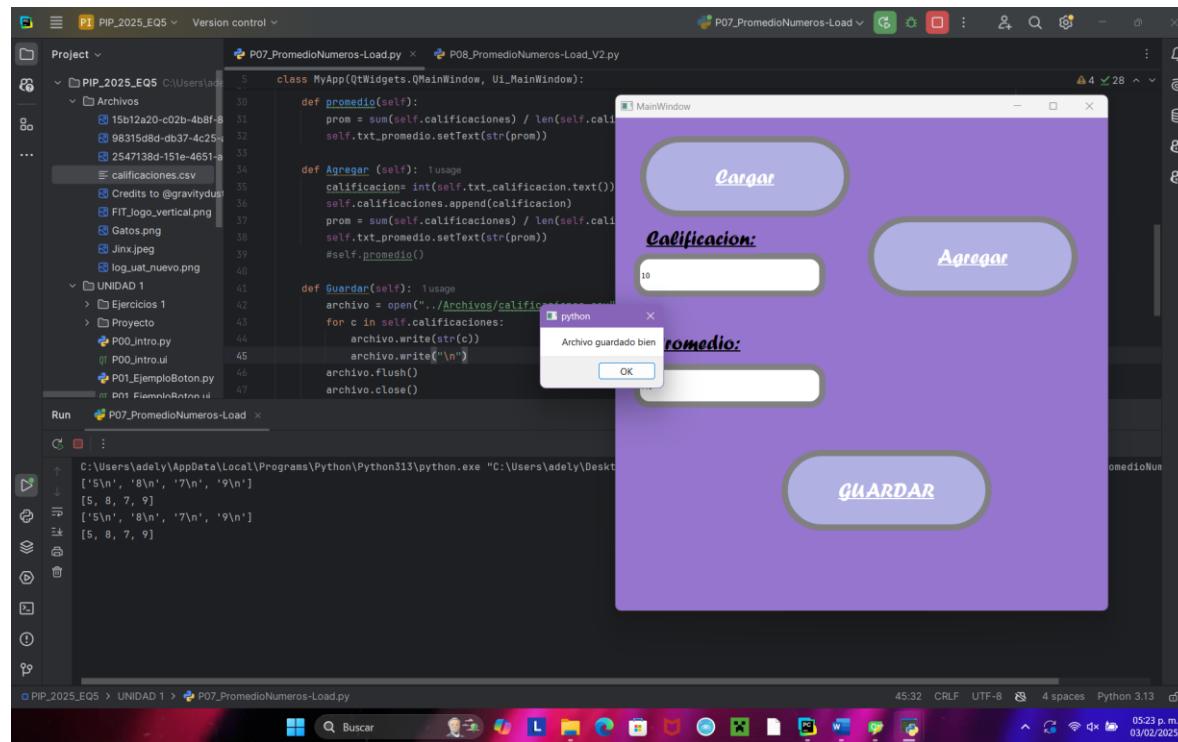


```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

    def promedio(self):
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))

    def Agregar(self):
        calificacion = int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones) / len(self.calificaciones)
        self.txt_promedio.setText(str(prom))
        self.promedio()

    def Guardar(self):
        archivo = open("./Archivos/calificaciones.csv", "w")
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
```



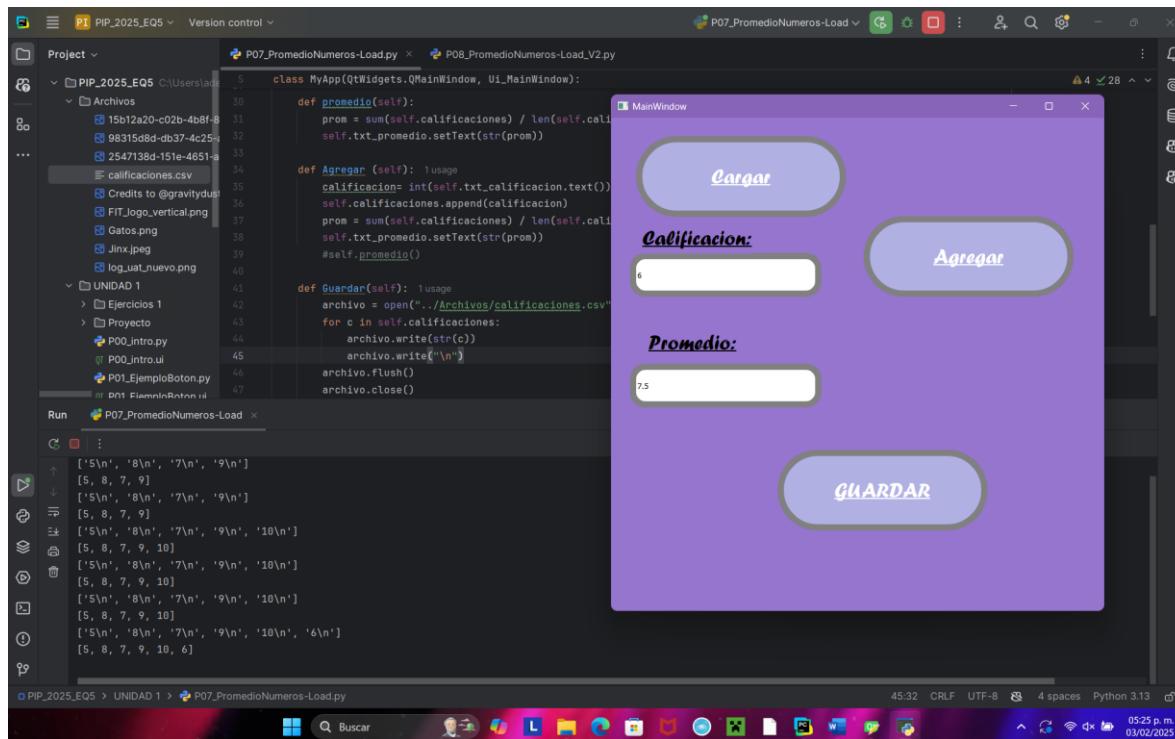
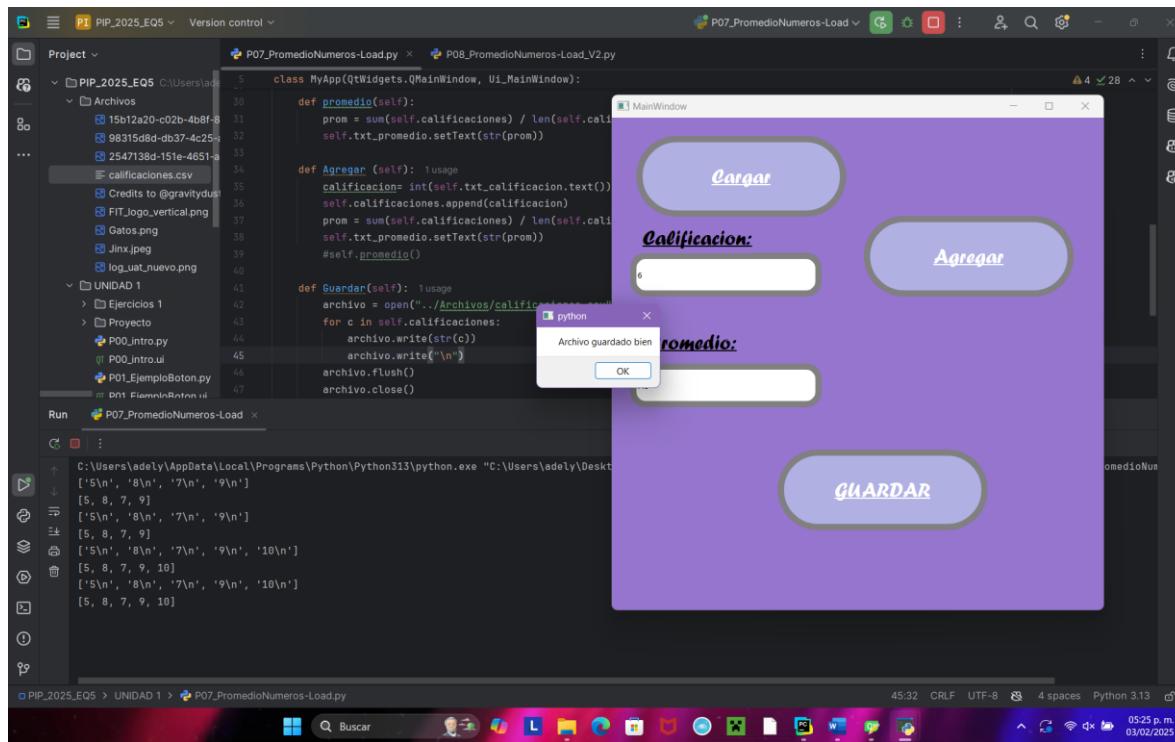


The screenshot shows the PyCharm IDE interface with the project structure and code editor. The code implements a simple application for calculating the average of numbers from a CSV file. The application window displays three text input fields: 'Calificación:' with value '6', 'Promedio:' with value '7.5', and a 'GUARDAR' button.

```
class MyApp(QtWidgets.QMainWindow):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle("Promedio Numeros")  
        self.setGeometry(300, 200, 400, 200)  
        self.txt_calificacion = QtWidgets.QLineEdit(self)  
        self.txt_promedio = QtWidgets.QLineEdit(self)  
        self.txt_guardar = QtWidgets.QPushButton("GUARDAR", self)  
        self.txt_cargar = QtWidgets.QPushButton("Cargar", self)  
        self.txt_agregar = QtWidgets.QPushButton("Agregar", self)  
  
        self.txt_promedio.setPlaceholderText("Promedio:")  
        self.txt_calificacion.setPlaceholderText("Calificación:")  
  
        self.txt_cargar.clicked.connect(self.cargar)  
        self.txt_agregar.clicked.connect(self.agregar)  
        self.txt_guardar.clicked.connect(self.guardar)  
  
    def cargar(self):  
        calificaciones = int(self.txt_calificacion.text())  
        self.txt_promedio.setText(str(calificaciones))  
  
    def agregar(self):  
        calificaciones = int(self.txt_calificacion.text())  
        self.txt_promedio.setText(str(calificaciones))  
        prom = sum(self.txt_promedio.text()) / len(self.txt_promedio.text())  
        self.txt_promedio.setText(str(prom))  
  
    def guardar(self):  
        archivo = open("./Archivos/calificaciones.csv", "w")  
        for c in self.txt_promedio.text():  
            archivo.write(str(c))  
            archivo.write("\n")  
        archivo.flush()  
        archivo.close()  
  
if __name__ == "__main__":  
    app = QtWidgets.QApplication(sys.argv)  
    window = MyApp()  
    window.show()  
    sys.exit(app.exec_())
```

The screenshot shows the PyCharm IDE interface with the project structure and code editor. The code is identical to the one in the previous screenshot, implementing a simple application for calculating the average of numbers from a CSV file. The application window displays three text input fields: 'Calificación:' with value '6', 'Promedio:' with value '7.5', and a 'GUARDAR' button.

```
class MyApp(QtWidgets.QMainWindow):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle("Promedio Numeros")  
        self.setGeometry(300, 200, 400, 200)  
        self.txt_calificacion = QtWidgets.QLineEdit(self)  
        self.txt_promedio = QtWidgets.QLineEdit(self)  
        self.txt_guardar = QtWidgets.QPushButton("GUARDAR", self)  
        self.txt_cargar = QtWidgets.QPushButton("Cargar", self)  
        self.txt_agregar = QtWidgets.QPushButton("Agregar", self)  
  
        self.txt_promedio.setPlaceholderText("Promedio:")  
        self.txt_calificacion.setPlaceholderText("Calificación:")  
  
        self.txt_cargar.clicked.connect(self.cargar)  
        self.txt_agregar.clicked.connect(self.agregar)  
        self.txt_guardar.clicked.connect(self.guardar)  
  
    def cargar(self):  
        calificaciones = int(self.txt_calificacion.text())  
        self.txt_promedio.setText(str(calificaciones))  
  
    def agregar(self):  
        calificaciones = int(self.txt_calificacion.text())  
        self.txt_promedio.setText(str(calificaciones))  
        prom = sum(self.txt_promedio.text()) / len(self.txt_promedio.text())  
        self.txt_promedio.setText(str(prom))  
  
    def guardar(self):  
        archivo = open("./Archivos/calificaciones.csv", "w")  
        for c in self.txt_promedio.text():  
            archivo.write(str(c))  
            archivo.write("\n")  
        archivo.flush()  
        archivo.close()  
  
if __name__ == "__main__":  
    app = QtWidgets.QApplication(sys.argv)  
    window = MyApp()  
    window.show()  
    sys.exit(app.exec_())
```



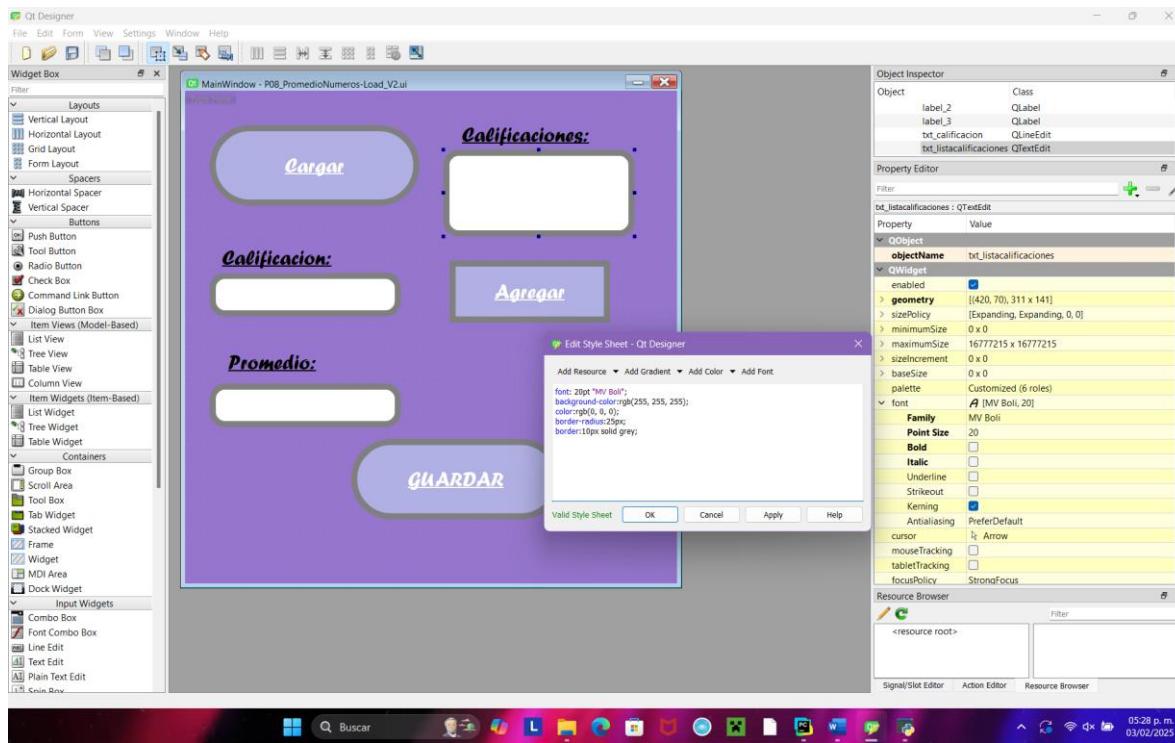


The screenshot shows the PyCharm IDE interface. The project structure on the left includes a folder 'PIP\_2025\_EQS' containing 'Archivos' and 'UNIDAD 1' subfolders. Inside 'Archivos' are several files: 'calificaciones.csv', 'Credits to @gravityduo', 'FIT\_Logo\_vertical.png', 'Gatos.png', 'Jinx.jpeg', and 'log\_uat\_nuevo.png'. The main code editor window displays the script 'P08\_PromedioNumeros\_Load\_V2.py'. The code reads data from 'calificaciones.csv' and prints it to the console. The output in the terminal shows the following data:

```
[5\n', '8\n', '7\n', '9\n']\n[5, 8, 7, 9]\n['5\n', '8\n', '7\n', '9\n']\n[5, 8, 7, 9]\n['5\n', '8\n', '7\n', '9\n', '10\n']\n[5, 8, 7, 9, 10]\n['5\n', '8\n', '7\n', '9\n', '10\n']\n[5, 8, 7, 9, 10]\n['5\n', '8\n', '7\n', '9\n', '10\n']\n[5, 8, 7, 9, 10]\n['5\n', '8\n', '7\n', '9\n', '10\n', '6\n']\n[5, 8, 7, 9, 10, 6]
```



## C9. P08\_PromedioNumeros-Load\_V2



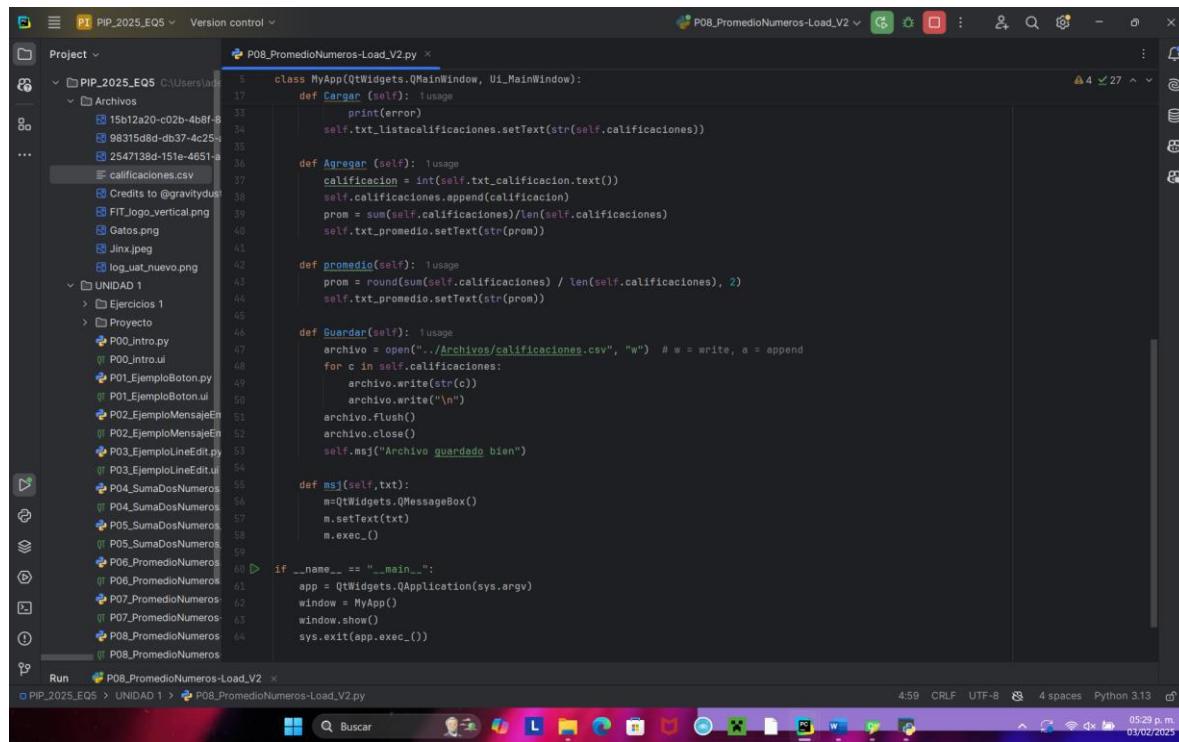
```
import sys
from PyQt5 import uic, QtWidgets
qtCreatorFile = "P08_PromedioNumeros-Load_V2.ui" # Nombre del archivo ".ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        self.btn_Cargar.clicked.connect(self.Cargar)
        self.btn_Agregar.clicked.connect(self.Agregar)
        self.btn_Guardar.clicked.connect(self.Guardar)
        self.calificaciones = []

    #Área de los Slots
    def Cargar(self):
        #Tarea como comproue si el archivo existe
        #ejercicio 10
        # tarea ej 11 en lugar de sobre escribir, concatenar
        # tarea ej 12 asegurarse de que solo se pueda cargar hasta antes de
        # agregar la primera calificación enables y/o código
        archivo = open("./archivos/calificaciones.csv")
        contenido = archivo.readlines()
        print(contenido)
        datos = [int(x) for x in contenido]
        print(datos)
        self.calificaciones = datos
        self.promedio()

    try:
        self.txt_listacalificaciones.setText(str(self.calificaciones))
    except Exception as error:
        print(error)
        self.txt_listacalificaciones.setText(str(self.calificaciones))

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    app.exec_()
```



```
Project: PIP_2025_EQ5 Version control: P08_PromedioNumeros-Load_V2.py

class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def Cargar(self): Usage
        print(error)
        self.txt_listacalificaciones.setText(str(self.calificaciones))

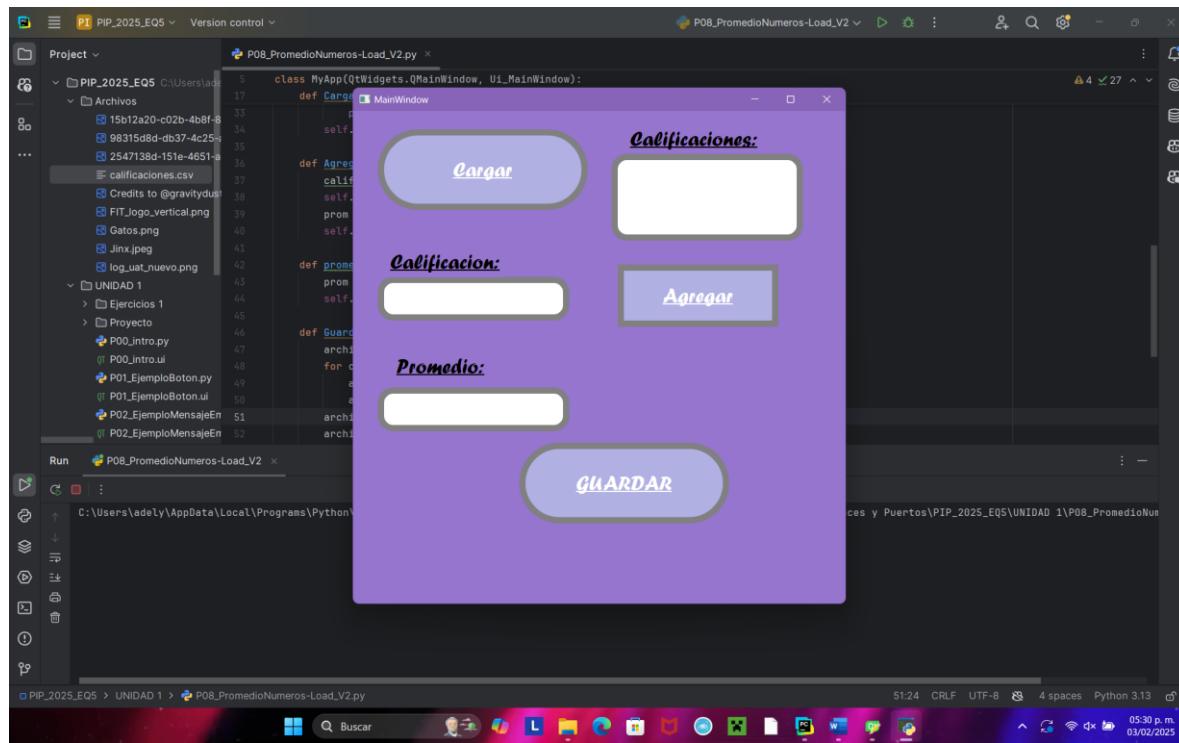
    def Agregar(self): Usage
        calificacion = int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones)/len(self.calificaciones)
        self.txt_promedio.setText(str(prom))

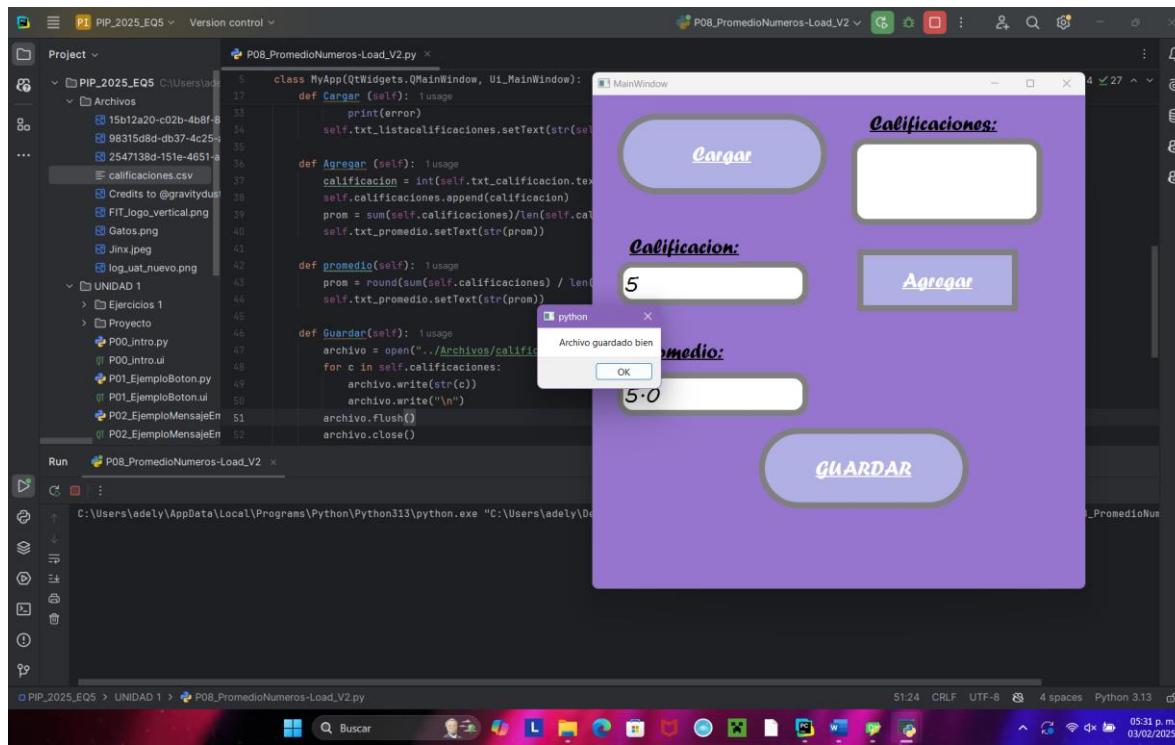
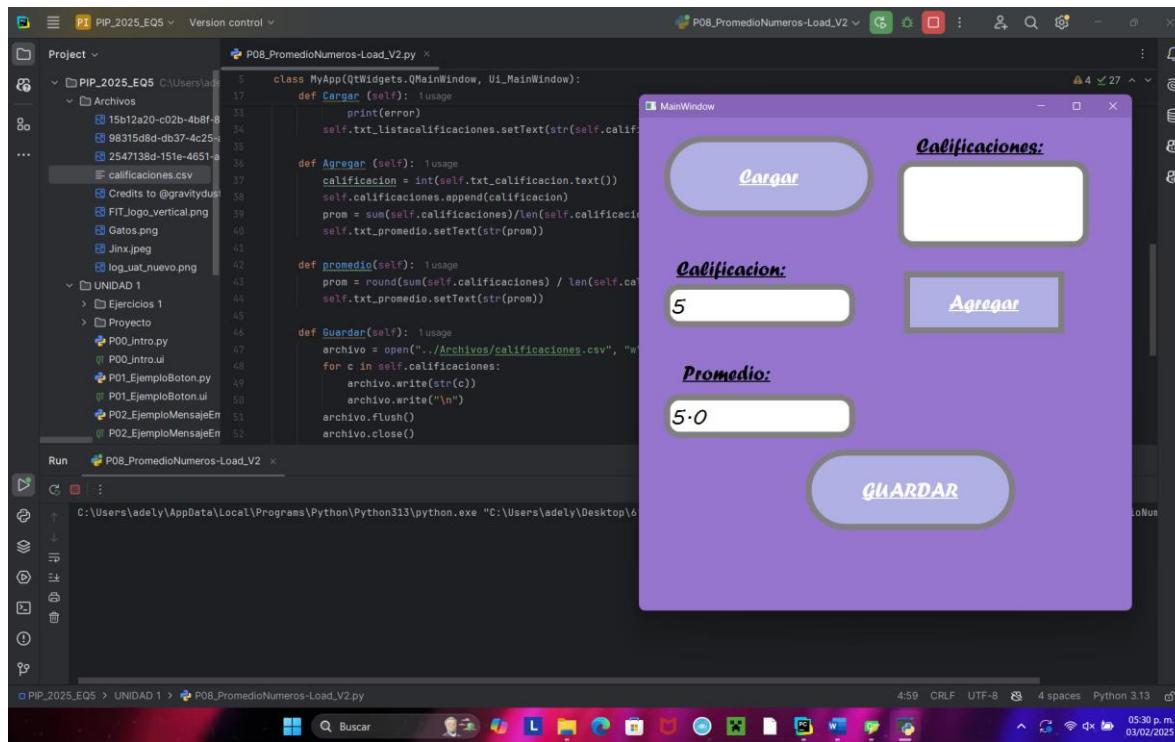
    def promedio(self): Usage
        prom = round(sum(self.calificaciones) / len(self.calificaciones), 2)
        self.txt_promedio.setText(str(prom))

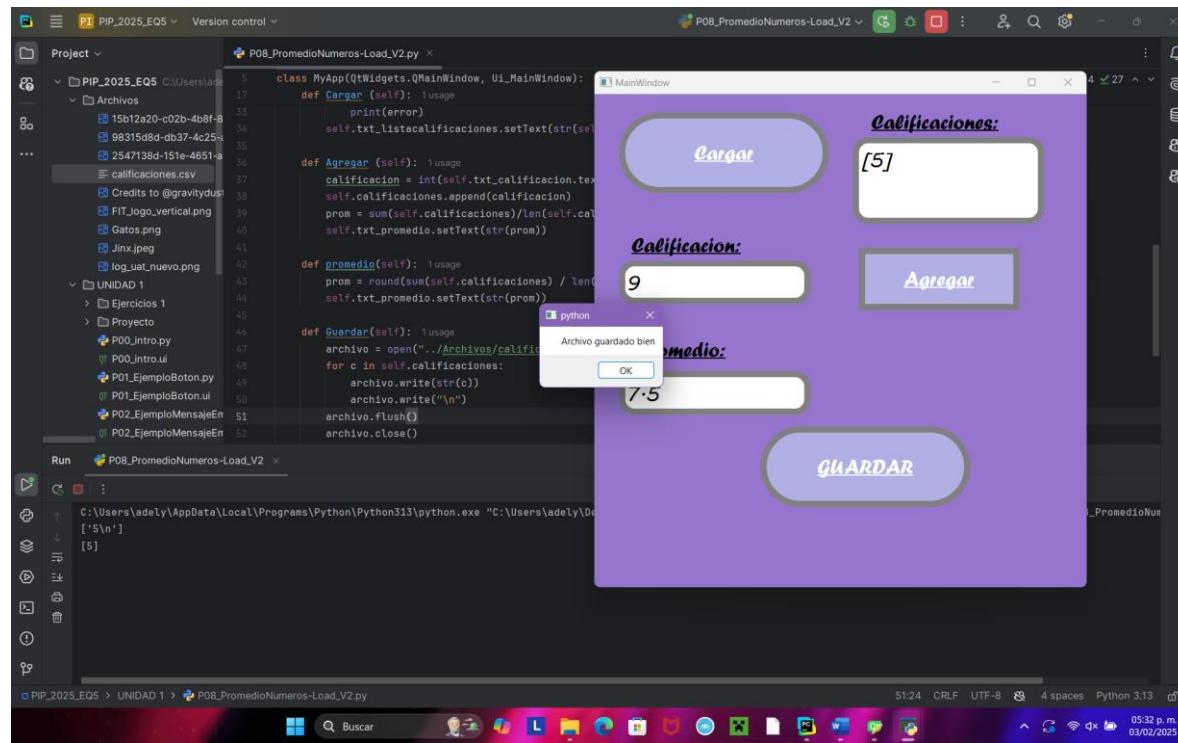
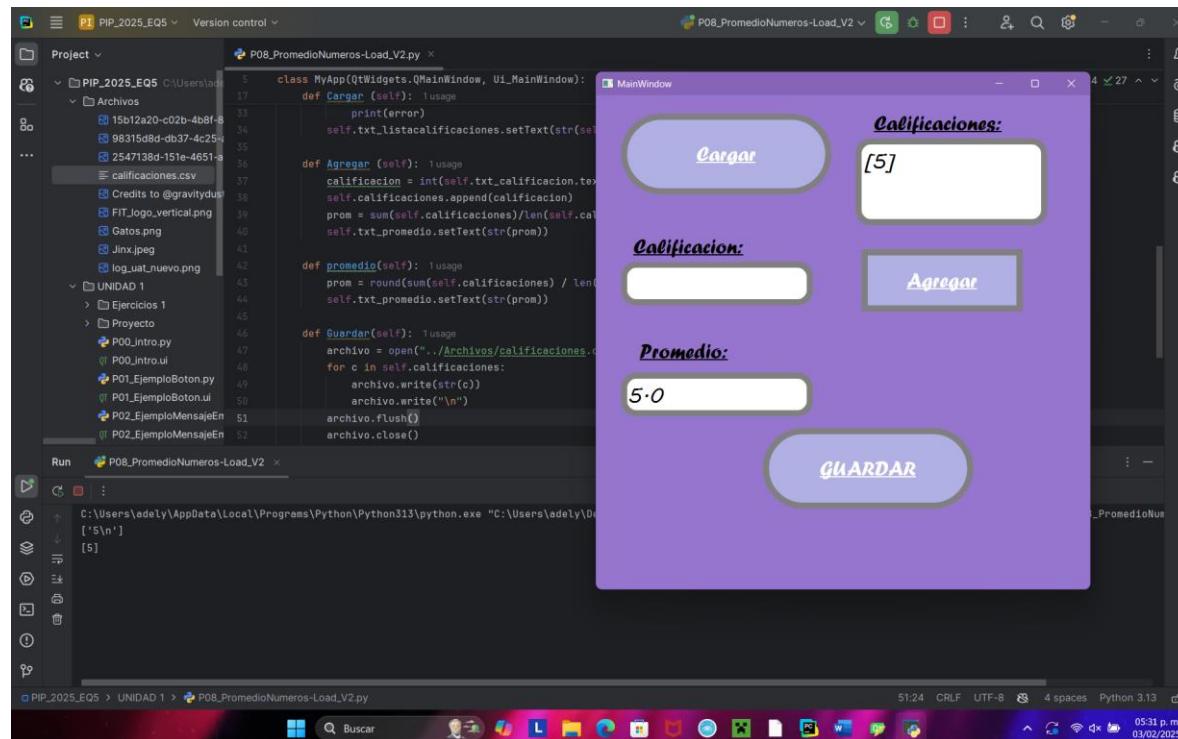
    def Guardar(self): Usage
        archivo = open("./Archivos/calificaciones.csv", "w") # w = write, a = append
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo guardado bien")

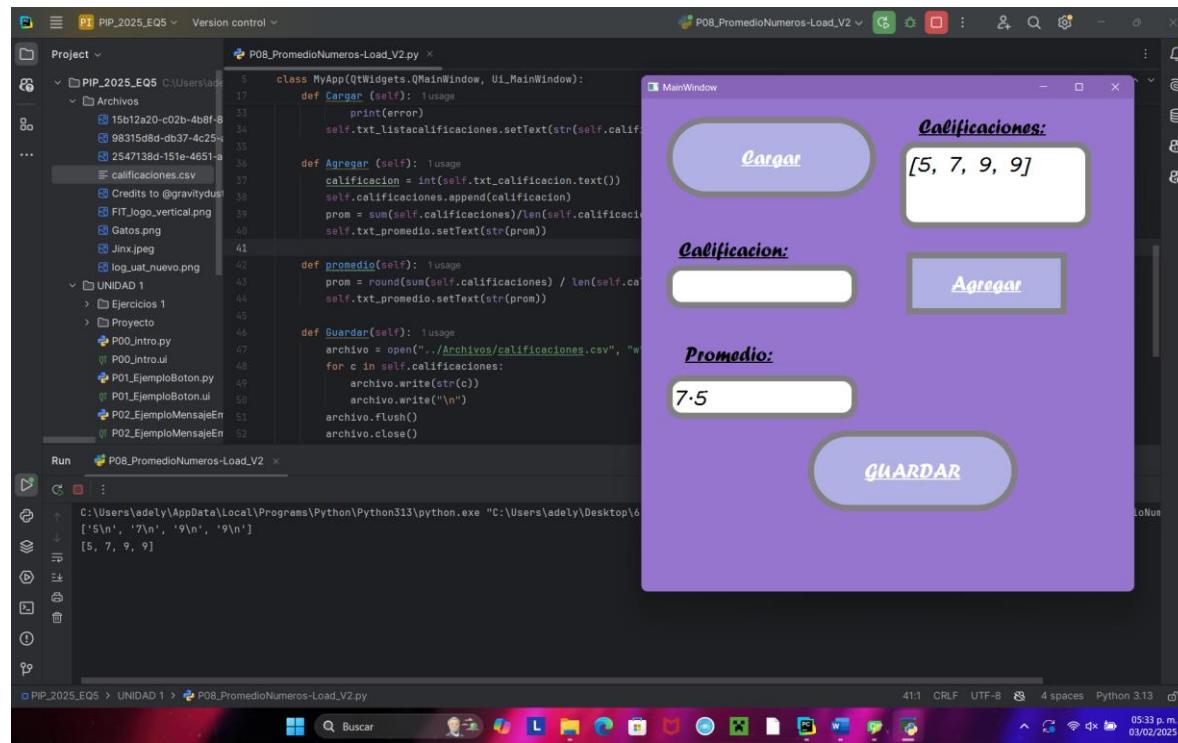
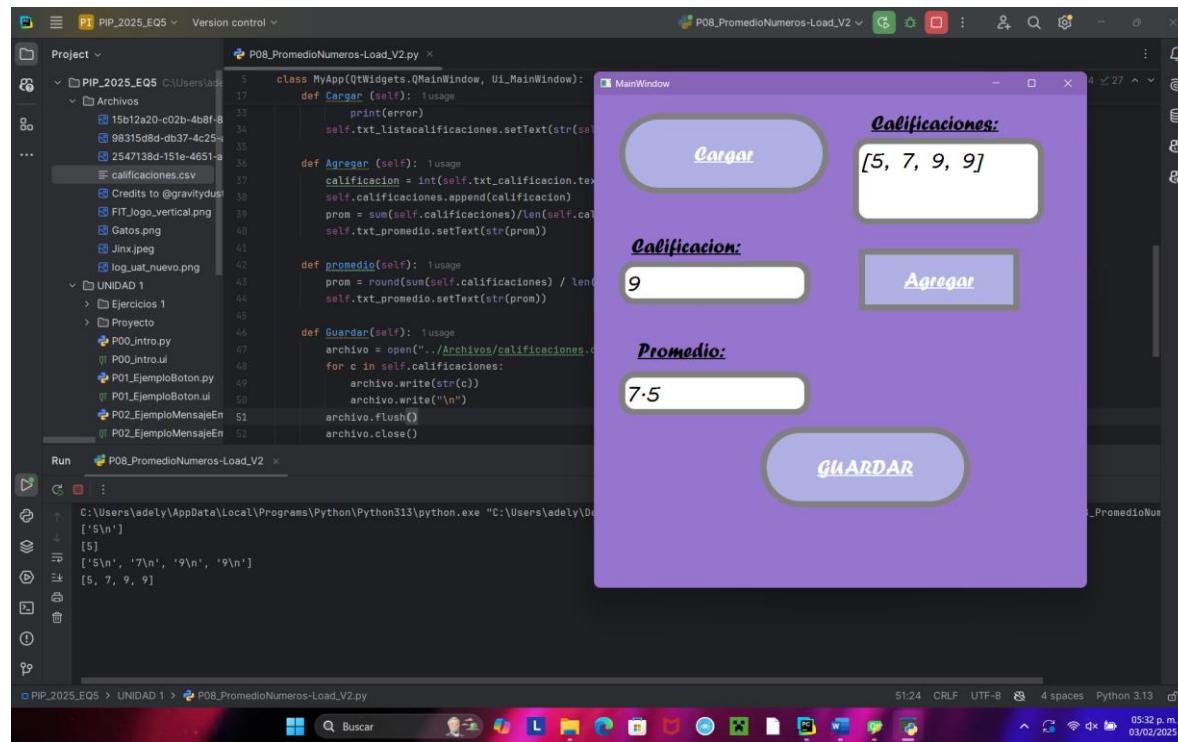
    def msj(self,txt):
        m=QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```











The screenshot shows the PyCharm IDE interface. On the left is the project tree for 'PIP\_2025\_EQ5' containing files like 'calificaciones.csv', 'P01\_EjemploBoton.py', and 'P02\_EjemploMensajeEn.py'. The main code editor shows the 'P08\_PromedioNumeros-Load\_V2.py' file with Python code for a GUI application. The right side shows a running application window titled 'MainWindow'. The window has three text input fields: 'Calificaciones:' containing '[5, 7, 9, 9]', 'Calificación:' containing '10', and 'Promedio:' containing '8.0'. It also has a 'Cargar' button, an 'Agregar' button, and a 'GUARDAR' button.

```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.txt_listacalificaciones.setText(str(self.calificaciones))
        self.txt_promedio.setText(str(prom))

    def Cargar(self):
        print(error)
        self.txt_listacalificaciones.setText(str(self.calificaciones))

    def Agregar(self):
        calificacion = int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones)/len(self.calificaciones)
        self.txt_promedio.setText(str(prom))

    def promedio(self):
        prom = round(sum(self.calificaciones) / len(self.calificaciones))
        self.txt_promedio.setText(str(prom))

    def Guardar(self):
        archivo = open("../Archivos/calificaciones.csv", "w")
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
```

The screenshot shows the PyCharm IDE interface. On the left is the project tree for 'PIP\_2025\_EQ5' containing files like 'calificaciones.csv', 'P01\_EjemploBoton.py', and 'P02\_EjemploMensajeEn.py'. The main code editor shows the 'P08\_PromedioNumeros-Load\_V2.py' file with Python code for a GUI application. The right side shows a running application window titled 'MainWindow'. The window has three text input fields: 'Calificaciones:' containing '[5, 7, 9, 9]', 'Calificación:' containing '10', and 'Promedio:' containing '7.5'. It also has a 'Cargar' button, an 'Agregar' button, and a 'GUARDAR' button.

```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.txt_listacalificaciones.setText(str(self.calificaciones))
        self.txt_promedio.setText(str(prom))

    def Cargar(self):
        print(error)
        self.txt_listacalificaciones.setText(str(self.calificaciones))

    def Agregar(self):
        calificacion = int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones)/len(self.calificaciones)
        self.txt_promedio.setText(str(prom))

    def promedio(self):
        prom = round(sum(self.calificaciones) / len(self.calificaciones))
        self.txt_promedio.setText(str(prom))

    def Guardar(self):
        archivo = open("../Archivos/calificaciones.csv", "w")
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
```



The screenshot shows the PyCharm IDE interface with the project **PIP\_2025\_EQ5** open. The code editor displays the **P08\_PromedioNumeros-Load\_V2.py** file. The application window titled **MainWindow** is displayed, showing a user interface with three buttons: **Cargar**, **Agregar**, and **GUARDAR**. The **Cargar** button has the value **[5, 7, 9, 9]**. The **Agregar** button has the value **10**. The **Promedio:** label has the value **8.0**. A small dialog box titled **python** shows the message **Archivo guardado bien** with an **OK** button.

```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def Cargar(self): usage
        print(error)
        self.txt_listacalificaciones.setText(str(self.calificaciones))

    def Agregar(self): usage
        calificacion = int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones)/len(self.calificaciones)
        self.txt_promedio.setText(str(prom))

    def promedio(self): usage
        prom = round(sum(self.calificaciones) / len(self.calificaciones))
        self.txt_promedio.setText(str(prom))

    def Guardar(self): usage
        archivo = open("./Archivos/calificaciones.csv", "w")
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
```

The screenshot shows the PyCharm IDE interface with the project **PIP\_2025\_EQ5** open. The code editor displays the **P08\_PromedioNumeros-Load\_V2.py** file. The application window titled **MainWindow** is displayed, showing a user interface with three buttons: **Cargar**, **Agregar**, and **GUARDAR**. The **Cargar** button has the value **[5, 7, 9, 9, 10]**. The **Agregar** button has the value **10**. The **Promedio:** label has the value **8.0**.

```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def Cargar(self): usage
        print(error)
        self.txt_listacalificaciones.setText(str(self.calificaciones))

    def Agregar(self): usage
        calificacion = int(self.txt_calificacion.text())
        self.calificaciones.append(calificacion)
        prom = sum(self.calificaciones)/len(self.calificaciones)
        self.txt_promedio.setText(str(prom))

    def promedio(self): usage
        prom = round(sum(self.calificaciones) / len(self.calificaciones))
        self.txt_promedio.setText(str(prom))

    def Guardar(self): usage
        archivo = open("./Archivos/calificaciones.csv", "w")
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
```

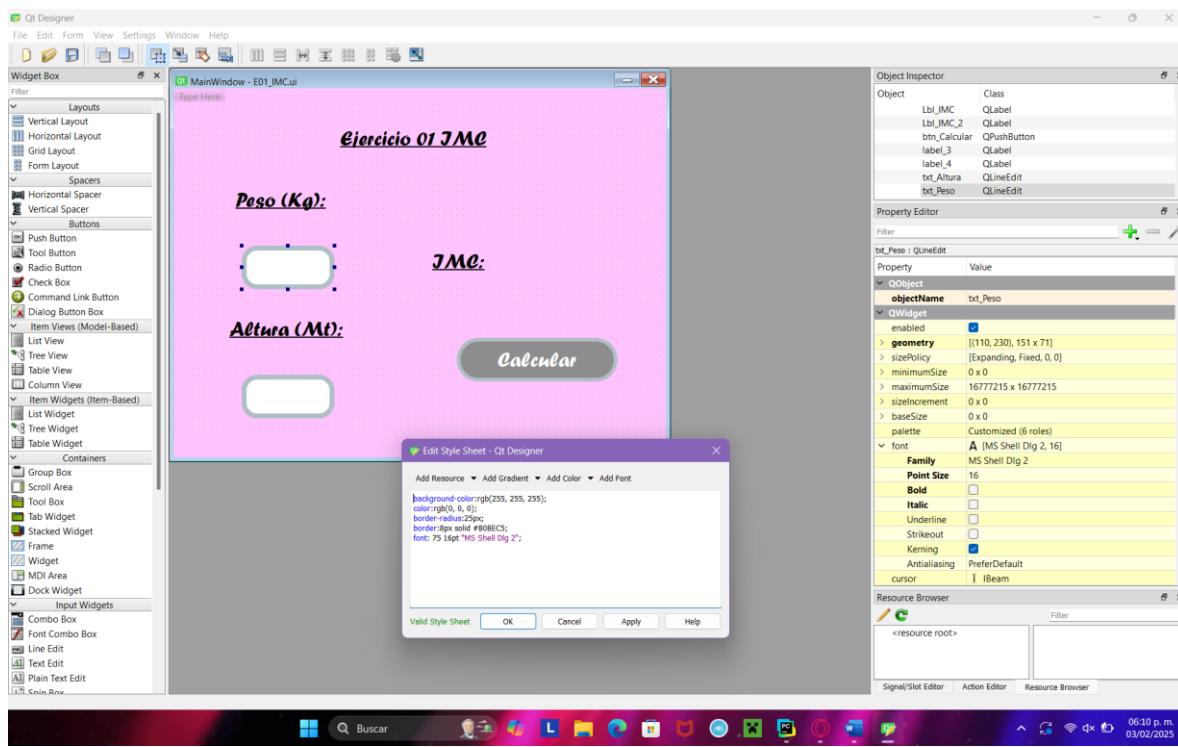
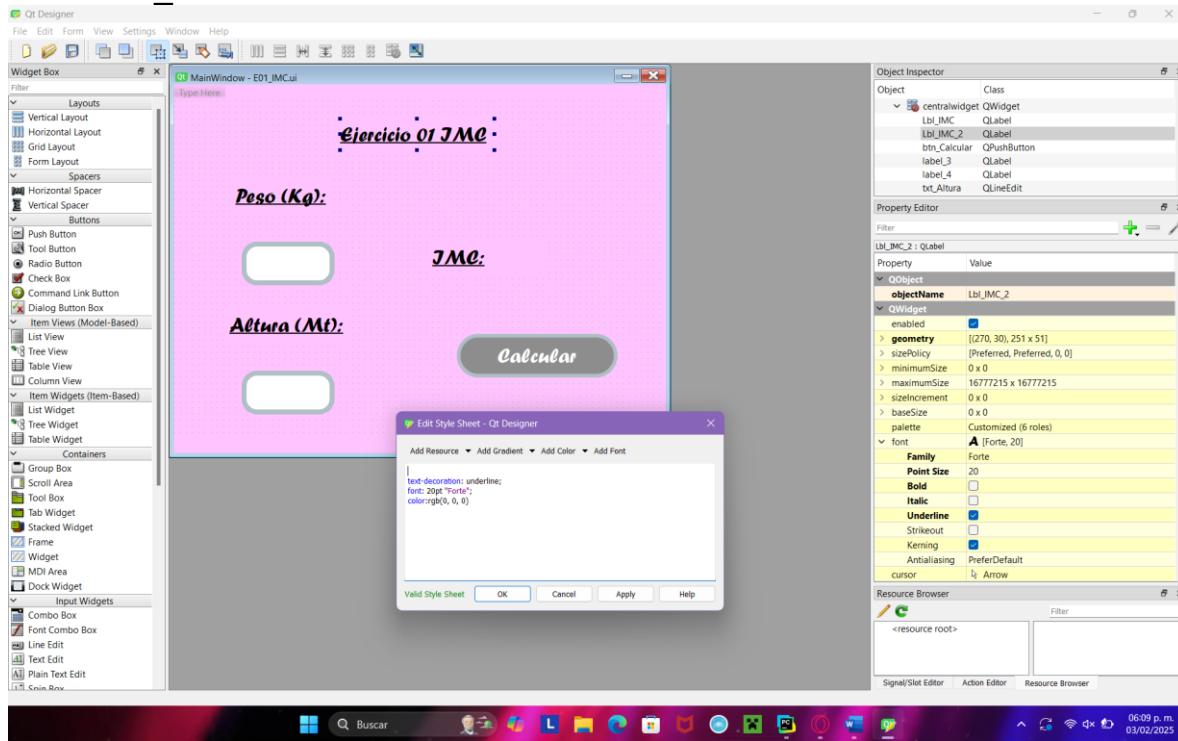


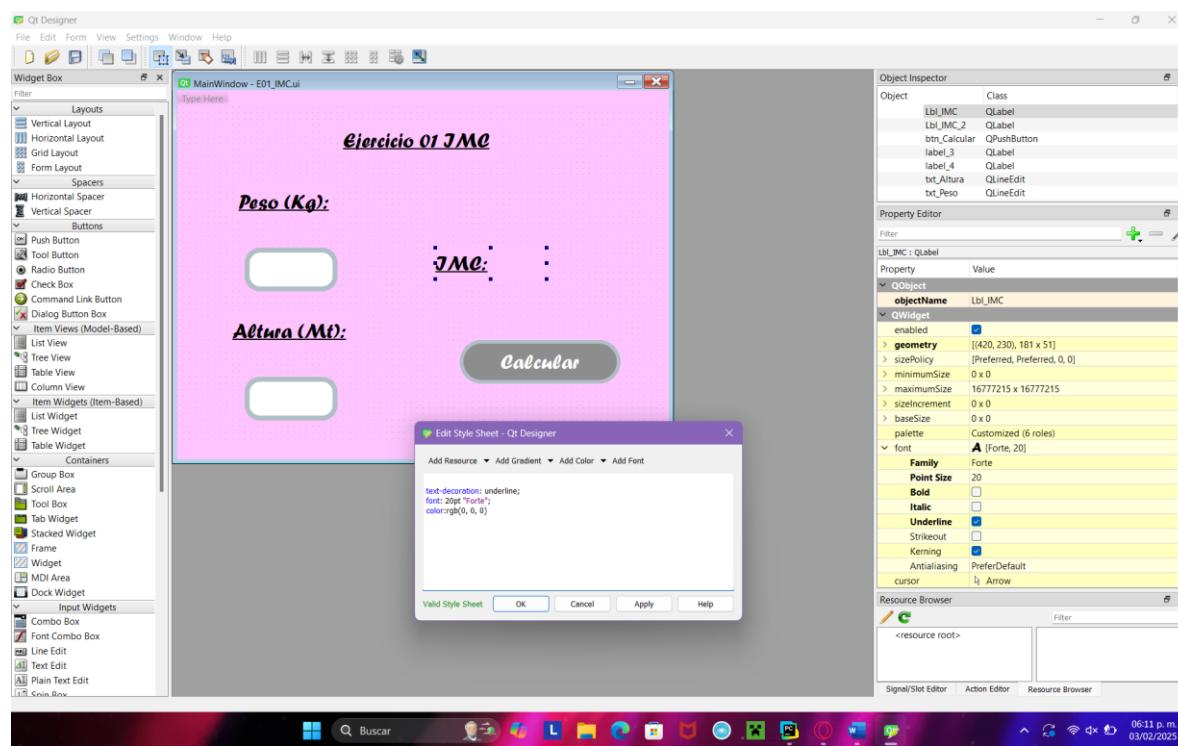
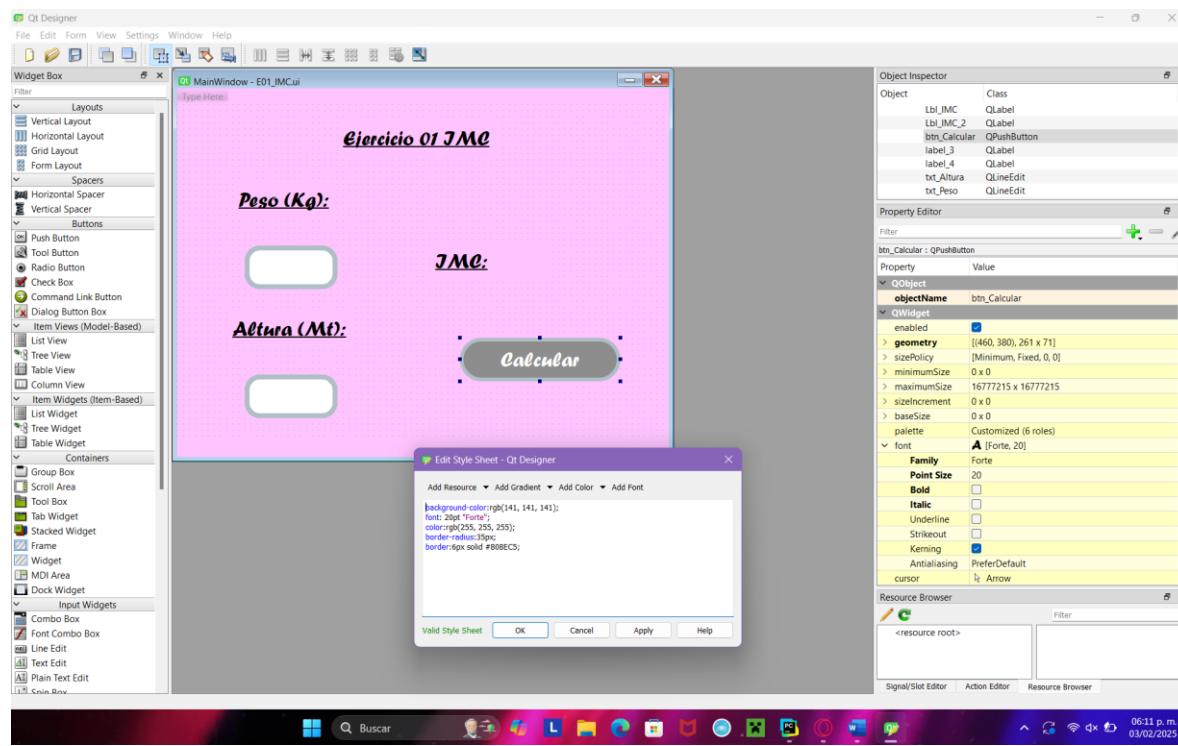
The screenshot shows the PyCharm IDE interface. The project structure on the left includes a folder 'PIP\_2025\_EQ5' containing 'Archivos' and 'UNIDAD 1'. Inside 'Archivos' are several files: 'calificaciones.csv', 'Credits to egravitydus', 'FIT\_Logo\_vertical.png', 'Gatos.png', 'Jinx.jpeg', and 'log\_uat\_nuevo.png'. The 'UNIDAD 1' folder contains 'Ejercicios 1' and 'Proyecto', which includes files 'P00\_intro.py', 'P00\_intro.ui', 'P01\_EjemploBoton.py', 'P01\_EjemploBoton.ui', 'P02\_EjemploMensajeError.py', and 'P02\_EjemploMensajeError.ui'. The main code editor window displays the script 'P08\_PromedioNumeros\_Load\_V2.py'. The code reads data from 'calificaciones.csv' and prints it to the terminal. The terminal output shows the following data:

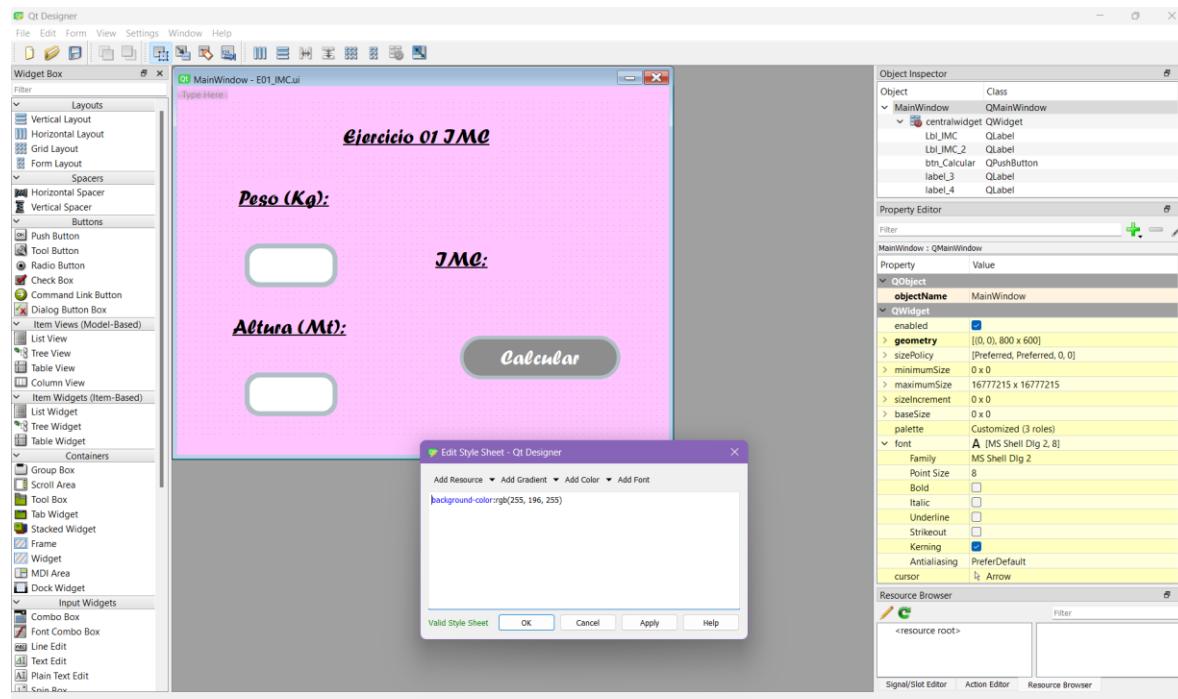
```
C:\Users\adely\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\adely\Desktop\6º semestre\Programacion de Interfaces y Puertos\PIP_2025_EQ5\UNIDAD 1\P08_PromedioNumeros_Load_V2.py"
[5\n', '7\n', '9\n', '9\n']
[5, 7, 9, 9]
['5\n', '7\n', '9\n', '9\n']
[5, 7, 9, 9]
['5\n', '7\n', '9\n', '9\n', '10\n']
[5, 7, 9, 9, 10]
```



## C10. E01\_IMC







Por la parte de diseño visual, tenemos cuatro labels: Altura, Peso, IMC, y el título del ejercicio. Constituidos también por dos entradas de texto, que le proporcionarán al usuario tener la libertad de ingresar cual fuese el valor para calcular su IMC. Añadiendo también que contamos con un botón de calcular, este, al accionarlo, hará que se ejecute el proceso lógico, para que al momento de tener el resultado, se vea reflejado en el label (IMC) y aparezca el valor a un lado de este.



The screenshot shows the PyCharm IDE interface with a project named 'PIP\_2025\_1\_eq5-main'. The code editor displays a file named 'E01\_IMC.py' which contains Python code for a Qt application. The code includes imports for sys, PyQt5, and QtCore, and defines a class 'MyApp' that inherits from QMainWindow. It contains methods for initializing the window, setting up UI components, and calculating the IMC value based on weight and height inputs. A try-except block handles potential errors like non-numeric input. The code also includes a 'changeStyleSheet' function for styling labels.

```
import sys
from PyQt5 import uic, QtWidgets, QtCore
qtCreatorFile = "E01_IMC.ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        #para agregar la funcionalidad de calcular el boton de calcular
        self.btn_Calcular.clicked.connect(self.calcular)
        self.txt_Peso.textChanged.connect(self.CheckLetter)
        self.txt_Altura.textChanged.connect(self.CheckLetterA)

    def calcular(self):
        try:
            peso = float(self.txt_Peso.text())
            altura = float(self.txt_Altura.text())
        except:
            #por si dejan espacios, o ponen letras
            self.es("Por favor, rellene ambos campos con valores numericos")
            return
        IMC = round(peso / (altura * altura),2)
        self.lbl_IMC.setText("IMC: "+str(IMC))
        self.setStyleSheet(self.lbl_IMC)

    def changeStyleSheet(self, label): #cambia el stylesheet de un label
        label.setStyleSheet("""
        QLabel {
            text-decoration: underline;
            font: 20pt "Forte";
            color:rgb(0, 0, 0);
        }
        """)
```

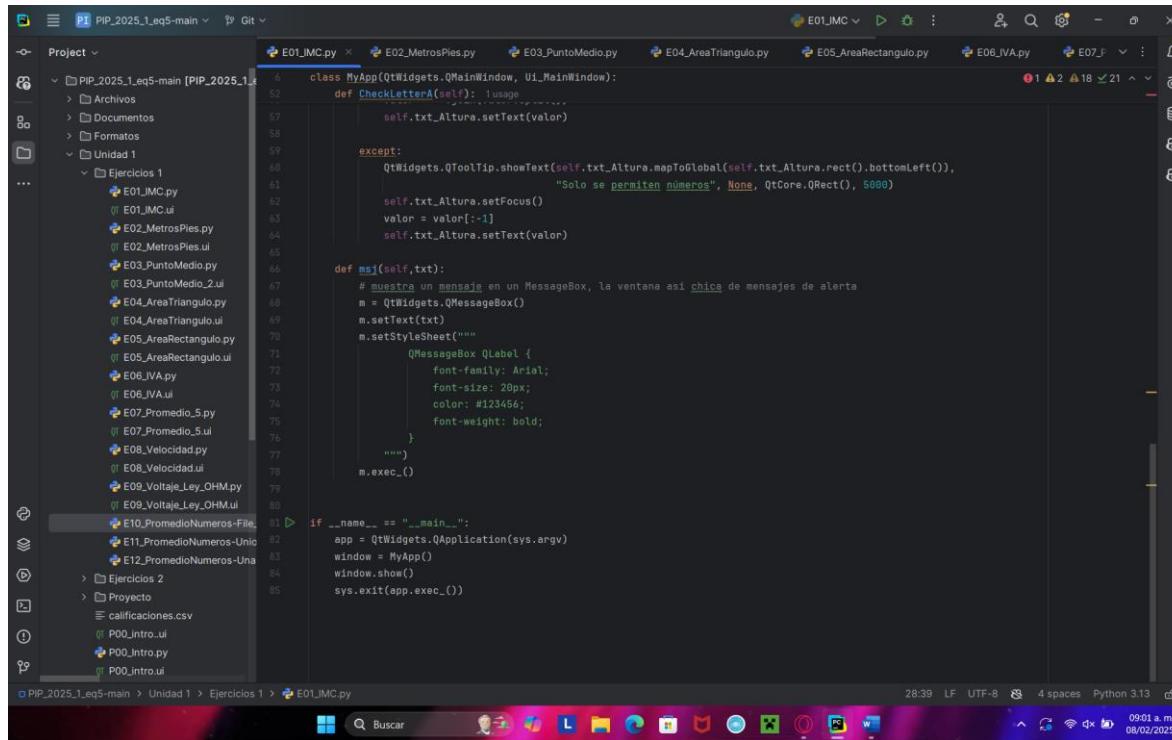
En el código, tenemos el “esqueleto” de todo programa visual hecho con QtDesigner implementado en Python. Primeramente, contamos con los signals, estos se encargan de recibir la señal proveniente de nuestro diseño en ejecución. Por otro lado, contamos con las primeras dos funciones. La función *calcular* emplea toda la lógica para que se haga el procedimiento al obtener ambos valores de los TextField. La función *changeStyleSheet* nos ayudará a agregarle una interacción visual más atractivo en cuanto se obtenga el resultado y al mostrarlo en nuestra aplicación visual.



The screenshot shows the PyCharm IDE interface with the following details:

- Project:** PIP\_2025\_1\_eq5-main
- File:** E01\_JMC.py
- Code Preview:** The code defines a class `MyApp` that inherits from `QtWidgets.QMainWindow` and `Ui_MainWindow`. It contains methods for validating input fields (e.g., `CheckLetter`, `CheckLetterA`) and displaying messages (e.g., `msj`). The code uses PyQt5 and includes comments explaining its functionality.
- Toolbars and Status Bar:** The status bar at the bottom shows the time as 09:01 a.m. and the date as 08/02/2025.

Tenemos dos funciones que pueden parecer lo mismo, pero no son para los mismos objetos. *ChekLetter* nos ayudará a que solamente se permitan números enteros o de punto flotante, ya que, esto es importante porque el usuario también puede cometer el error de ingresar una letra. Por consiguiente, el programa se cerraría debido a un error de dato inválido, y esta función está directamente asignada al TextField del peso. *ChekLetterA* hace exactamente lo mismo pero con la otra entrada de datos mediante el TextField de la altura, y ambas cuentan con una funcionalidad que permite mostrar un mensaje chico que indica al usuario que solamente se permiten números.



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def CheckLetterA(self):
        valor = self.txt_Altura.text()
        try:
            float(valor)
            self.txt_Altura.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_Altura.mapToGlobal(self.txt_Altura.rect().bottomLeft()), "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_Altura.setFocus()
            valor = valor[:-1]
            self.txt_Altura.setText(valor)

    def msj(self,txt):
        # muestra un mensaje en un QMessageBox, la ventana así chica de mensajes de alerta
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.setStyleSheet('''
            QMessageBox QLabel {
                font-family: Arial;
                font-size: 20px;
                color: #123456;
                font-weight: bold;
            }
        ''')
        m.exec_()

    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```

Tenemos la función `msj`, esta es solamente un MessageBox, un tipo de mensaje de alerta, y el main donde vamos a ejecutar nuestro programa visual.

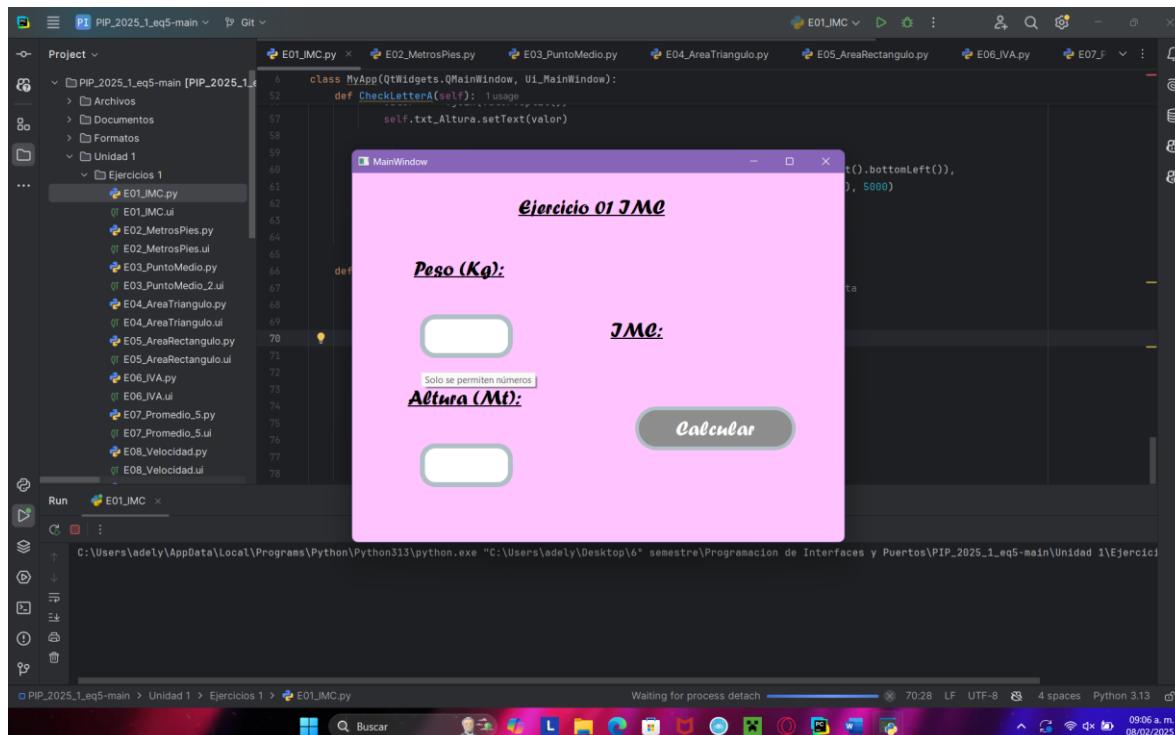


```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def CheckLetterA(self):
        valor = self.txt_Altura.text()
        try:
            float(valor)
            self.txt_Altura.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_Altura.mapToGlobal(self.txt_Altura.rect().bottomLeft()), "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_Altura.setFocus()
            valor = valor[:-1]
            self.txt_Altura.setText(valor)

    def msj(self,txt):
        # muestra un mensaje en un QMessageBox, la ventana así chica de mensajes de alerta
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.setStyleSheet('''
            QMessageBox QLabel {
                font-family: Arial;
                font-size: 20px;
                color: #123456;
                font-weight: bold;
            }
        ''')
        m.exec_()

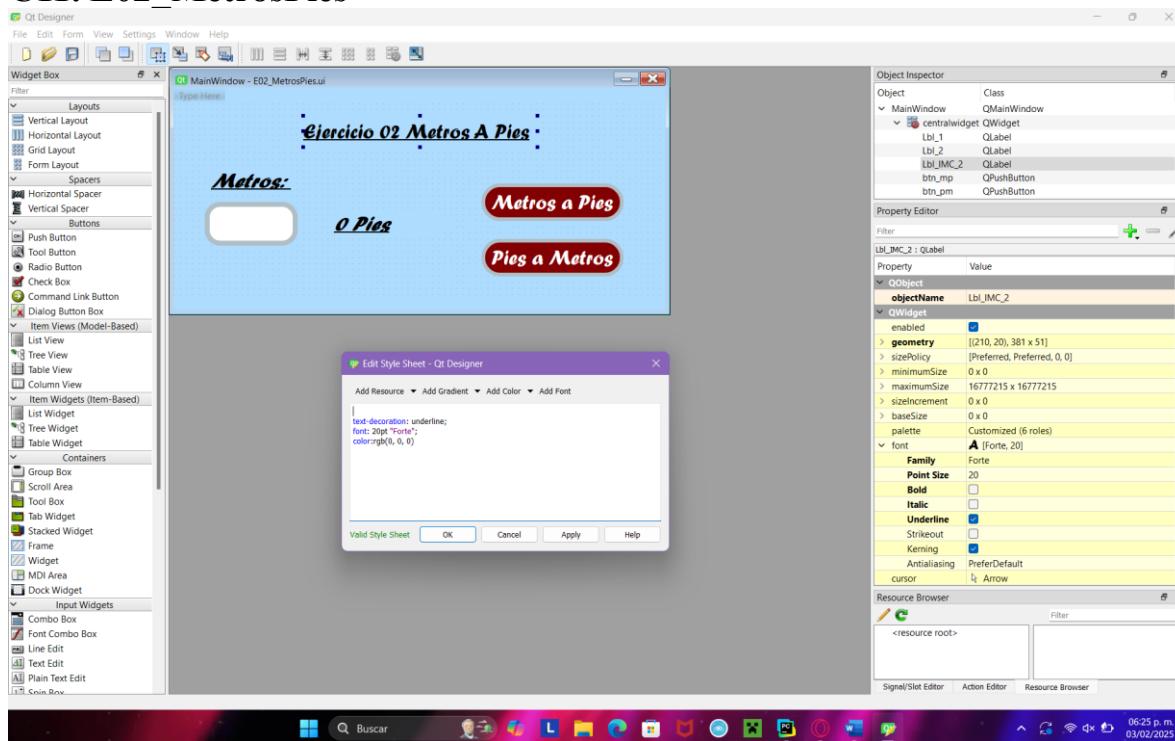
    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```

Ingresamos los datos y nos muestra el resultado al accionar el botón “calcular”.



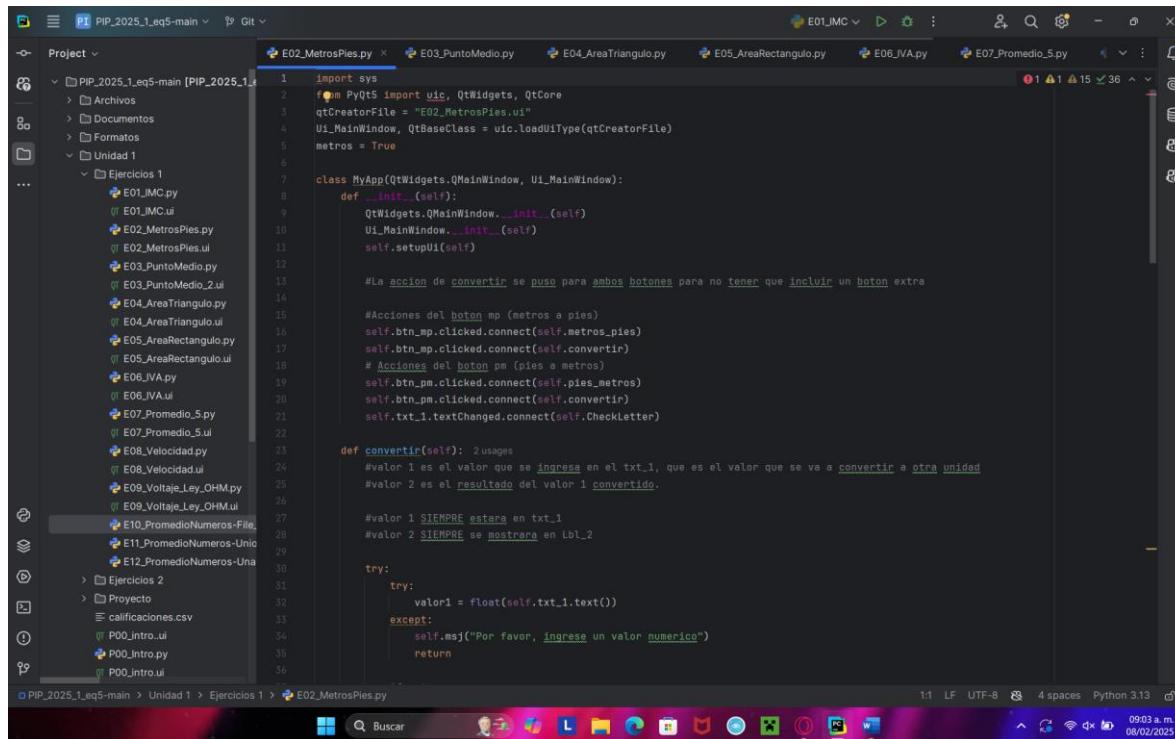
Si queremos ingresar letras o caracteres, el programa no nos dejará y seguirá en ejecución hasta que ingresemos números.

## C11. E02\_MetrosPies



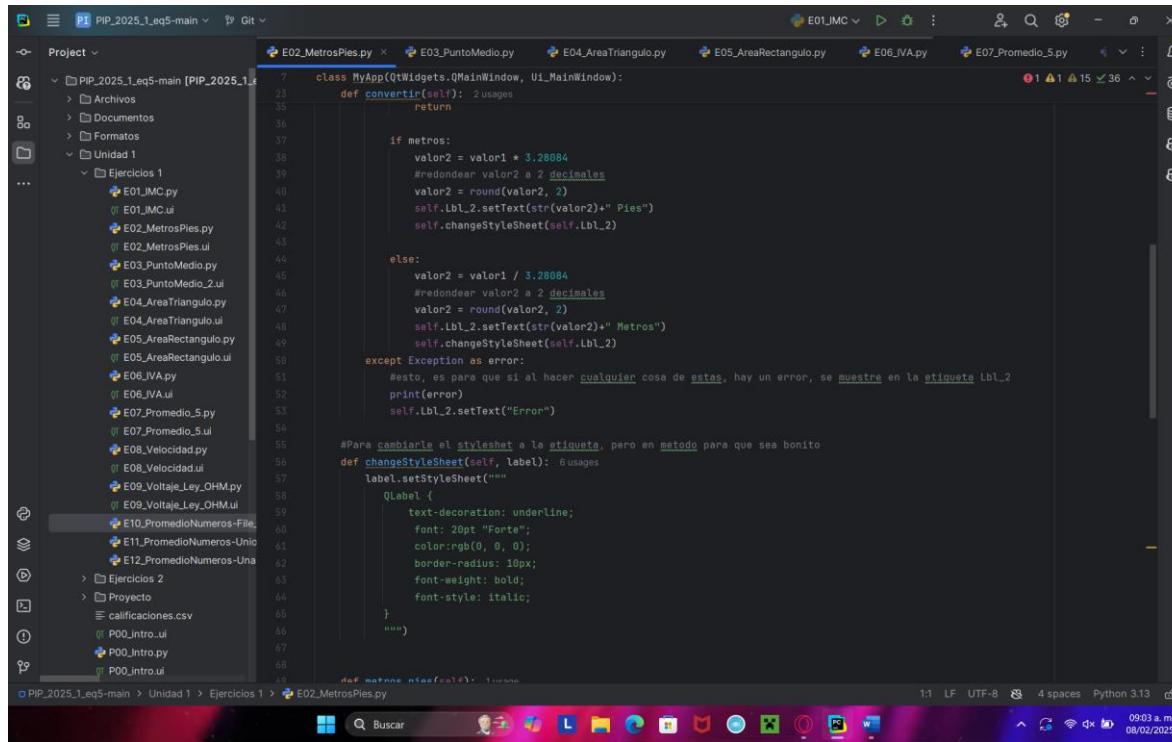


De igual manera que con el ejercicio anterior, en este ejercicio tenemos un apartado visual parecido. Aquí solamente tenemos un TextField y dos botones. Esto con la finalidad de ejercer la conversión sin el uso de dos entradas de texto.



```
1 import sys
2 from PyQt5 import uic, QtWidgets, QtCore
3 qtCreatorFile = "E02_MetrosPies.ui"
4 Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
5 metros = True
6
7 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
8     def __init__(self):
9         QtWidgets.QMainWindow.__init__(self)
10        Ui_MainWindow.__init__(self)
11        self.setupUi(self)
12
13        # La acción de convertir se puso para ambos botones para no tener que incluir un botón extra
14
15        # Acciones del botón mp (metros a pies)
16        self.btn_mp.clicked.connect(self.metros_pies)
17        self.btn_mp.clicked.connect(self.convertir)
18
19        # Acciones del botón pm (pies a metros)
20        self.btn_pm.clicked.connect(self.pies_metros)
21        self.btn_pm.clicked.connect(self.convertir)
22        self.txt_1.textChanged.connect(self.CheckLetter)
23
24    def convertir(self): 2 usages
25        # valor 1 es el valor que se ingresa en el txt_1, que es el valor que se va a convertir a otra unidad
26        # valor 2 es el resultado del valor 1 convertido.
27
28        # valor 1 SIEMPRE estará en txt_1
29        # valor 2 SIEMPRE se mostrará enLbl_2
30
31    try:
32        try:
33            valor1 = float(self.txt_1.text())
34        except:
35            self.msj("Por favor, ingrese un valor numérico")
36            return
37
38        if metros == True:
39            resultado = valor1 * 3.28084
40            self.lbl_2.setText(str(resultado))
41            self.txt_1.setText(str(valor1))
42
43        else:
44            resultado = valor1 / 3.28084
45            self.lbl_2.setText(str(resultado))
46            self.txt_1.setText(str(valor1))
47
48    except:
49        self.msj("Por favor, ingrese un valor numérico")
50
51
52
53
54
55
56
57
58
59
59
```

Tal cual como se describe en los comentarios del programa, la función se usó de esta manera para no hacer uso de dos botones para la conversión de unidades. Contamos con la función *convertir()*, esta nos ayudará a generar la lógica necesaria para obtener ambos resultados mediante la obtención de los datos. Cabe destacar que tenemos por entendido que ambas señales deben estar incluidas en el método, de lo contrario, el programa no funcionaría correctamente.



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()

    def convertir(self):
        usages
        return

        if metros:
            valor2 = valor1 * 3.28084
            #redondear valor2 a 2 decimales
            valor2 = round(valor2, 2)
            self.lbl_2.setText(str(valor2)+" Pies")
            self.setStyleSheet(self.lbl_2)

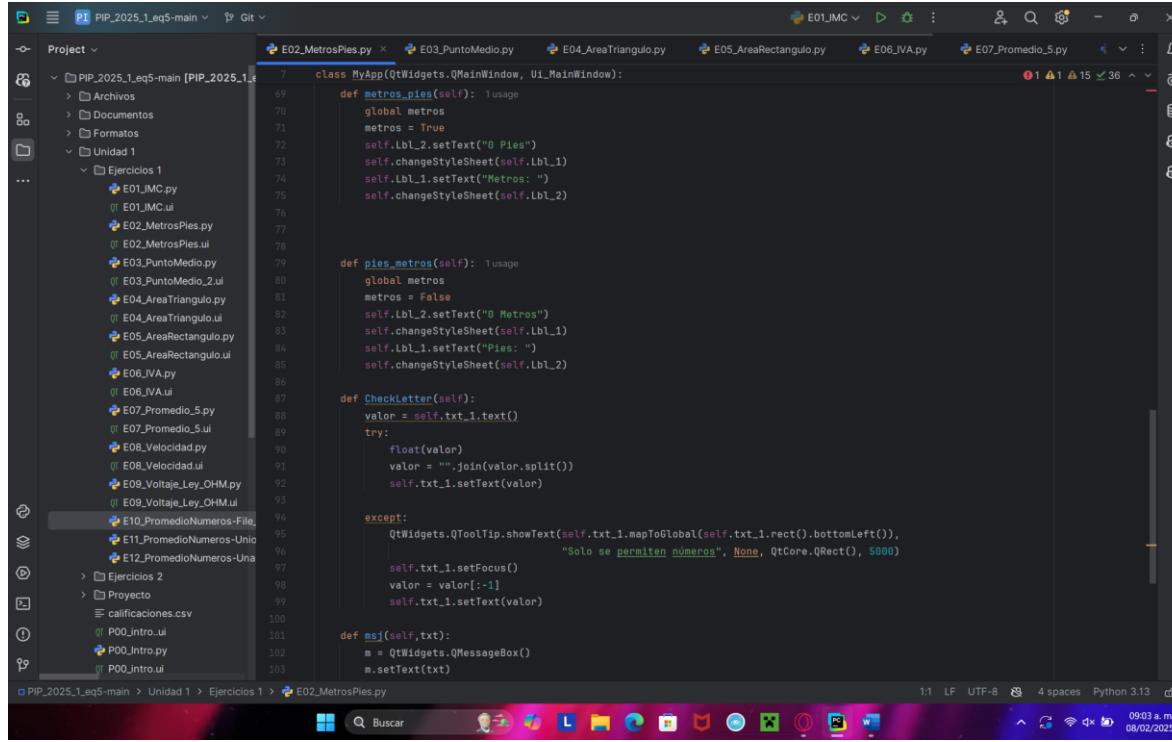
        else:
            valor2 = valor1 / 3.28084
            #redondear valor2 a 2 decimales
            valor2 = round(valor2, 2)
            self.lbl_2.setText(str(valor2)+" Metros")
            self.setStyleSheet(self.lbl_2)

    except Exception as error:
        #esto, es para que si el hacer cualquier cosa de estas, hay un error, se muestre en la etiquetaLbl_2
        print(error)
        self.lbl_2.setText("Error")

#Para  cambiarle el stylesheet a la etiqueta, pero en metodo para que sea bonito
def changeStyleSheet(self, label):
    usages
    label.setStyleSheet("""
        QLabel {
            text-decoration: underline;
            font: 20pt "Forte";
            color:rgb(0, 0, 0);
            border-radius: 10px;
            font-weight: bold;
            font-style: italic;
        }
    """)

    #Para cambiarle el stylesheet a la etiqueta, pero en metodo para que sea bonito
def changeStyleSheet(self, label):
    usages
    label.setStyleSheet("""
        QLabel {
            text-decoration: underline;
            font: 20pt "Forte";
            color:rgb(0, 0, 0);
            border-radius: 10px;
            font-weight: bold;
            font-style: italic;
        }
    """)
```

Se valida inicialmente antes de proceder a la conversión de unidades, y posteriormente se muestran en el label correspondiente. De lo contrario, el except actúa y se muestra un mensaje de error.



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()

    def metros_pies(self):
        global metros
        metros = True
        self.lbl_2.setText("0 Pies")
        self.setStyleSheet(self.lbl_1)
        self.lbl_1.setText("Metros: ")
        self.setStyleSheet(self.lbl_2)

    def pies_metros(self):
        global metros
        metros = False
        self.lbl_2.setText("0 Metros")
        self.setStyleSheet(self.lbl_1)
        self.lbl_1.setText("Pies: ")
        self.setStyleSheet(self.lbl_2)

    def CheckLetter(self):
        valor = self.txt_1.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_1.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_1.mapToGlobal(self.txt_1.rect().bottomLeft()),
                                        "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_1.setFocus()
            valor = valor[:-1]
            self.txt_1.setText(valor)

    def msj(self,txt):
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.setStandardButtons(QtWidgets.QMessageBox.Ok)
        m.exec()

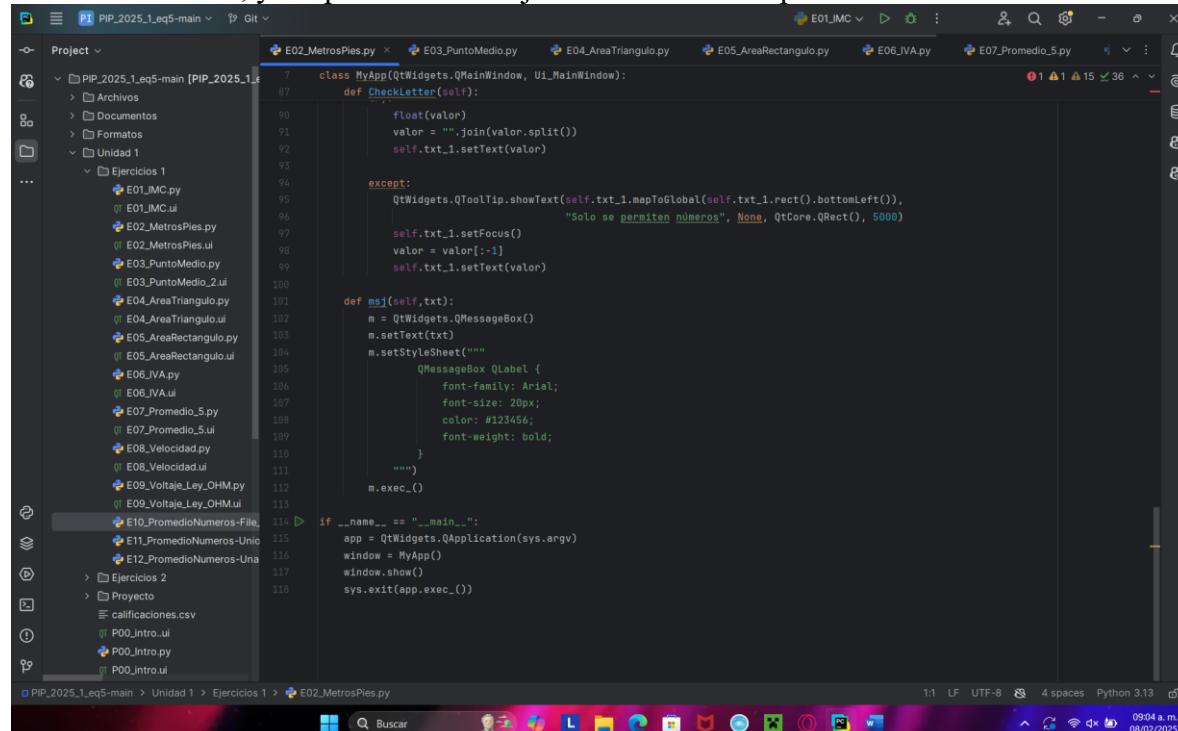
    def convertir(self):
        usages
        if self.txt_1.text() == "":
            self.msj("Por favor ingrese un número")
            self.txt_1.setFocus()
            return
        valor = self.txt_1.text()
        if self.txt_1.text().find(',') != -1:
            valor = valor.replace(',', '.')
        valor = float(valor)
        if metros:
            valor2 = valor * 3.28084
            #redondear valor2 a 2 decimales
            valor2 = round(valor2, 2)
            self.lbl_2.setText(str(valor2)+" Pies")
            self.setStyleSheet(self.lbl_2)

        else:
            valor2 = valor / 3.28084
            #redondear valor2 a 2 decimales
            valor2 = round(valor2, 2)
            self.lbl_2.setText(str(valor2)+" Metros")
            self.setStyleSheet(self.lbl_2)
```



Tanto en *metros\_pies* como en *pies\_metros* se hace uso de los labels, ya que, estos no implementan lógica, solamente se llaman para mostrar el resultado en los labels haciendo uso de otro método para que se vea estilizado.

Empleamos también la validación de datos, que solamente contengan números y no caracteres o letras, y así procedemos a ejecutar lo necesario para mostrar el resultado.



```
class MyApp(QtWidgets.QMainWindow, U1_MainWindow):
    def CheckLetter(self):
        float(valor)
        valor = "".join(valor.split())
        self.txt_1.setText(valor)

    except:
        QtWidgets.QToolTip.showText(self.txt_1.mapToGlobal(self.txt_1.rect().bottomLeft()),
                                    "Solo se permiten numeros", None, QtCore.QRect(), 5000)
        self.txt_1.setFocus()
        valor = valor[:-1]
        self.txt_1.setText(valor)

    def msg(self,txt):
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.setStyleSheet("""
            QMessageBox QLabel {
                font-family: Arial;
                font-size: 20px;
                color: #123456;
                font-weight: bold;
            }
        """)
        m.exec_()

    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```

De igual manera, también se añade una función para estilizar los mensajes mediante el uso de un MessageBox.

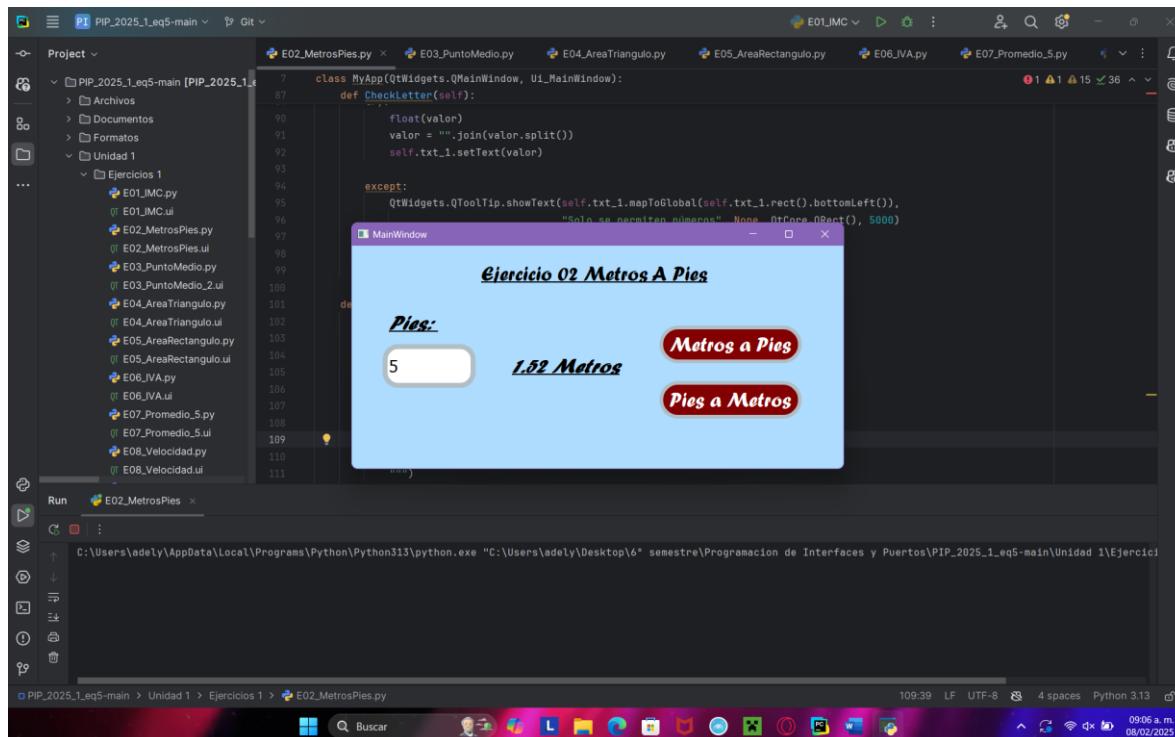


```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):  
    def CheckLetter(self):  
        valor = self.txt_1.text()  
        float(valor)  
        valor = ''.join(valor.split())  
        self.txt_1.setText(valor)  
  
    except:  
        QtWidgets.QToolTip.showText(self.txt_1.mapToGlobal(self.txt_1.rect().bottomLeft()),  
                                    "Sólo se permiten números", None, QtCore.QRect(), 5000)
```

Validamos la entrada de datos.

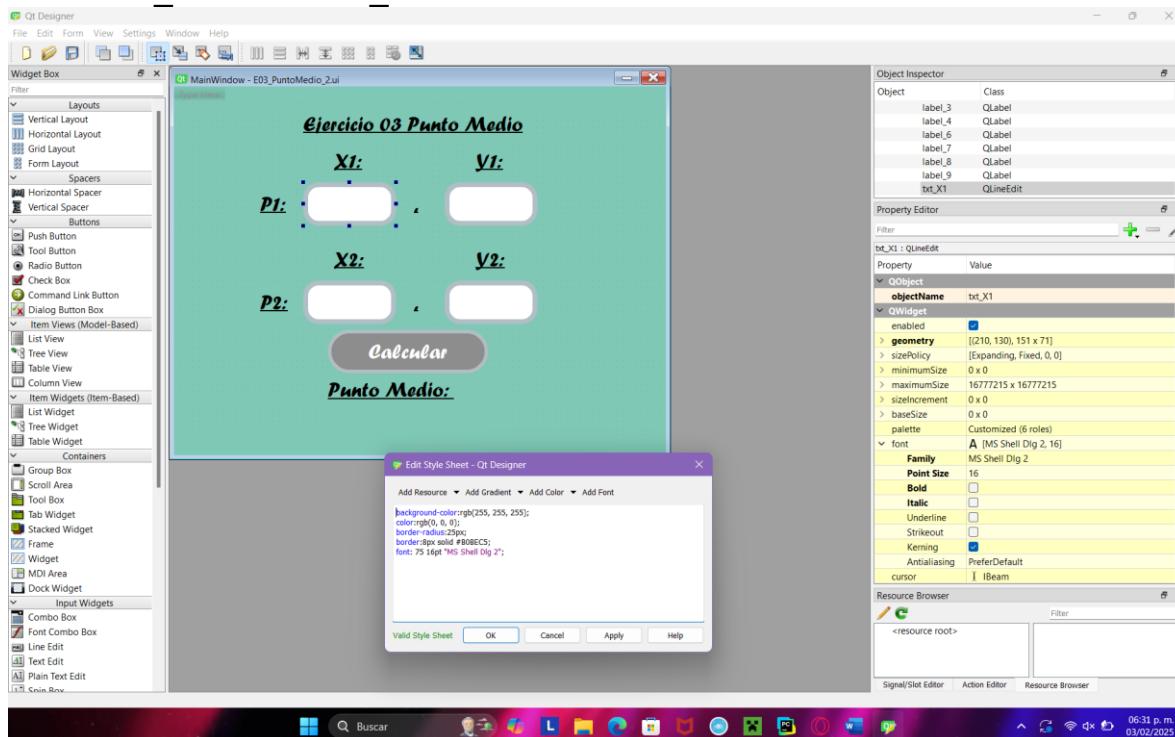
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):  
    def CheckLetter(self):  
        valor = self.txt_1.text()  
        float(valor)  
        valor = ''.join(valor.split())  
        self.txt_1.setText(valor)  
  
    except:  
        QtWidgets.QToolTip.showText(self.txt_1.mapToGlobal(self.txt_1.rect().bottomLeft()),  
                                    "Sólo se permiten números", None, QtCore.QRect(), 5000)
```

Ingresamos los valores a calcular.

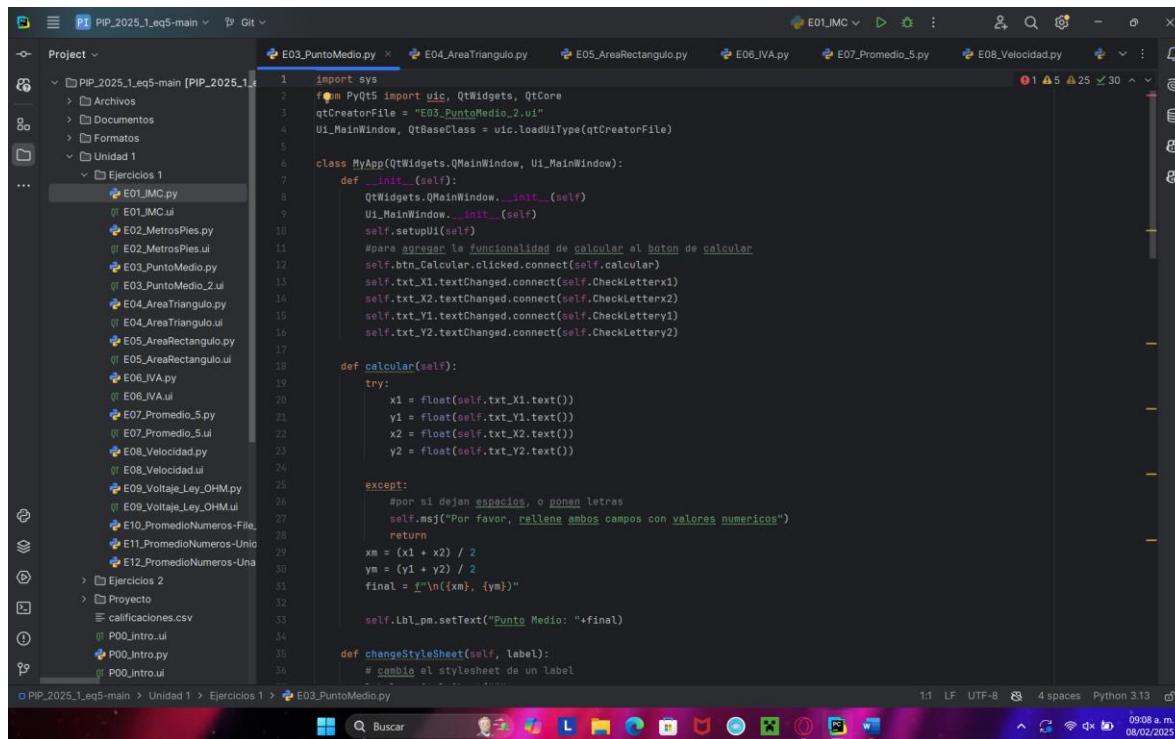


Se muestra el resultado dependiendo qué botón accionemos.

## C12. E03\_PuntoMedio\_2



En este programa, como estaremos calculando el punto medio, requerimos forzosamente la entrada de cuatro datos diferentes para hacer los cálculos necesarios que nos conducirán al resultado. Por lo tanto, requerimos cuatro TextField, y solamente un botón y un label son lo más esencial para poder ejercer la lógica y mostrar al usuario el resultado esperado.



```
import sys
from PyQt5 import uic, QtWidgets, QtCore
qtCreatorFile = "E03_PuntoMedio_2.ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        #para agregar la funcionalidad de calcular al botón de calcular
        self.btn_Calcular.clicked.connect(self.calcular)
        self.txt_X1.textChanged.connect(self.CheckLetterx1)
        self.txt_X2.textChanged.connect(self.CheckLetterx2)
        self.txt_Y1.textChanged.connect(self.CheckLettery1)
        self.txt_Y2.textChanged.connect(self.CheckLettery2)

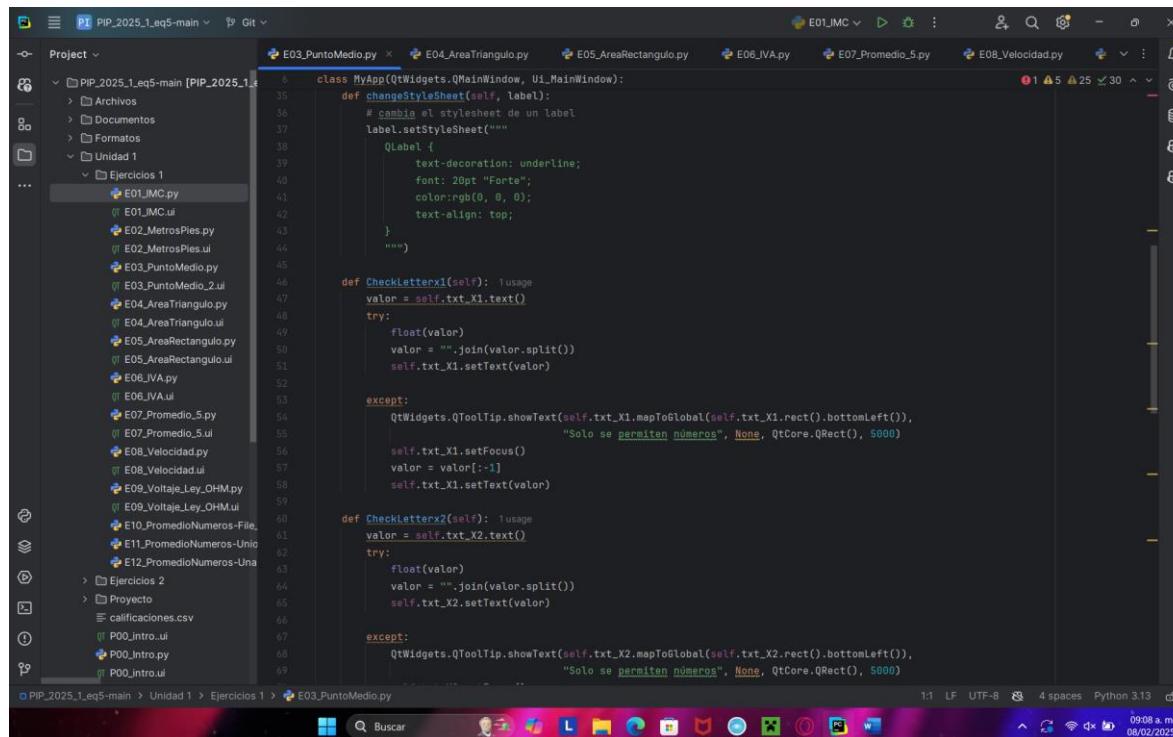
    def calcular(self):
        try:
            x1 = float(self.txt_X1.text())
            y1 = float(self.txt_Y1.text())
            x2 = float(self.txt_X2.text())
            y2 = float(self.txt_Y2.text())

        except:
            #por si dejan espacios, o ponen letras
            self.msg("Por favor, rellene ambos campos con valores numéricos")
            return
        xm = (x1 + x2) / 2
        ym = (y1 + y2) / 2
        final = f"\n\n{xm}, {ym}"

        self.lbl_pm.setText("Punto Medio: "+final)

    def changeStyleSheet(self, label):
        # cambia el stylesheet de un label
```

Contamos primeramente con los signals, como en todo programa. Y también con la función *calcular*, aquí vamos a poner toda la lógica para implementar una solución esperada. Primeramente, convertimos la entrada de datos en números de punto flotante para que puedan ser procesados correctamente, y en caso de que no sea posible, se muestra un mensaje al usuario. Seguido de eso, procesamos el resultado y lo mostramos en el label correspondiente.



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def changeStyleSheet(self, label):
        # cambia el stylesheet de un label
        label.setStyleSheet("""
            QLabel {
                text-decoration: underline;
                font: 20pt "Forte";
                color:rgb(0, 0, 0);
                text-align: top;
            }
        """)

    def CheckLetterx1(self): 1 usage
        valor = self.txt_X1.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_X1.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_X1.mapToGlobal(self.txt_X1.rect().bottomLeft()),
                                         "Solo se permiten numeros", None, QtCore.QRect(), 5000)
            self.txt_X1.setFocus()
            valor = valor[:-1]
            self.txt_X1.setText(valor)

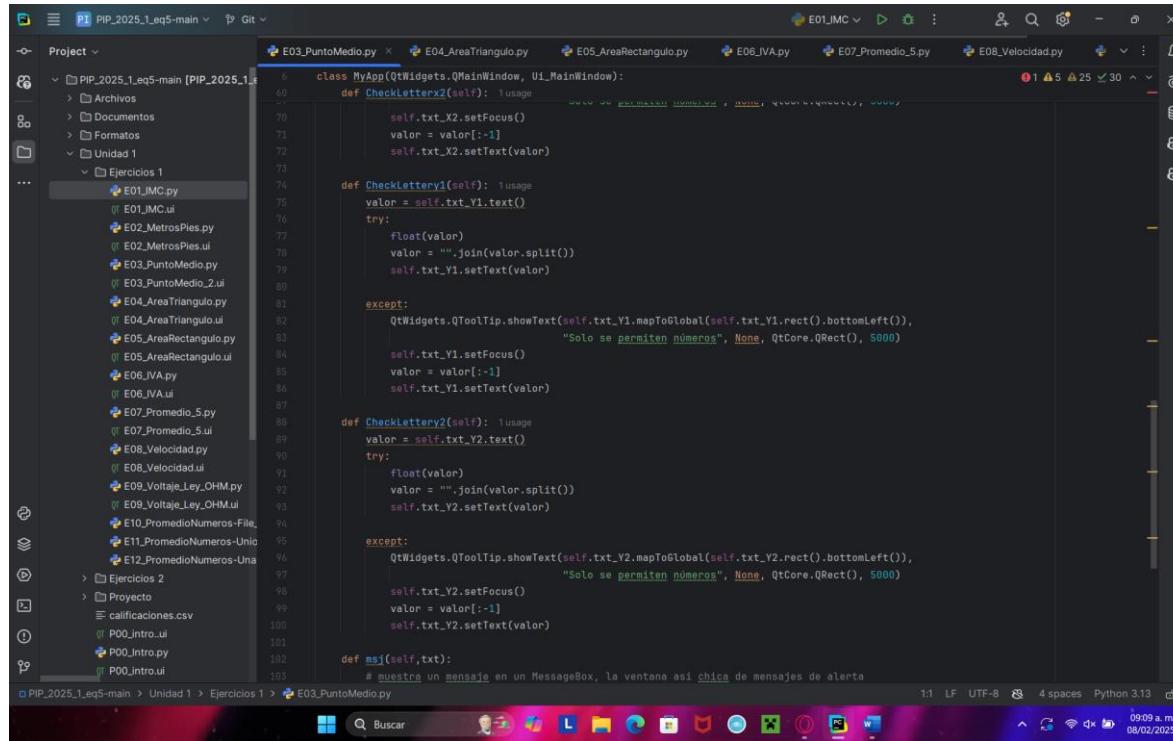
    def CheckLetterx2(self): 1 usage
        valor = self.txt_X2.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_X2.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_X2.mapToGlobal(self.txt_X2.rect().bottomLeft()),
                                         "Solo se permiten numeros", None, QtCore.QRect(), 5000)
            self.txt_X2.setFocus()
            valor = valor[:-1]
            self.txt_X2.setText(valor)

    def CheckLettery1(self): 1 usage
        valor = self.txt_Y1.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_Y1.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_Y1.mapToGlobal(self.txt_Y1.rect().bottomLeft()),
                                         "Solo se permiten numeros", None, QtCore.QRect(), 5000)
            self.txt_Y1.setFocus()
            valor = valor[:-1]
            self.txt_Y1.setText(valor)

    def CheckLettery2(self): 1 usage
        valor = self.txt_Y2.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_Y2.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_Y2.mapToGlobal(self.txt_Y2.rect().bottomLeft()),
                                         "Solo se permiten numeros", None, QtCore.QRect(), 5000)
            self.txt_Y2.setFocus()
            valor = valor[:-1]
            self.txt_Y2.setText(valor)

    def msj(self,txt):
        # muestra un mensaje en un MessageBox, la ventana asi chica de mensajes de alerta
```

Y así como en todos los programas, nuestro objetivo es el uso correcto del mismo. Por ende, también se aplica la validación de datos, verificando si son números y no letras, porque de lo contrario, se le notificará al usuario que solamente son permitidos los valores de tipo numérico. Esto se valida en las cuatro entradas de datos.



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def changeStyleSheet(self, label):
        # cambia el stylesheet de un label
        label.setStyleSheet("""
            QLabel {
                text-decoration: underline;
                font: 20pt "Forte";
                color:rgb(0, 0, 0);
                text-align: top;
            }
        """)

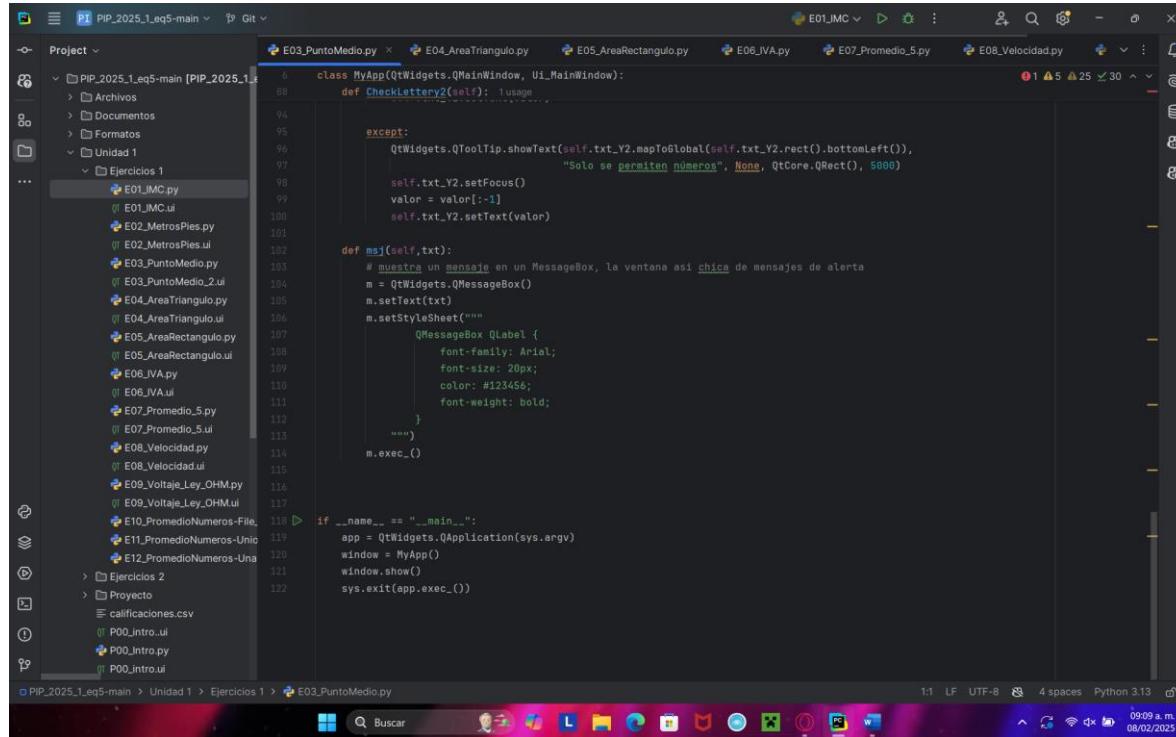
    def CheckLetterx1(self): 1 usage
        valor = self.txt_X1.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_X1.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_X1.mapToGlobal(self.txt_X1.rect().bottomLeft()),
                                         "Solo se permiten numeros", None, QtCore.QRect(), 5000)
            self.txt_X1.setFocus()
            valor = valor[:-1]
            self.txt_X1.setText(valor)

    def CheckLetterx2(self): 1 usage
        valor = self.txt_X2.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_X2.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_X2.mapToGlobal(self.txt_X2.rect().bottomLeft()),
                                         "Solo se permiten numeros", None, QtCore.QRect(), 5000)
            self.txt_X2.setFocus()
            valor = valor[:-1]
            self.txt_X2.setText(valor)

    def CheckLettery1(self): 1 usage
        valor = self.txt_Y1.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_Y1.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_Y1.mapToGlobal(self.txt_Y1.rect().bottomLeft()),
                                         "Solo se permiten numeros", None, QtCore.QRect(), 5000)
            self.txt_Y1.setFocus()
            valor = valor[:-1]
            self.txt_Y1.setText(valor)

    def CheckLettery2(self): 1 usage
        valor = self.txt_Y2.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_Y2.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_Y2.mapToGlobal(self.txt_Y2.rect().bottomLeft()),
                                         "Solo se permiten numeros", None, QtCore.QRect(), 5000)
            self.txt_Y2.setFocus()
            valor = valor[:-1]
            self.txt_Y2.setText(valor)

    def msj(self,txt):
        # muestra un mensaje en un MessageBox, la ventana asi chica de mensajes de alerta
```



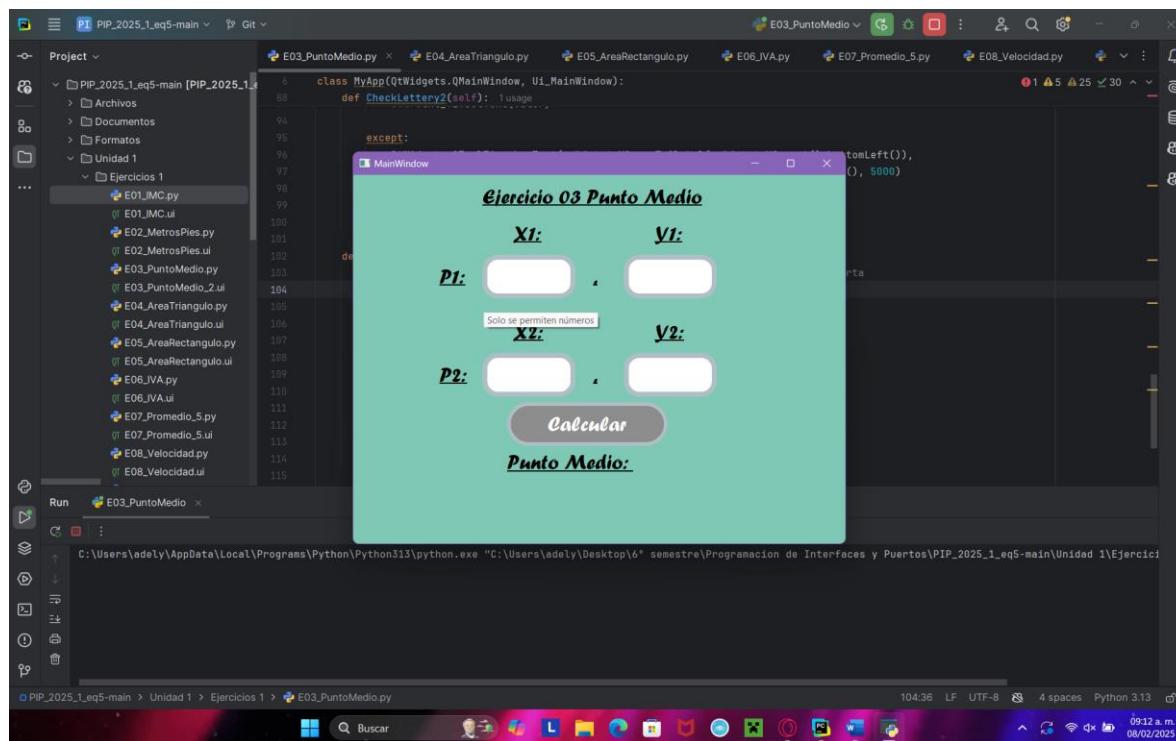
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def CheckLetter2(self): 1 usage

    except:
        QtWidgets.QToolTip.showText(self.txt_Y2.mapToGlobal(self.txt_Y2.rect().bottomLeft()),
                                    "Solo se permiten números", None, QtCore.QRect(), 5000)

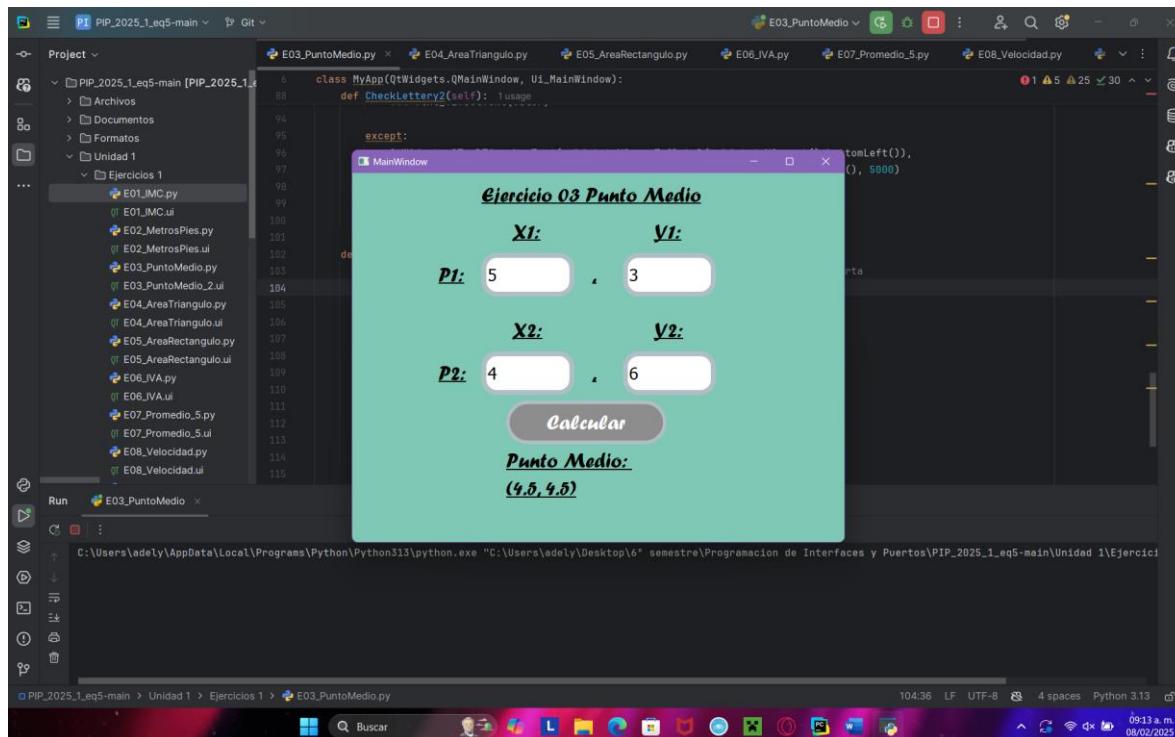
        self.txt_Y2.setFocus()
        valor = valor[:-1]
        self.txt_Y2.setText(valor)

    def msg(self,txt):
        # muestra un mensaje en un MessageBox, la ventana así chica de mensajes de alerta
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.setStyleSheet("""
            QMessageBox QLabel {
                font-family: Arial;
                font-size: 20px;
                color: #123456;
                font-weight: bold;
            }
        """)
        m.exec_()

    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = Myapp()
        window.show()
        sys.exit(app.exec_())
```

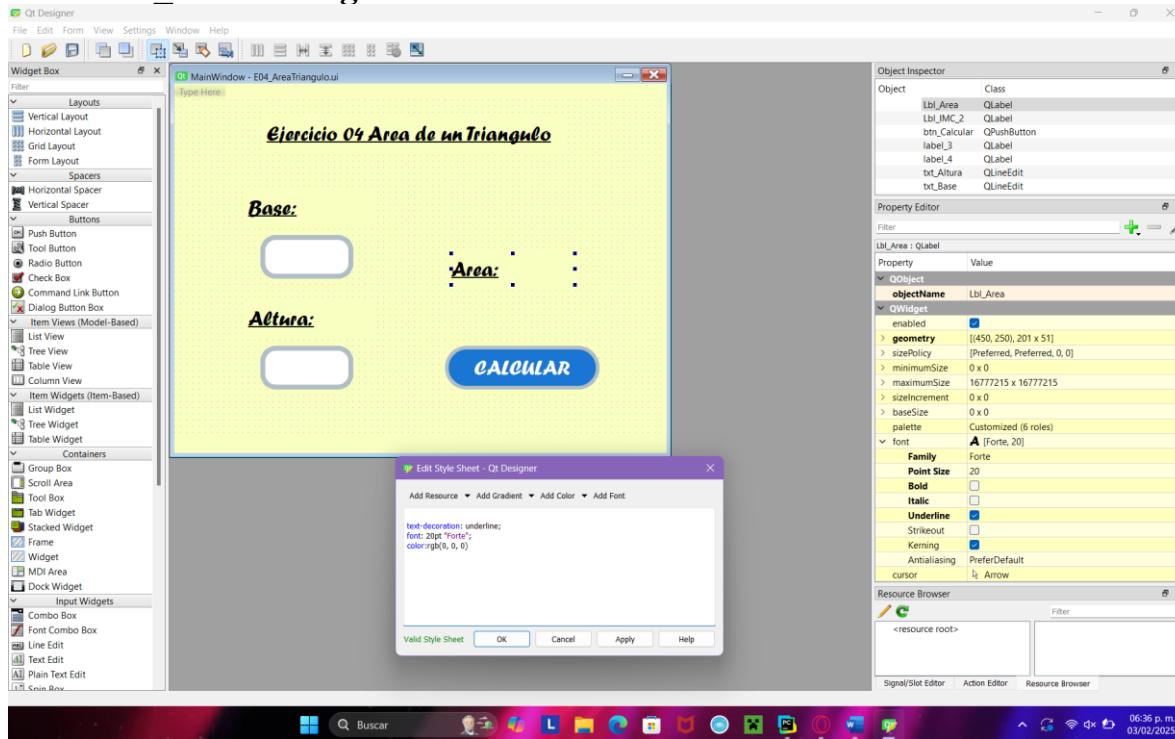


Nos muestra un mensaje emergente en donde dice que solamente se permiten números.

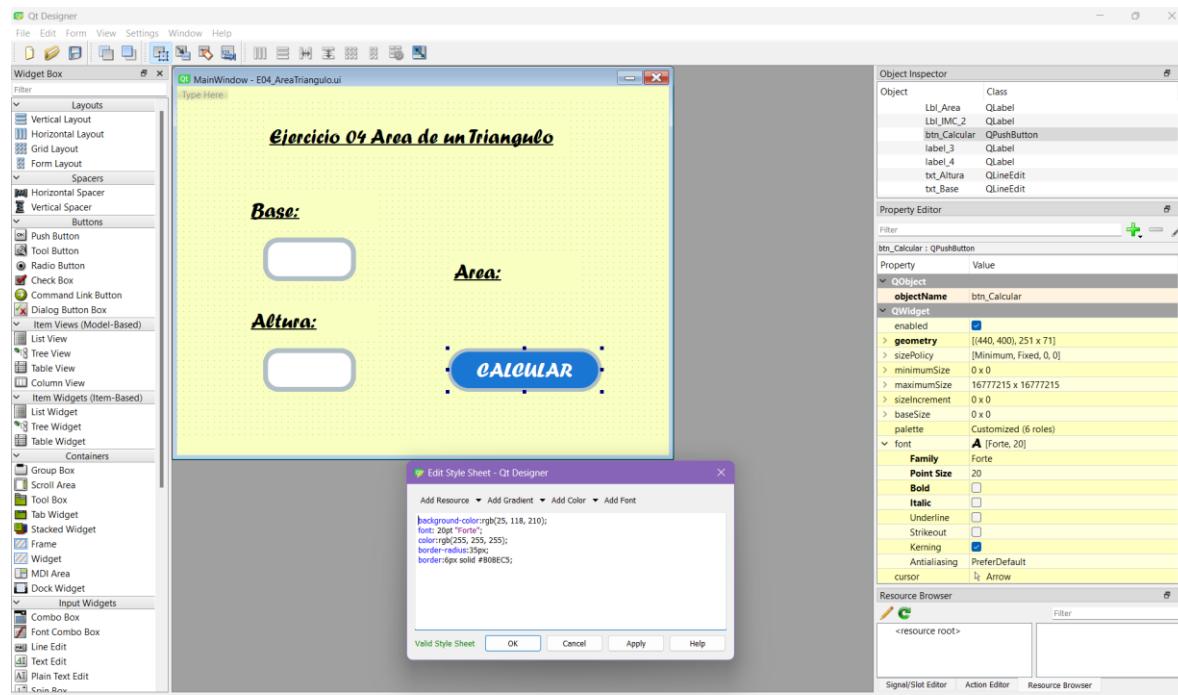


Si ingresamos datos válidos, se hace el proceso y se muestra el resultado correcto.

### C13. E04\_AreaTriangulo



Necesitamos calcular el área de un triángulo, así que para eso requerimos su base y su altura, dos entradas de datos para mostrar su área en el label correspondiente dentro del programa.



```
1 import sys
2 from PyQt5 import uic, QtWidgets, QtCore
3 qtCreatorFile = "E04_AreaTriangulo.ui"
4 UI_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
5
6 class MyApp(QtWidgets.QMainWindow, UI_MainWindow):
7     def __init__(self):
8         QtWidgets.QMainWindow.__init__(self)
9         UI_MainWindow.__init__(self)
10        self.setupUi(self)
11        #para agregar la funcionalidad de calcular el botón de calcular
12        self.btn_Calcular.clicked.connect(self.calcular)
13        self.txt_Base.textChanged.connect(self.CheckLetter)
14        self.txt_Altura.textChanged.connect(self.CheckLetterA)
15
16    def calcular(self):
17        try:
18            base = float(self.txt_Base.text())
19            altura = float(self.txt_Altura.text())
20        except:
21            #por si dejan espacios, o ponen letras
22            self.esj("Por favor, rellene ambos campos con valores numéricos")
23            return
24        area = (base * altura) / 2
25        self.lbl_Area.setText("Área: "+str(area))
26        self.setStyleSheet(self.lbl_Area)
27
28    def changeStyleSheet(self, label): 1 usage
29        # cambia el stylesheet de un label
30        label.setStyleSheet("""
31            QLabel {
32                text-decoration: underline;
33                font: 20pt "Forte";
34                color:rgb(0, 0, 0);
35            }
36        """)
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
```

De igual forma que con todos los programas, se hacen las mismas validaciones para que los datos sean ingresados de manera correcta. Calculamos su área usando la fórmula del triángulo, y procedemos a mostrarla.



The screenshot shows the PyCharm IDE interface with the project 'PIP\_2025\_1\_eq5-main' open. The current file is 'E04\_AreaTriangulo.py'. The code implements a Qt application to calculate the area of a triangle. It includes validation logic to ensure input is a number. If not, it displays a message box indicating only numbers are allowed.

```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self, txt):
        # muestra un mensaje en un QMessageBox, la ventana así chica de mensajes de alerta
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.setStyleSheet("""
            QMessageBox QLabel {
                font-family: Arial;
                font-size: 20px;
                color: #123456;
                font-weight: bold;
            }
        """)
        m.exec_()

    def CheckLetter(self):
        valor = self.txt_Base.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_Base.setText(valor)

        except:
            QtWidgets.QToolTip.showText(self.txt_Base.mapToGlobal(self.txt_Base.rect().bottomLeft()), "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_Base.setFocus()
            valor = valor[:-1]
            self.txt_Base.setText(valor)

    def CheckLetterA(self): 1 usage
        valor = self.txt_Altaura.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_Altaura.setText(valor)

        except:
            QtWidgets.QToolTip.showText(self.txt_Altaura.mapToGlobal(self.txt_Altaura.rect().bottomLeft()), "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_Altaura.setFocus()
            valor = valor[:-1]
            self.txt_Altaura.setText(valor)
```

Se validan que sean números, de lo contrario, se le indica al usuario que ingrese solo números.

The screenshot shows the PyCharm IDE interface with the project 'PIP\_2025\_1\_eq5-main' open. The current file is 'E04\_AreaTriangulo.py'. The code is identical to the previous one but includes a main block at the bottom. This block creates a QApplication instance, initializes the application, creates a window, shows it, and then exits the application.

```
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```



The screenshot shows the PyCharm IDE interface with a project named "PIP\_2025\_1.eq5-main". The "Ejercicios 1" folder contains several files, including "E04\_AreaTriangulo.py". The code in this file defines a class `MyApp` that inherits from `QtWidgets.QMainWindow`. It includes methods for handling base input and calculating the area. A UI window titled "Ejercicio 04 Área de un Triángulo" is displayed, featuring two input fields labeled "Base:" and "Altura:", a "CALCULAR" button, and a text output field showing the result "Área: 17.5".

Asimismo, validamos que solamente funcione con números.

This screenshot shows the same application running in PyCharm. The user has entered "7" into the "Base:" input field and "5" into the "Altura:" input field. The "CALCULAR" button is highlighted. The output field displays "Área: 17.5", indicating the correct calculation of the triangle's area.

Ingresamos los datos y vemos que el valor del área se muestra correctamente.



## C14. E05\_AreaRectangulo

The screenshot shows the Qt Designer interface for a window titled "Ejercicio 05 Area de un Rectángulo". The window contains the following elements:

- A title bar with the window title.
- A central area with the text "Ejercicio 05 Area de un Rectángulo".
- A label "Largo:" followed by a text input field.
- A label "Ancho:" followed by a text input field.
- A label "Area:" followed by a text output field.
- A large orange button labeled "CALCULAR".
- A "Widget Box" palette on the left containing various UI components like Vertical Layout, Horizontal Layout, Buttons, etc.
- An "Object Inspector" palette on the right listing objects: Lbl\_Area, Lbl\_IMC\_2, btn\_Calcular, label\_3, label\_4, txt\_Ancho, and txt\_Largo.
- A "Property Editor" palette for the "Lbl\_Area" object, showing properties like geometry, font (Forte, 20), and color (black).
- A "Edit Style Sheet - Qt Designer" dialog box showing CSS rules for the "label" element.
- A taskbar at the bottom with various application icons and the system clock showing 06:39 p.m. 03/02/2025.

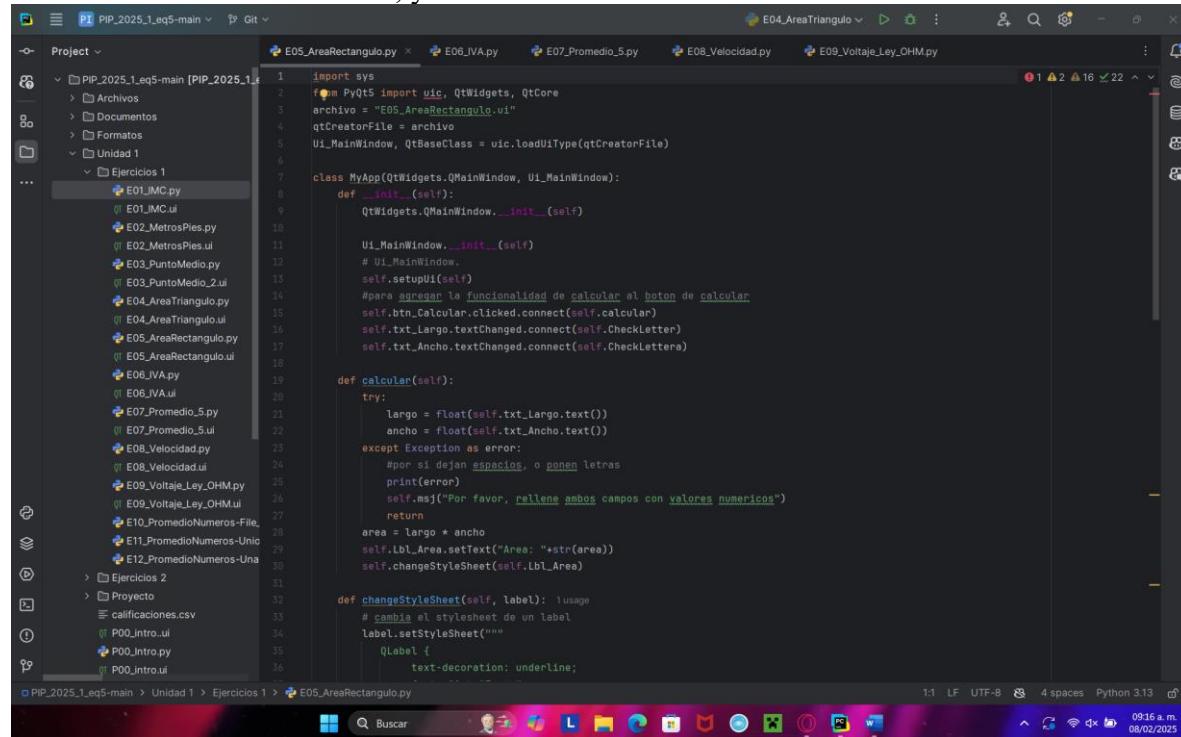
The second part of the screenshot shows the same window after applying a style sheet. The "CALCULAR" button now has a background color of #FF7043, a font size of 20pt "Forte", and a border radius of 15px. The "Edit Style Sheet" dialog shows the updated CSS rule:

```
background-color: #FF7043;  
font: 20pt "Forte";  
color:rgb(255, 255, 255);  
border-radius:15px;  
border:1px solid #B0EBC5;
```

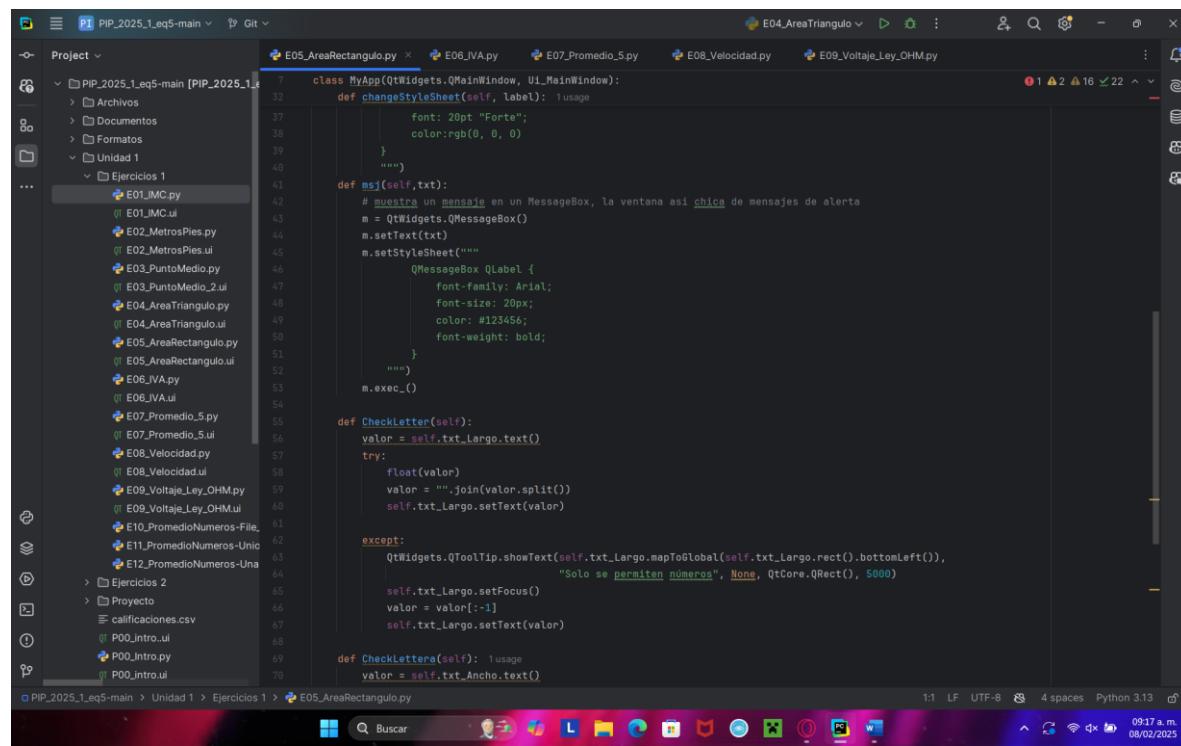
En este programa necesitamos calcular el área del rectángulo, así que solamente es como si se tratara de una copia del anterior programa considerando su facilidad por el uso de su fórmula, porque si juntamos dos triángulos, formaremos un rectángulo, y, por consiguiente,



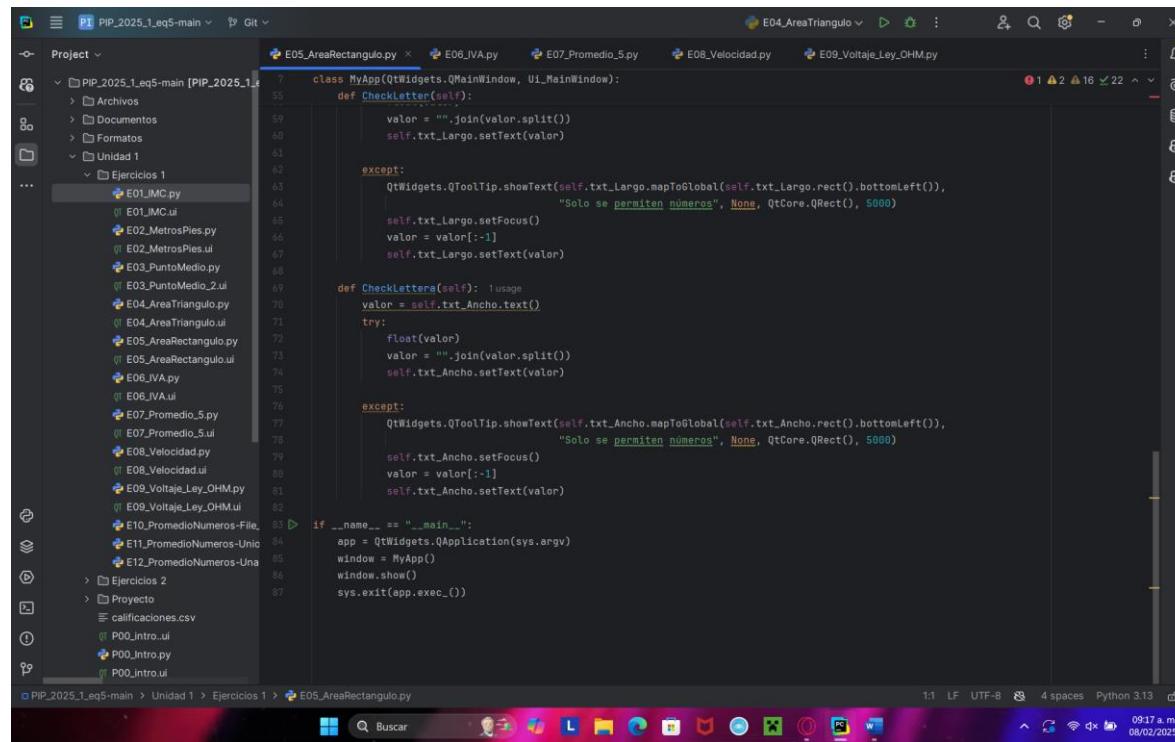
la fórmula que se emplea es tan similar. Como todo es relativamente igual, también lo serán el número de entrada de datos, y el resultado es una sola área.



```
1 import sys
2 from PyQt5 import uic, QtWidgets, QtCore
3 archivo = "E05_AreaRectangulo.ui"
4 qtCreatorFile = archivo
5 ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
6
7 class MyApp(QtWidgets.QMainWindow, ui_MainWindow):
8     def __init__(self):
9         QtWidgets.QMainWindow.__init__(self)
10        # UI_MainWindow.
11        self.setupUi(self)
12        #para agregar la funcionalidad de calcular al boton de calcular
13        self.btn_Calcular.clicked.connect(self.calcular)
14        self.txt_Largo.textChanged.connect(self.CheckLetter)
15        self.txt_Ancho.textChanged.connect(self.CheckLetters)
16
17    def calcular(self):
18        try:
19            largo = float(self.txt_Largo.text())
20            ancho = float(self.txt_Ancho.text())
21        except Exception as error:
22            #por si dejan espacios, o ponen letras
23            print(error)
24            self.msg("Por favor, rellene ambos campos con valores numéricos")
25            return
26        area = largo * ancho
27        self.lbl_Area.setText("Area: "+str(area))
28        self.setStyleSheet("QLabel { color: red; font-weight: bold; }")
29
30    def changeStyleSheet(self, label): 1 usage
31        # cambia el stylesheet de un label
32        label.setStyleSheet("""
33            QLabel {
34                text-decoration: underline;
35            }
36        """)
37
38    def msj(self,txt):
39        # muestra un mensaje en un MessageBox, la ventana así chica de mensajes de alerta
40        m = QtWidgets.QMessageBox()
41        m.setText(txt)
42        m.setStyleSheet("""
43            QMessageBox QLabel {
44                font-family: Arial;
45                font-size: 20px;
46                color: #123456;
47                font-weight: bold;
48            }
49        """)
50        m.exec_()
51
52    def CheckLetter(self):
53        valor = self.txt_Largo.text()
54        try:
55            float(valor)
56        except:
57            QtWidgets.QToolTip.showText(self.txt_Largo.mapToGlobal(self.txt_Largo.rect().bottomLeft()),
58                                         "Solo se permiten números", None, QtCore.QRect(), 5000)
59            self.txt_Largo.setFocus()
60            valor = valor[:-1]
61            self.txt_Largo.setText(valor)
62
63    def CheckLetters(self): 1 usage
64        valor = self.txt_Ancho.text()
```



```
65
66
67
68
69
70
```



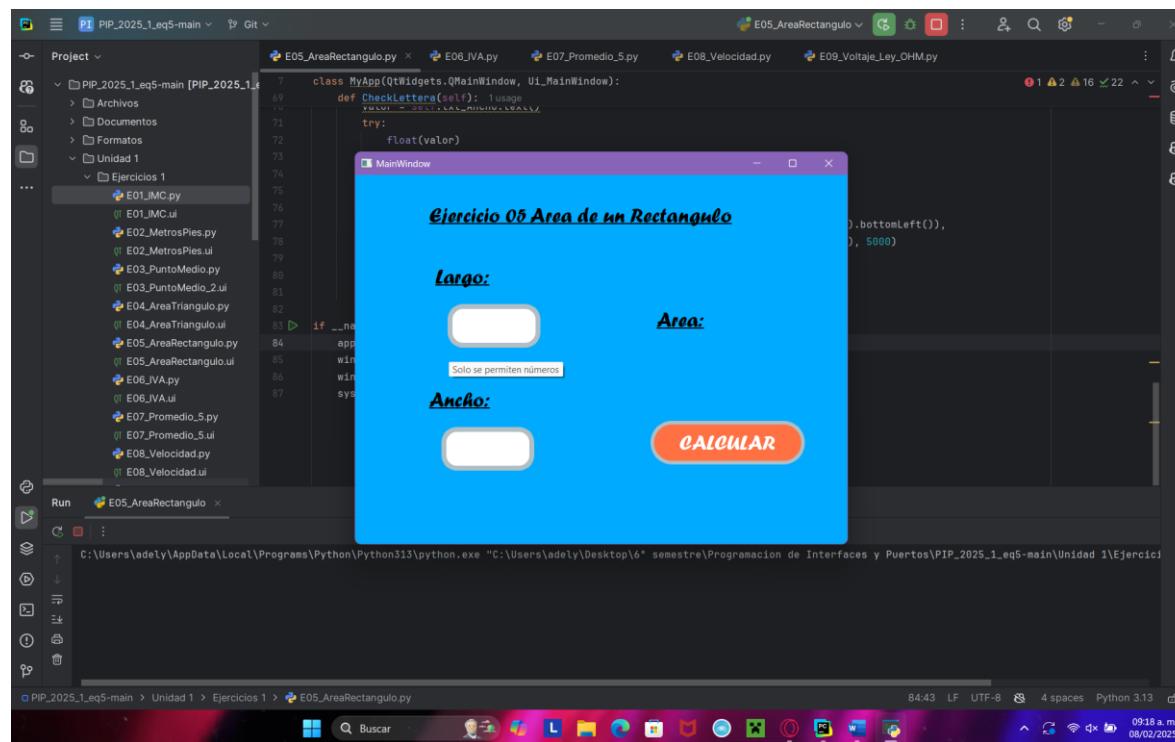
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def CheckLetter(self):
        valor = " ".join(valor.split())
        self.txt_Largo.setText(valor)

    except:
        QtWidgets.QToolTip.showText(self.txt_Largo.mapToGlobal(self.txt_Largo.rect().bottomLeft()),
                                    "Solo se permiten números", None, QtCore.QRect(), 5000)
        self.txt_Largo.setFocus()
        valor = valor[:-1]
        self.txt_Largo.setText(valor)

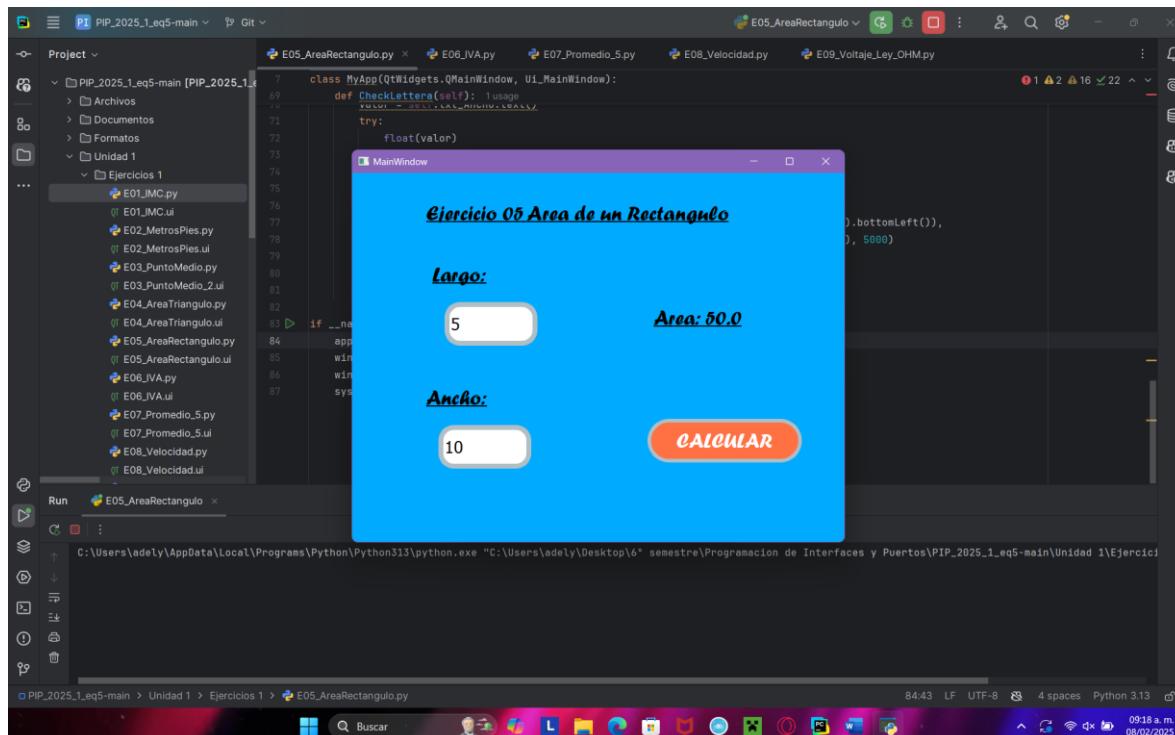
    def CheckAncho(self):
        valor = self.txt_Ancho.text()
        try:
            float(valor)
            valor = " ".join(valor.split())
            self.txt_Ancho.setText(valor)

        except:
            QtWidgets.QToolTip.showText(self.txt_Ancho.mapToGlobal(self.txt_Ancho.rect().bottomLeft()),
                                        "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_Ancho.setFocus()
            valor = valor[:-1]
            self.txt_Ancho.setText(valor)

    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```



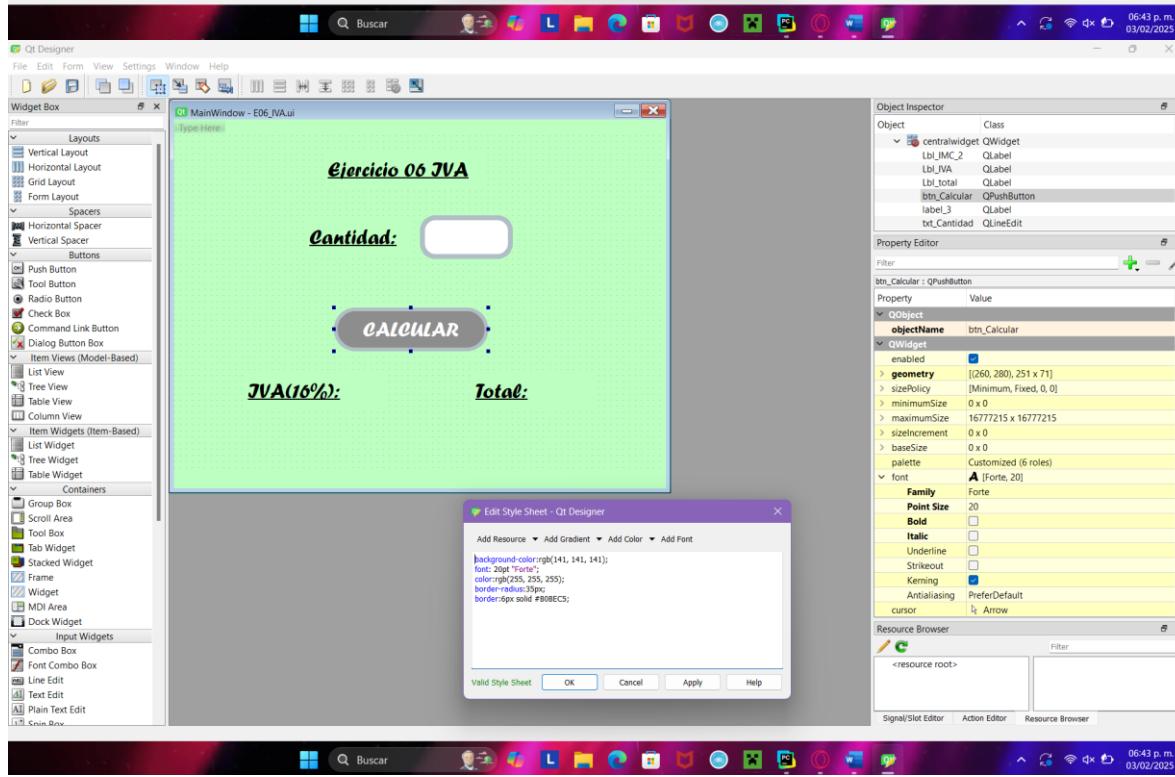
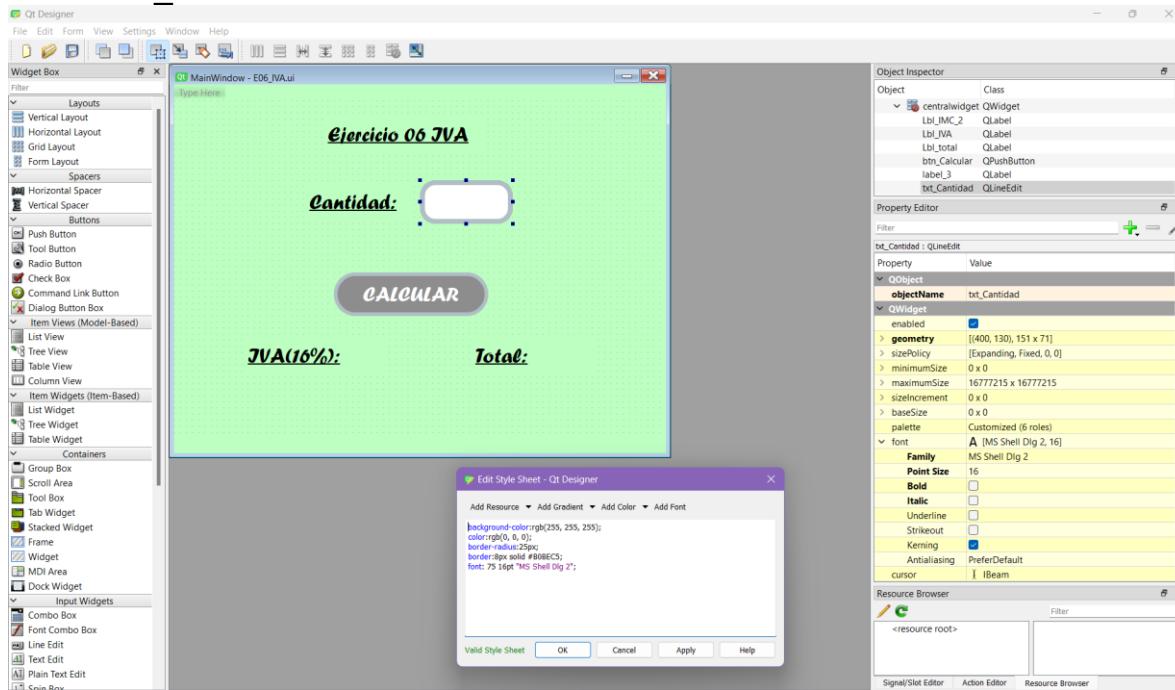
Se valida correctamente que se ingresen números.

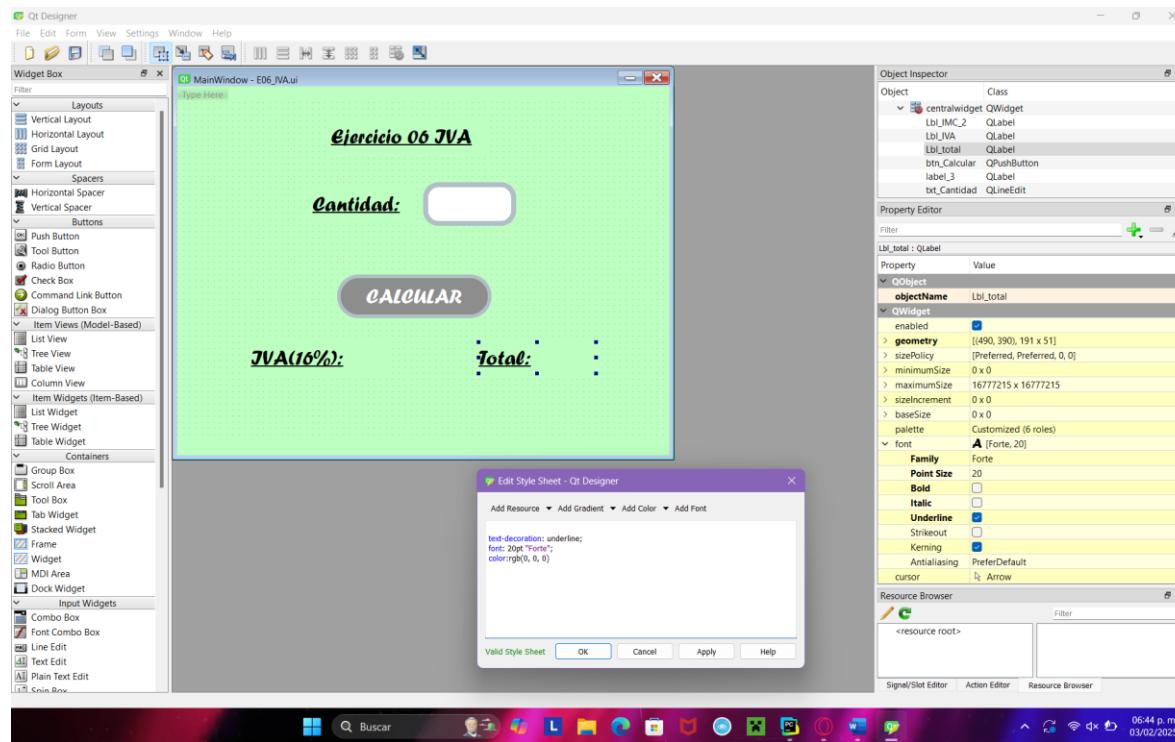


Empleamos la fórmula y se muestra el resultado correspondiente.



## C15. E06\_IVA

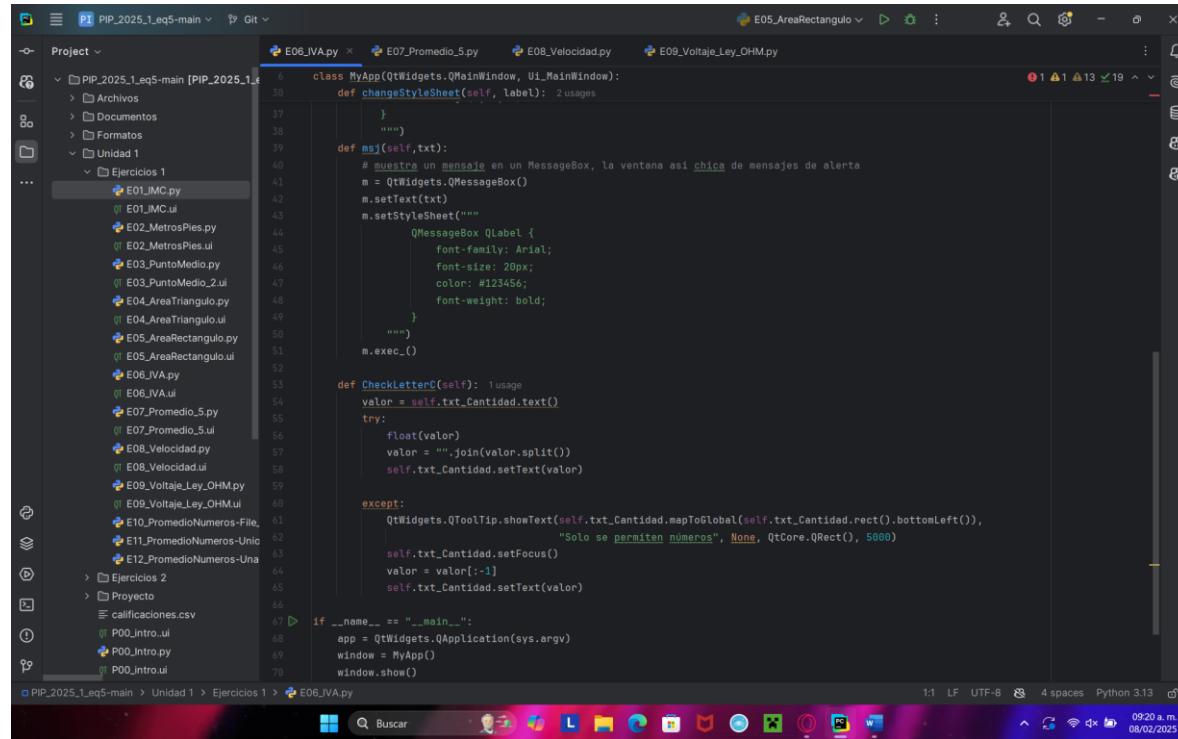




El IVA en México es utilizado por casi todos los comercios, si no es que todos, y en este programa nos encargaremos de implementar la lógica necesaria para poder calcular el IVA en base a una cantidad dada por el usuario.

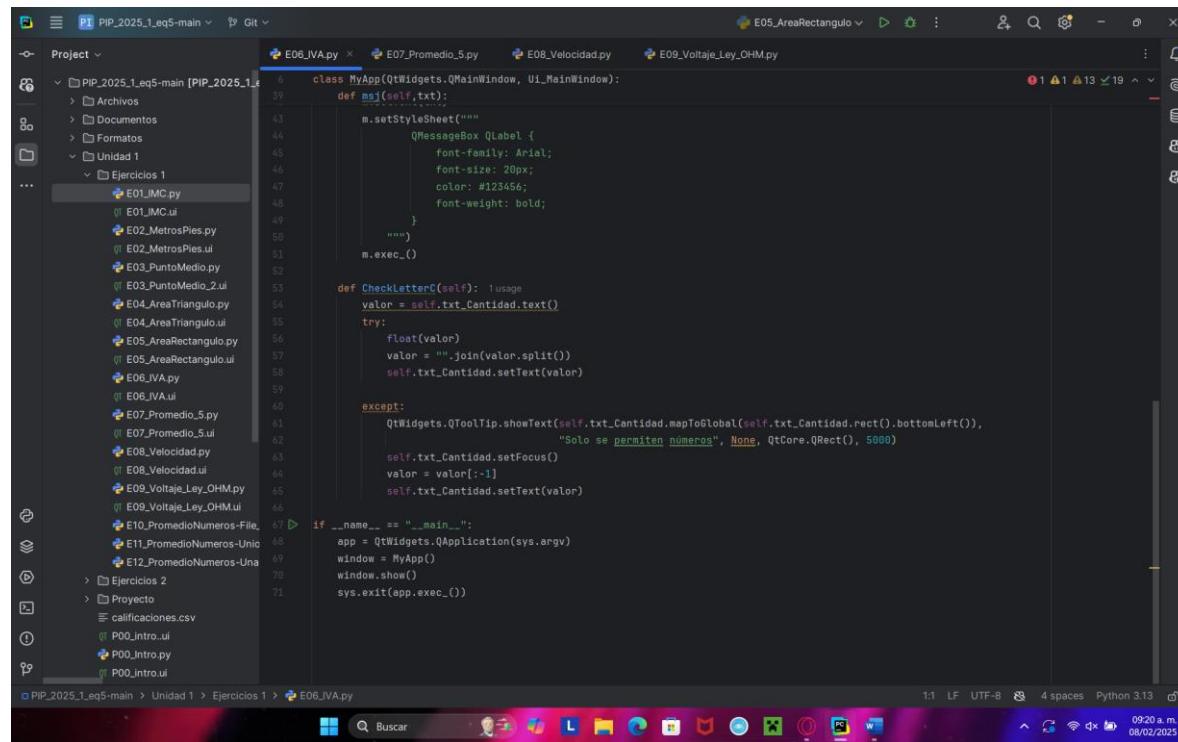


Ya que requerimos del IVA de una cantidad, la fórmula es muy sencilla. Conectamos los signals para que hagan acción y usamos la función correspondiente para emplear la cantidad de IVA, y lo seteamos al label que le corresponde.



The screenshot shows the PyCharm IDE interface with the following details:

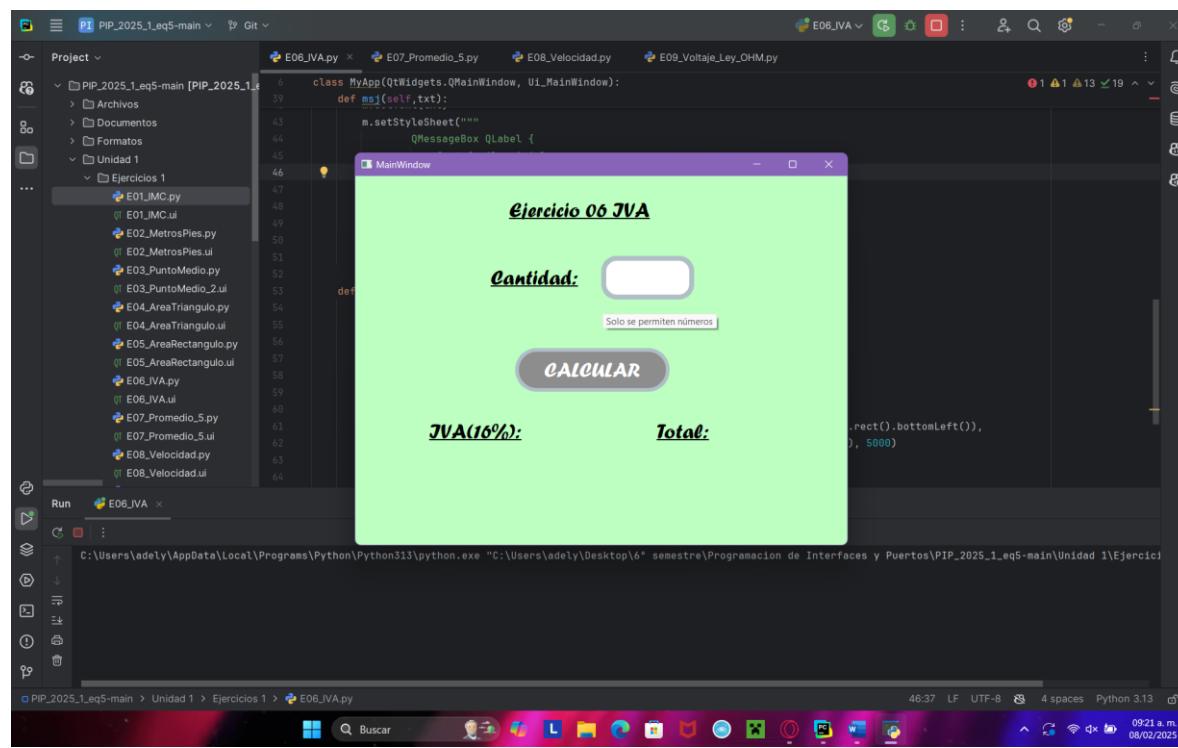
- Project Structure:** The project is named "PIP\_2025\_1\_eq5-main". It contains several sub-directories like "Archivos", "Documentos", "Formatos", and "Unidad 1". Under "Unidad 1/Ejercicios 1", there are multiple files: E01\_IMC.py, E02\_MetrosPies.py, E03\_PuntoMedio.py, E04\_AreaTriangulo.py, E05\_AreaRectangulo.py, E06\_IVA.py, E07\_Promedio\_5.py, E08\_Velocidad.py, and E09\_Voltaje\_Ley\_OHM.py. There are also UI files like E01\_IMC.ui, E02\_MetrosPies.ui, E03\_PuntoMedio.ui, E04\_AreaTriangulo.ui, E05\_AreaRectangulo.ui, E06\_IVA.ui, E07\_Promedio\_5.ui, E08\_Velocidad.ui, and E09\_Voltaje\_Ley\_OHM.ui.
- Code Editor:** The main editor window displays the code for E06\_IVA.py. The code defines a class MyApp that inherits from QMainWindow. It includes methods for changing stylesheets and displaying messages. It also includes a method for checking if input is a float and another for handling exceptions related to non-numeric input. Finally, it has a main function that creates the application and shows the window.
- Status Bar:** The status bar at the bottom shows the file path "PIP\_2025\_1\_eq5-main > Unidad 1 > Ejercicios 1 > E06\_IVA.py", the line number "67", and the status "1:1 LF UTF-8 4 spaces Python 3.13". It also shows the current date and time: "09:20 a.m. 08/02/2025".



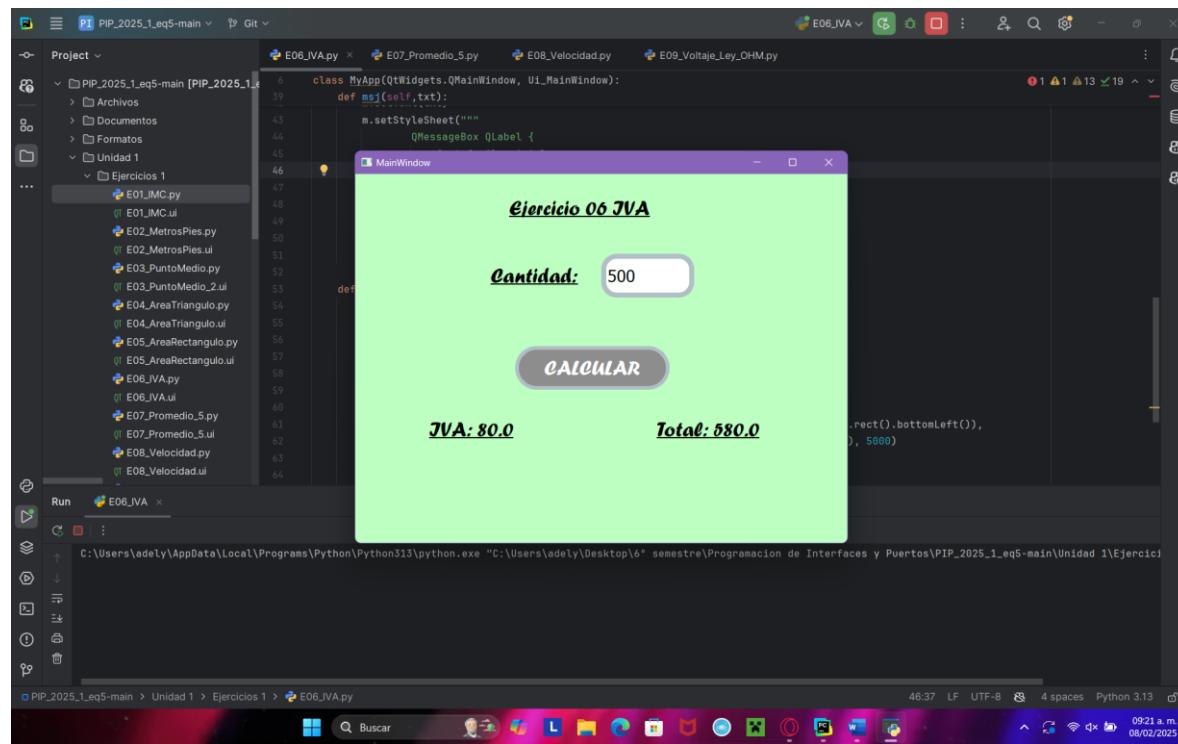
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self, txt):
        super().__init__()
        self.txt = txt
        m.setStyleSheet("""
            QMessageBox QLabel {
                font-family: Arial;
                font-size: 20px;
                color: #123456;
                font-weight: bold;
            }
        """)
        m.exec_()

    def CheckLetterC(self):
        usage
        valor = self.txt_Cantidad.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_Cantidad.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_Cantidad.mapToGlobal(self.txt_Cantidad.rect().bottomLeft()),
                                         "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_Cantidad.setFocus()
            valor = valor[:-1]
            self.txt_Cantidad.setText(valor)

    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```

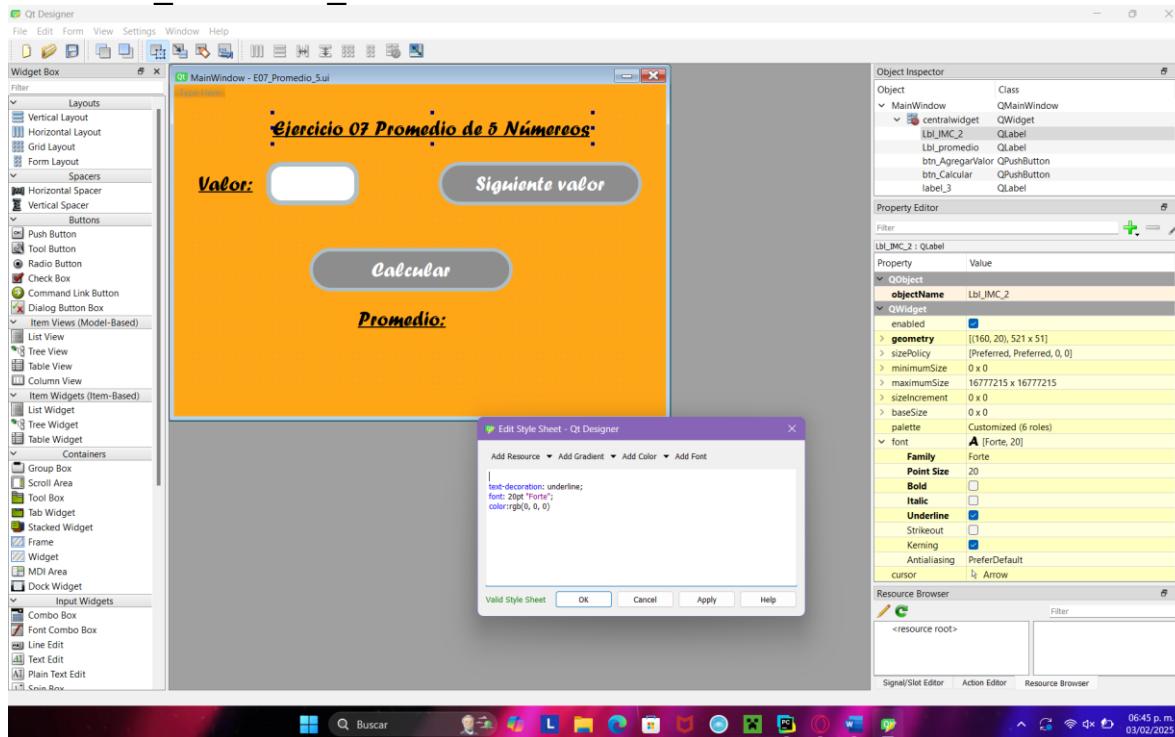


Se valida correctamente con las funciones reutilizadas en anteriores programas que el usuario pueda ingresar solo números. Y nos muestra tanto el IVA de la cantidad, como la cantidad total ya con el IVA incluido.

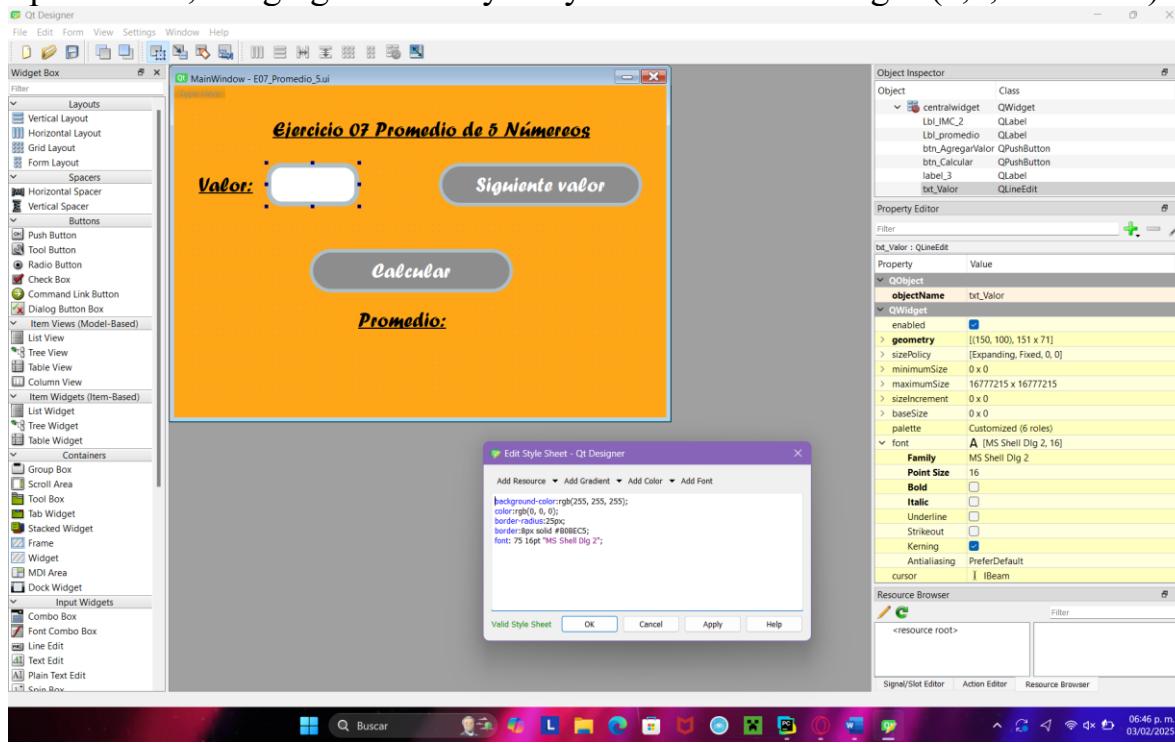




## C16. E07\_Promedio\_5

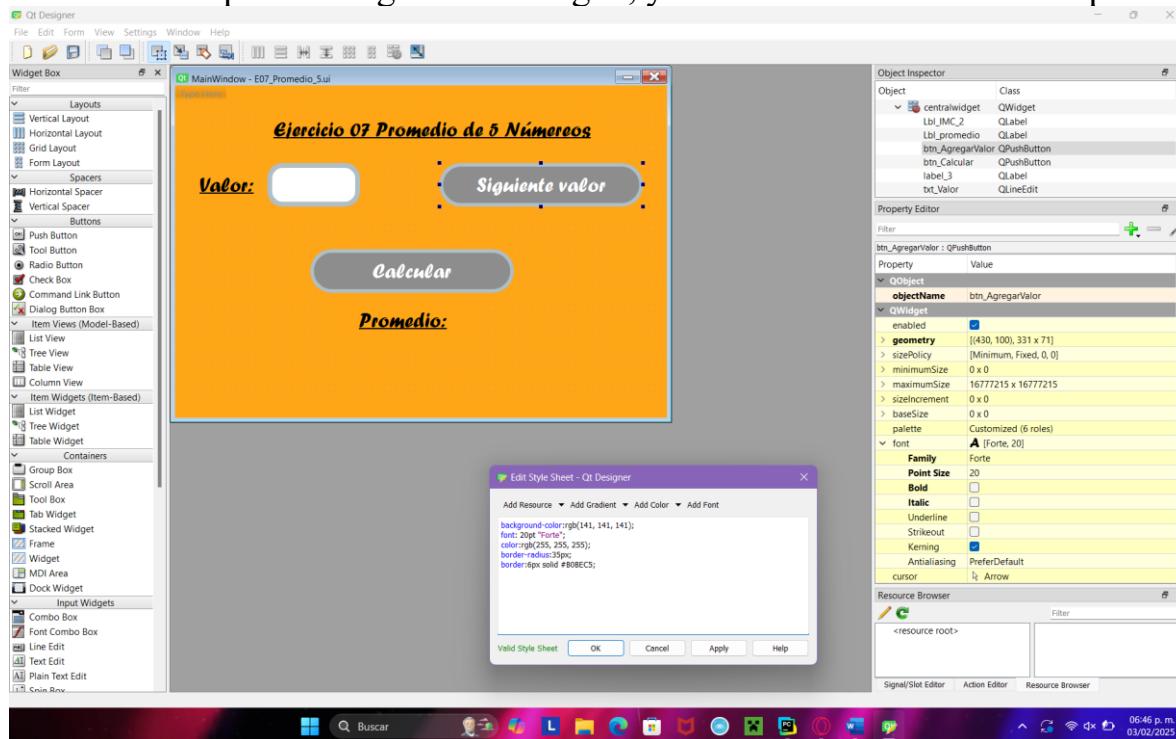


Estilo visual de las etiquetas usadas, con el Style Sheet, mediante cambio de tipo de letra, se agrega un subrayado y se le da el color negro (0,0,0 en RGB)

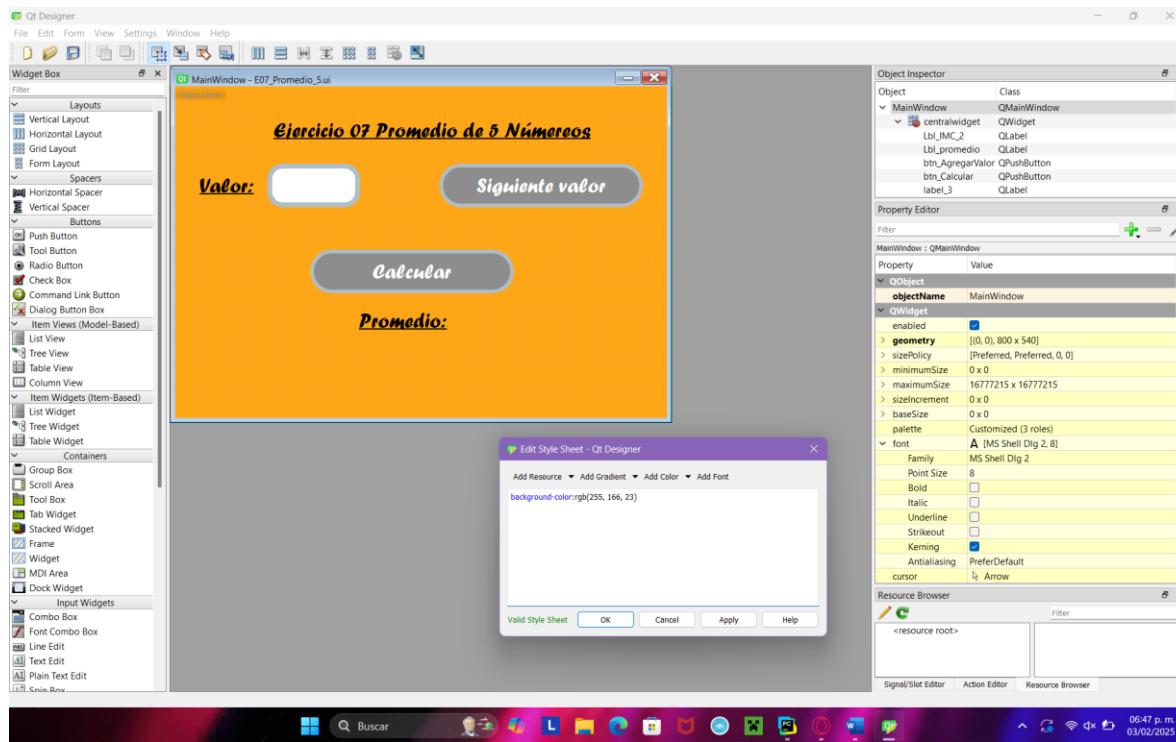




Estilo usado en los TextEdit para su diseño, se le da un color blanco de fondo, además de un color negro para las letras, se establece un tamaño de letra de 16, un borde de 8 puntos de grueso color gris, y con bordes redondeados a 25px



Estilo utilizado en los botones para su diseño, color gris de fondo, Letras blanca y tipo de letra “forte” a 20pt, bordes de 6px con radio a 35px



Color de fondo de la ventana principal

```
import sys
from PyQt5 import QtWidgets, QtCore
qtCreatorFile = "E07_Promedio_5.ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
valores = []
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.btn_Calcular.clicked.connect(self.calcular)
        self.btn_AgregarValor.clicked.connect(self.agregarValor)
        self.txt_Valor.textChanged.connect(self.CheckLetterV)
    def agregarValor(self):  # usage
        if len(valores) == 5:
            self.msg("No se pueden agregar mas valores")
        return
        try:
            valor = float(self.txt_Valor.text())
        except:
            self.msg("Por favor, rellene el campo con valores numericos")
        return
        if len(valores) == 4:
            self.msg("Ya se han ingresado todos los valores")
        valores.append(valor)
        self.txt_Valor.setText("")
        self.txt_Valor.setFocus()
    def calcular(self):
        suma = 0
        for n in valores:
            suma += n
        promedio = suma / 5
        self.lbl_promedio.setText(f"Promedio: {promedio}")
        # self.setStyleSheet(self.lbl_promedio)
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myApp = MyApp()
    myApp.show()
    sys.exit(app.exec_())
```



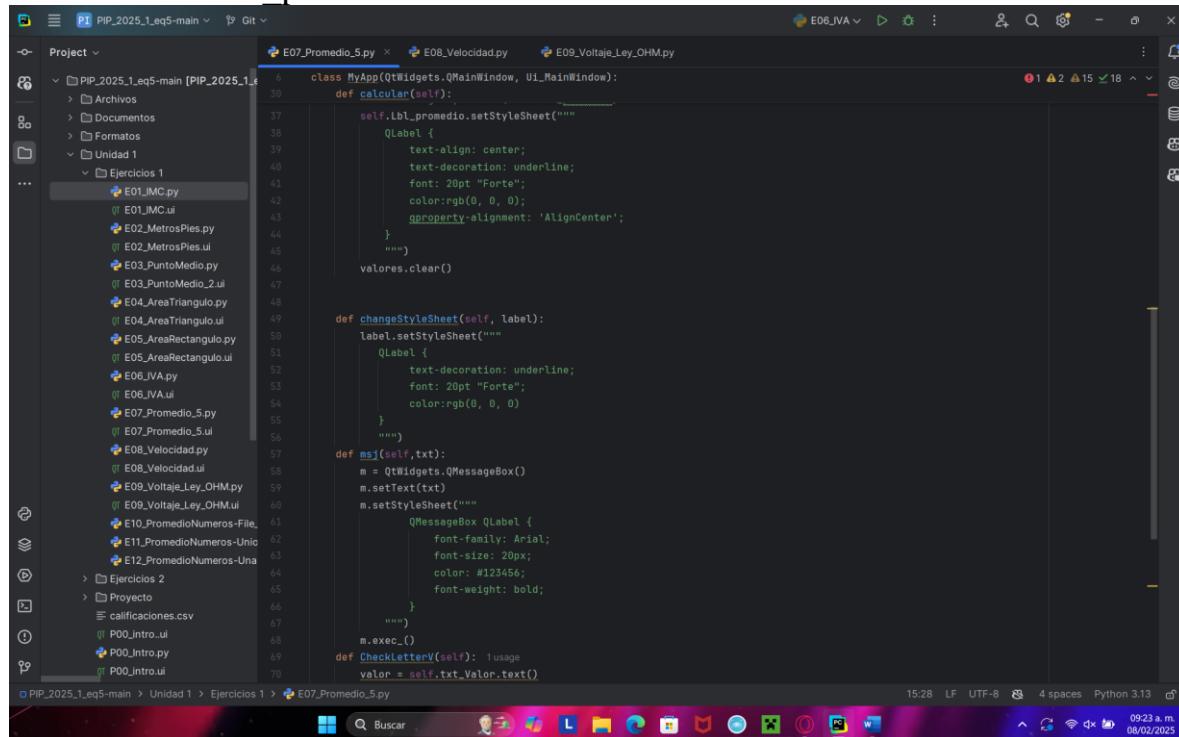
Se le asigna a los botones “btn\_Calcular” y “btn\_AgregarValor” su función de su mismo nombre, y se utilizo la función “textChanged” en “txt\_Valor” para asignarle una función de revisión de caracteres introducidos.

Agregar valor:

Revisa que aun pueda entrar otro valor, guarda el valor en “Valor” para agregarlo a la lista de “valores”, se vacia el txt\_Value y se pone el “Focus” en txt\_Value (el cursor)

Calcular:

Guarda la suma de valores en “suma” para después sacar el promedio y mostrarlo en “lbl\_promedio” con un “SetText”



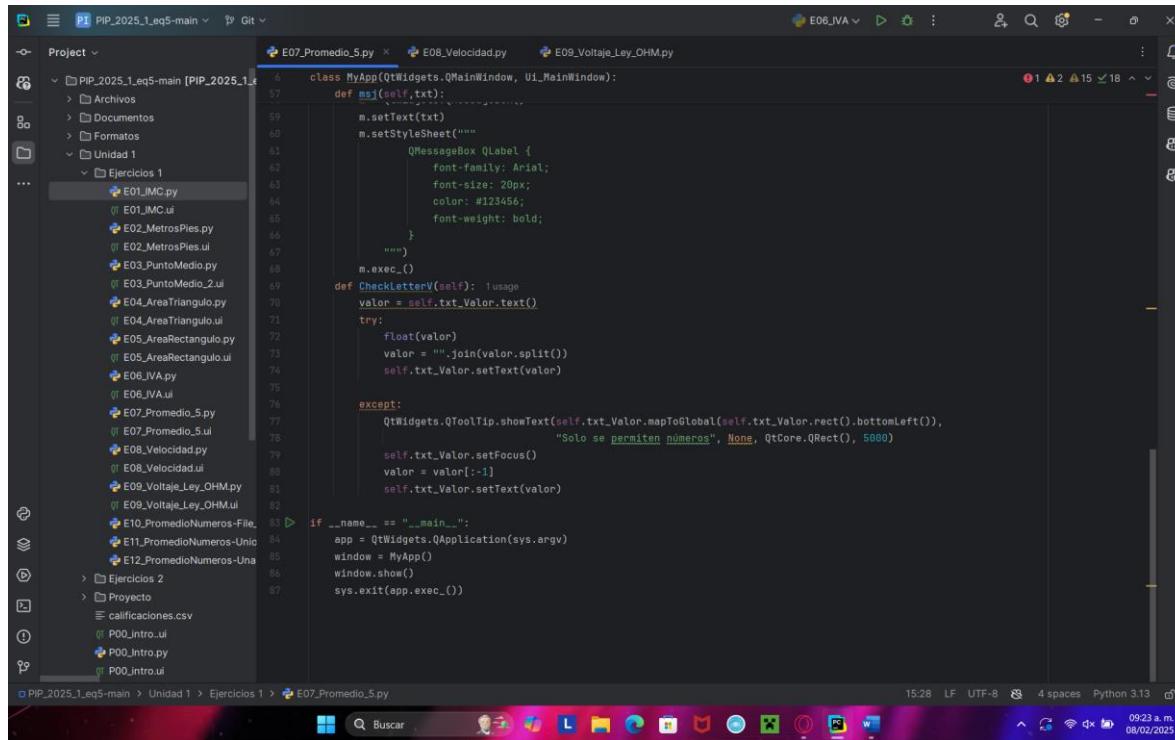
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def calcular(self):
        self.lbl_promedio.setStyleSheet("""
            QLabel {
                text-align: center;
                text-decoration: underline;
                font: 20pt "Forte";
                color:rgb(0, 0, 0);
                qproperty-alignment: 'AlignCenter';
            }
        """)
        valores.clear()

    def changeStyleSheet(self, label):
        label.setStyleSheet("""
            QLabel {
                text-decoration: underline;
                font: 20pt "Forte";
                color:rgb(0, 0, 0)
            }
        """)
        msj(self,txt):
            m = QtWidgets.QMessageBox()
            m.setText(txt)
            m.setStyleSheet("""
                QMessageBox QLabel {
                    font-family: Arial;
                    font-size: 20px;
                    color: #123456;
                    font-weight: bold;
                }
            """)
            m.exec_()
        def CheckLetterV(self): 1 usage
            valor = self.txt_Valor.text()
```

changeStyleSheet:

Cambia la vista de un label, para regresarlo a como se veía antes y no haya cambio.

Msj: muestra una ventana emergente con el texto introducido como atributo.

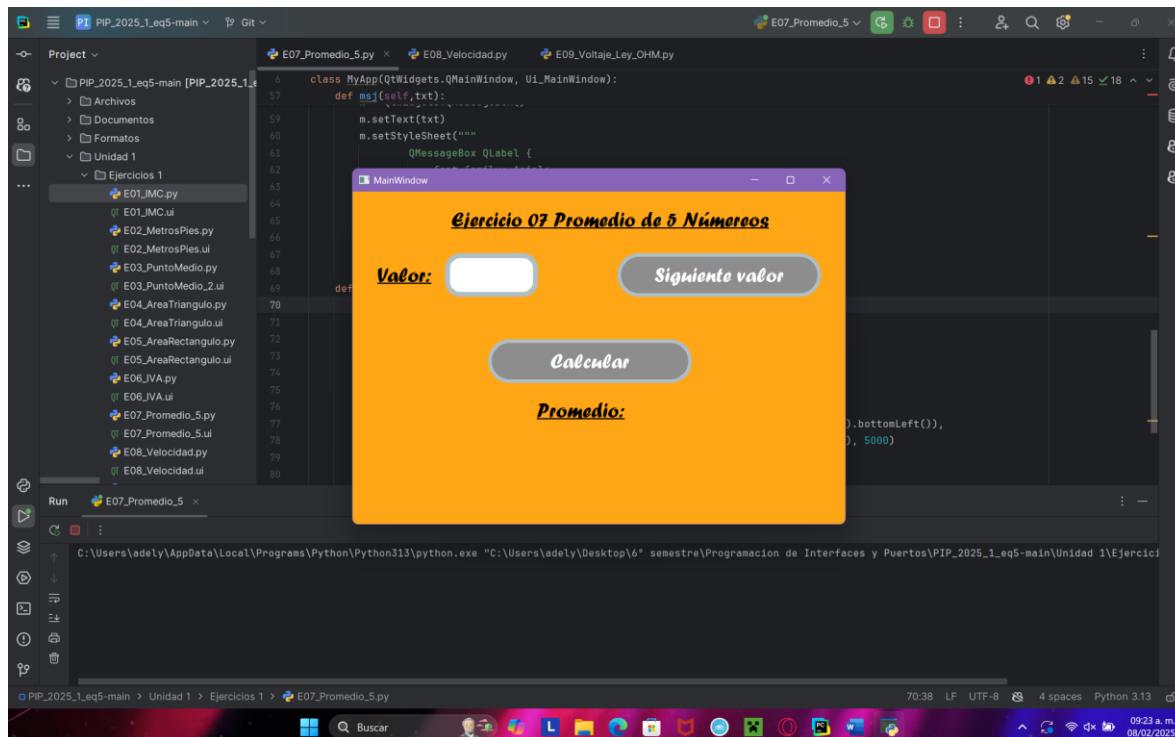


The screenshot shows the PyCharm IDE interface with a project named "PIP\_2025\_1\_eq5-main". The code editor displays a file named "E07\_Promedio\_5.py". The code implements a PyQt application with a central window containing a text input field. It includes validation logic to ensure the input is a float, displaying an error message if it's not. A tooltip is shown above the input field with the message "Solo se permiten números". The status bar at the bottom indicates the current time as 15:28 and the Python version as 3.13.

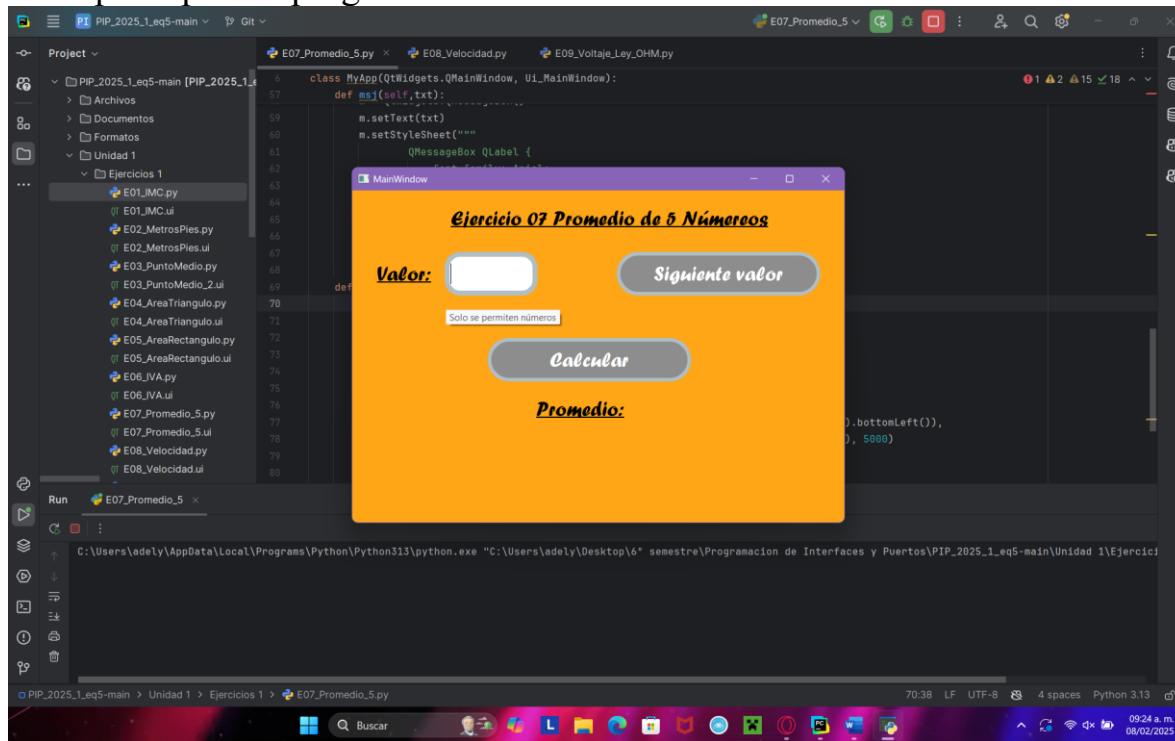
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self, txt):
        m.setText(txt)
        m.setStyleSheet("QLabel {
            font-family: Arial;
            font-size: 20px;
            color: #123456;
            font-weight: bold;
        }")
        m.exec_()
    def CheckLetterV(self):
        valor = self.txt_Vvalor.text()
        try:
            float(valor)
            valor = ''.join(valor.split())
            self.txt_Vvalor.setText(valor)
        except:
            QtWidgets.QToolTip.showText(self.txt_Vvalor.mapToGlobal(self.txt_Vvalor.rect().bottomLeft()),
                                         "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_Vvalor.setFocus()
            valor = valor[:-1]
            self.txt_Vvalor.setText(valor)
    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```

### CheckLetterV:

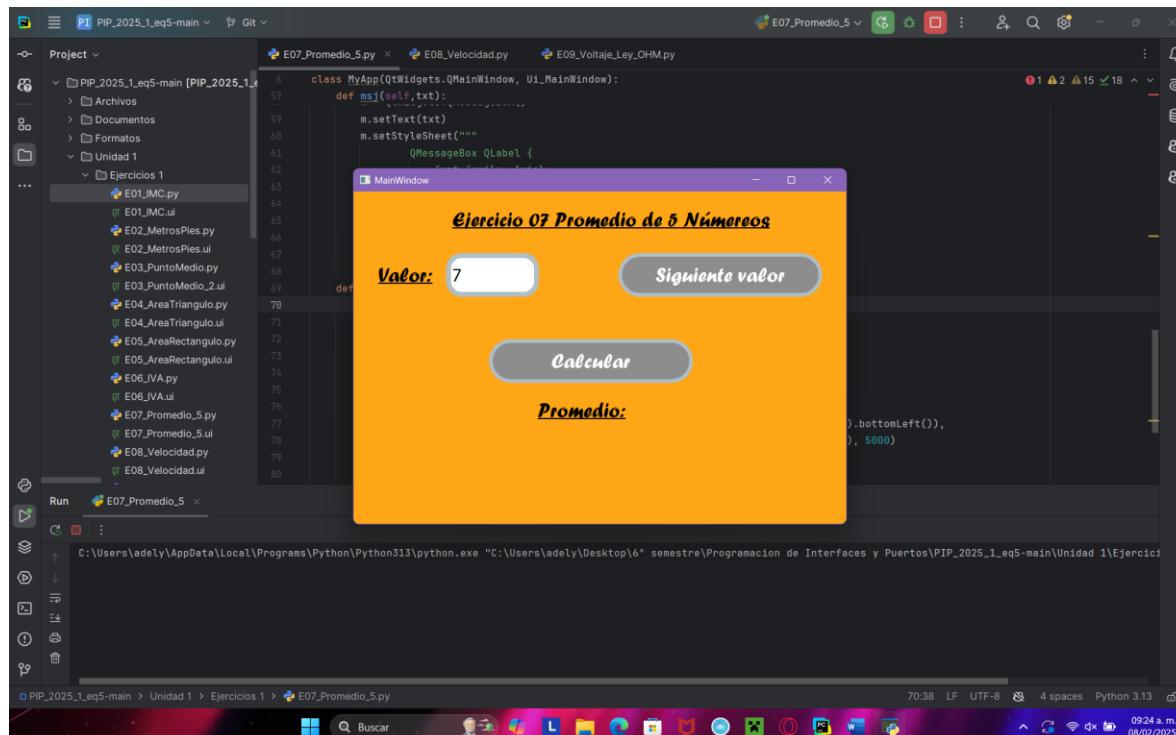
Para revisar y asegurarse que en “txt\_Vvalor” no haya valores inválidos. Guarda el valor, lo cambia a float, si hay un error significa que no es un numero, por lo que no sea valido, al no ser valido, muestra un QToolTip que es como un mensajito chico que le dice que “solo se permiten números”, después borra el ultimo digito introducido para que no vuelva a haber ese error.



Vista principal del programa

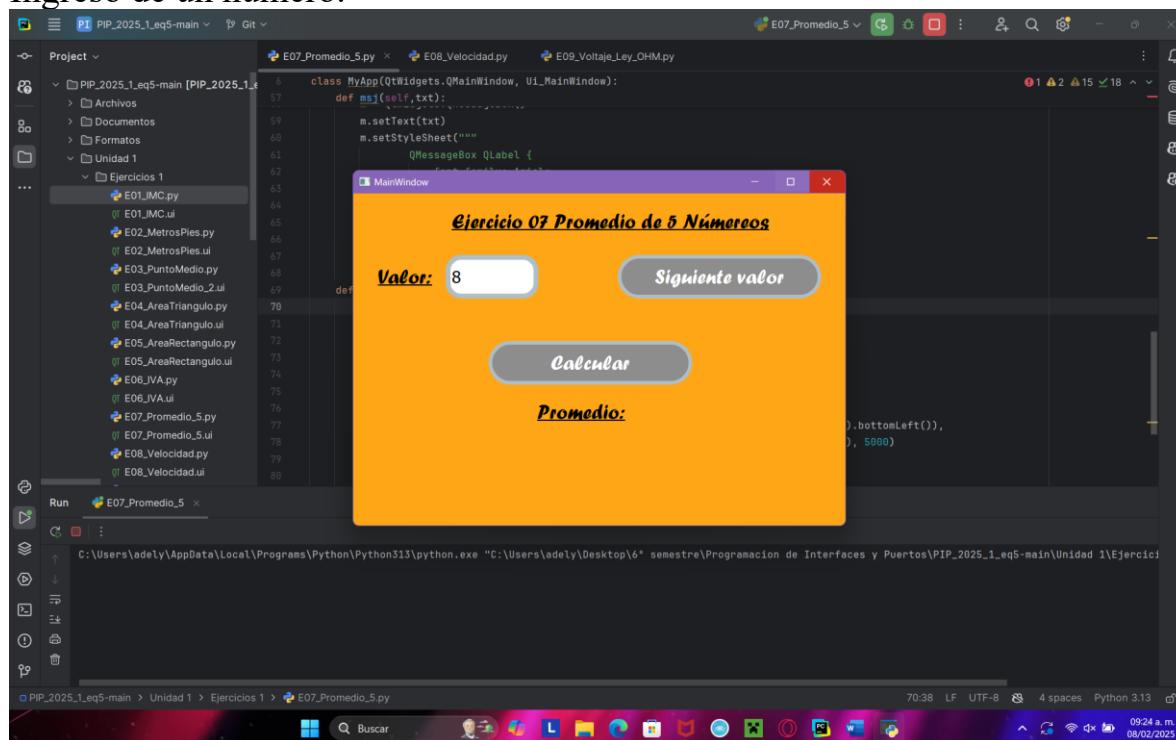


Aviso cuando se presiona una tecla que no es un numero



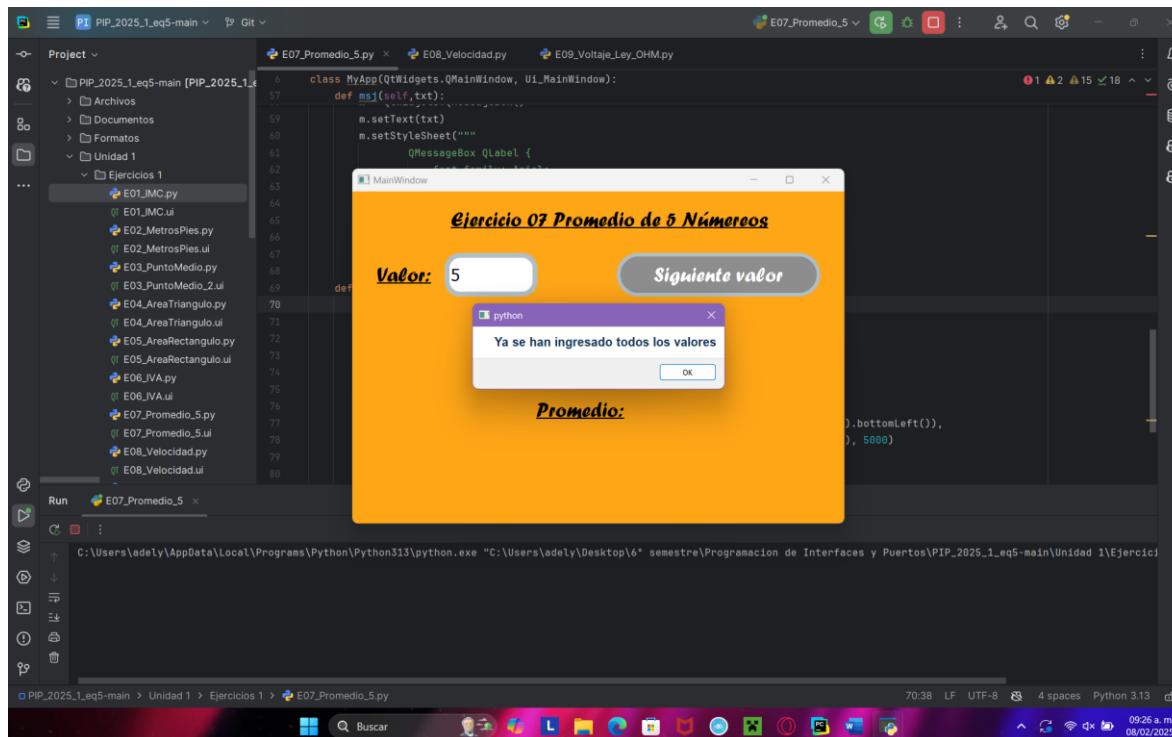
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self, txt):
        super().__init__()
        self.setupUi(self)
        self.setWindowTitle("Ejercicio 07 Promedio de 5 Números")
        self.txt = txt
        self.txt.setPlaceholderText("Valor:")
        self.txt.setText("7")
        self.txt.selectAll()
        self.txt.setFocus()
        self.txt.setStyleSheet("background-color: #f0f0f0; border: 1px solid #ccc; padding: 5px; border-radius: 10px; width: 150px; height: 30px; margin-bottom: 10px; font-size: 14px; color: black; font-weight: bold; text-align: center; border: none; outline: none; font-family: sans-serif; font-style: italic; font-weight: bold; text-decoration: underline; text-decoration-color: black; text-decoration-style: underline; text-decoration-width: 2px; text-decoration-position: under;">Value: 7
```

Ingreso de un numero.

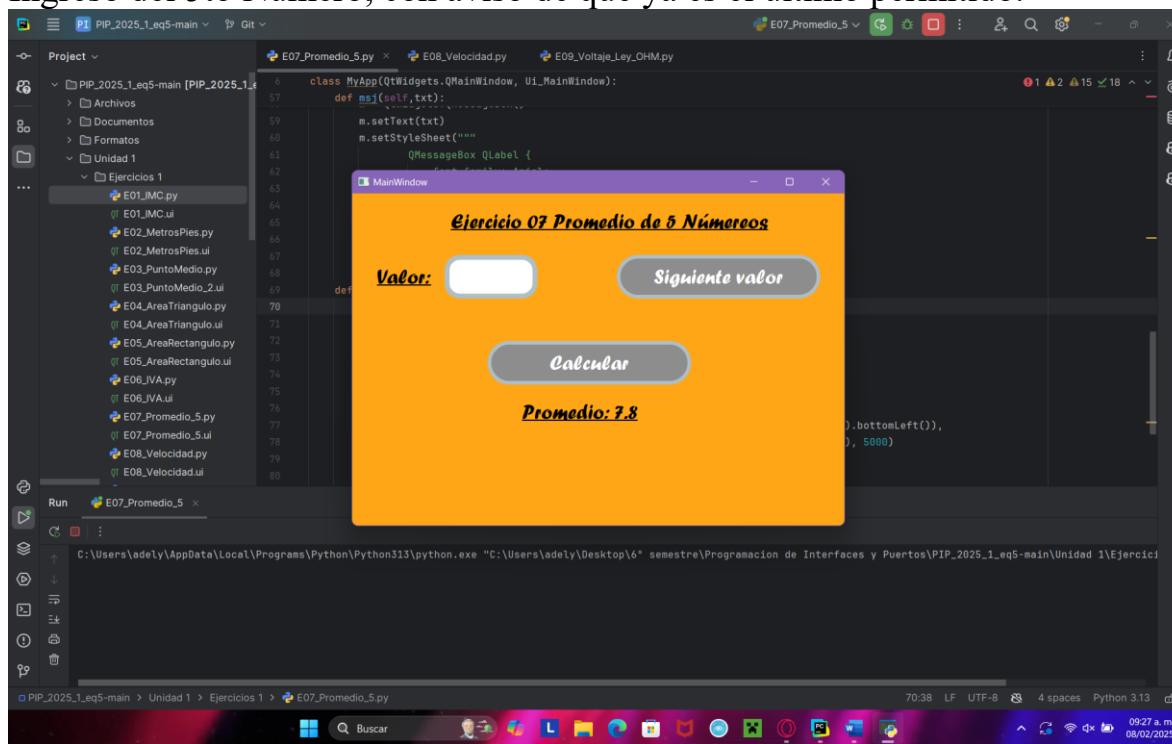


```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self, txt):
        super().__init__()
        self.setupUi(self)
        self.setWindowTitle("Ejercicio 07 Promedio de 5 Números")
        self.txt = txt
        self.txt.setPlaceholderText("Valor:")
        self.txt.setText("8")
        self.txt.selectAll()
        self.txt.setFocus()
        self.txt.setStyleSheet("background-color: #f0f0f0; border: 1px solid #ccc; padding: 5px; border-radius: 10px; width: 150px; height: 30px; margin-bottom: 10px; font-size: 14px; color: black; font-weight: bold; text-align: center; border: none; outline: none; font-family: sans-serif; font-style: italic; font-weight: bold; text-decoration: underline; text-decoration-color: black; text-decoration-style: underline; text-decoration-width: 2px; text-decoration-position: under;">Value: 8
```

Ingreso de otro numero (hasta el ultimo que deja x<sub>5</sub>)



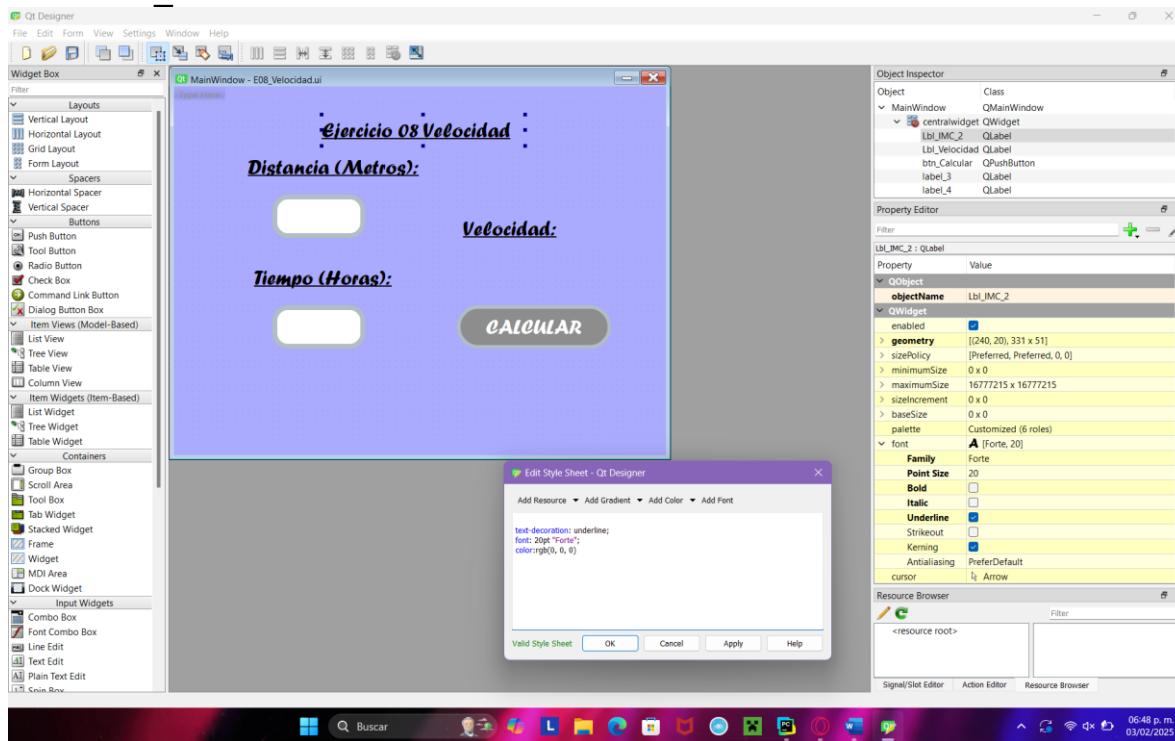
Ingreso del 5to Número, con aviso de que ya es el último permitido.



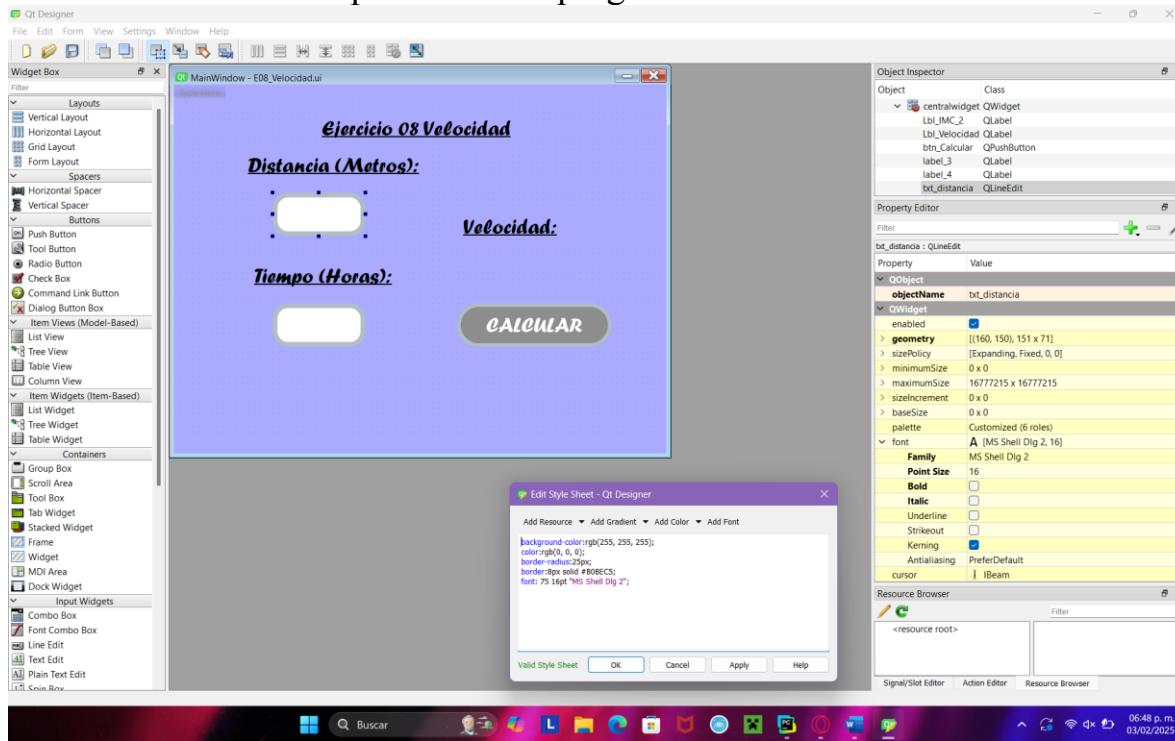
Acción al pulsar “Calcular”, se muestra el promedio.



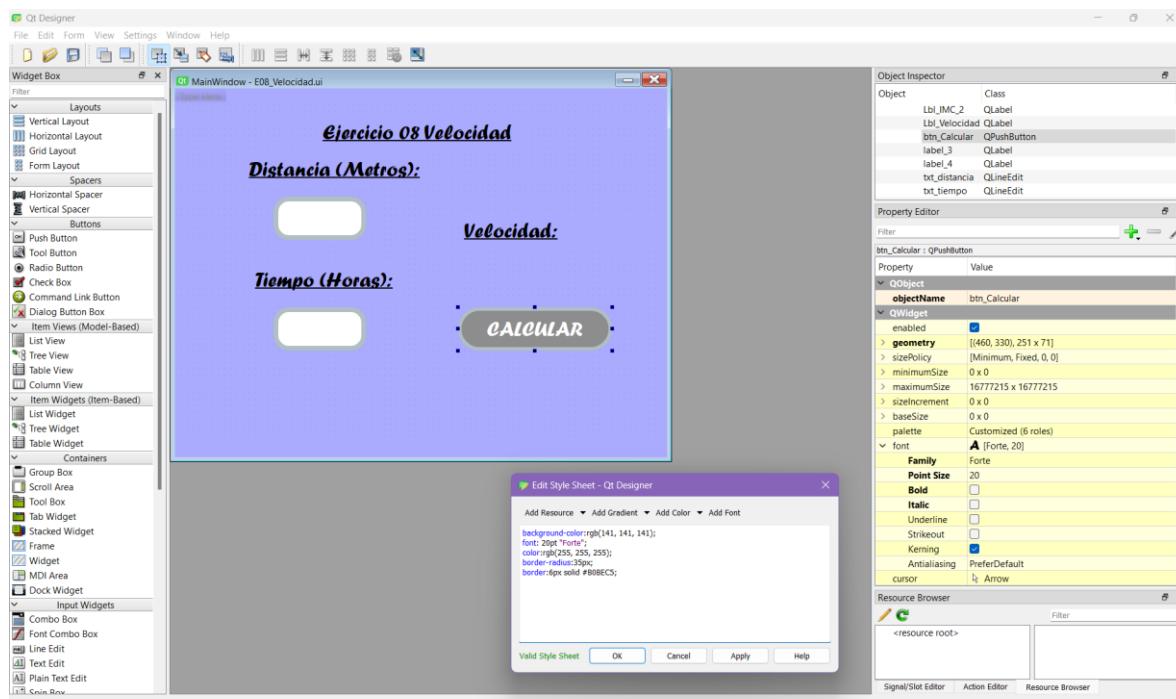
## C17. E08\_Velocidad



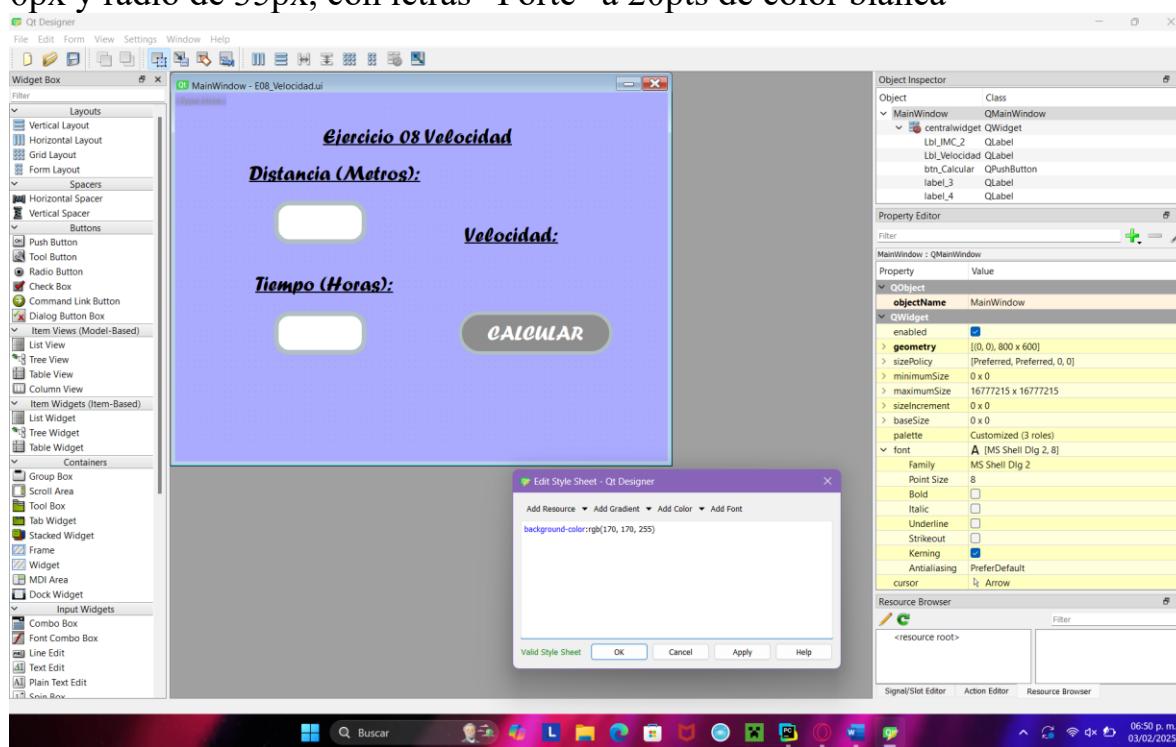
Estilo usado en las Etiquetas de este programa.



Estilo usado en los botones de este programa, color blanco de fondo, borde de 8 puntos y radio a 25px, letra blanca a 16pt y “MS SHELL DIG”



Estilo Usado en los botones de este programa, Color gris de fondo, borde de 6px y radio de 35px, con letras “Forte” a 20pts de color blanca



Estilo de la ventana principal, con un color morado



The screenshot shows the PyCharm IDE interface with the following details:

- Project:** PIP\_2025\_1\_eq5-main
- File:** E08\_Velocidad.py
- Code Preview:** The code defines a class `MyApp` that inherits from `QtWidgets.QMainWindow` and `Ui_MainWindow`. It contains a constructor, a `calcular` method, and event handlers for textChanged signals from two `QLineEdit` objects.
- Code Content:**

```
import sys
from PyQt5 import uic, QtWidgets, QtCore
qtCreatorFile = "E08_Velocidad.ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.btn_Calcular.clicked.connect(self.calcular)
        self.txt_distancia.textChanged.connect(self.CheckLetter0)
        self.txt_tiempo.textChanged.connect(self.CheckLetter1)

    def calcular(self):
        try:
            distancia = float(self.txt_distancia.text())
            print(distancia)
            tiempo = float(self.txt_tiempo.text())
            print(tiempo)
        except:
            self.msgj("Por favor, rellene ambos campos con distancias numéricos")
            return
        if distancia >= 1000:
            distancia = distancia/1000
            velocidad = round(distancia/tiempo,2)
            velocidad = "Velocidad: "+str(velocidad)+" km/h"
        #si es menor a 1km, lo mejor sera darla en m/h
        else:
            velocidad = round(distancia/tiempo,2)
            velocidad = "Velocidad: "+str(velocidad)+" m/h"
        self.lbl_Velocidad.setText(velocidad)

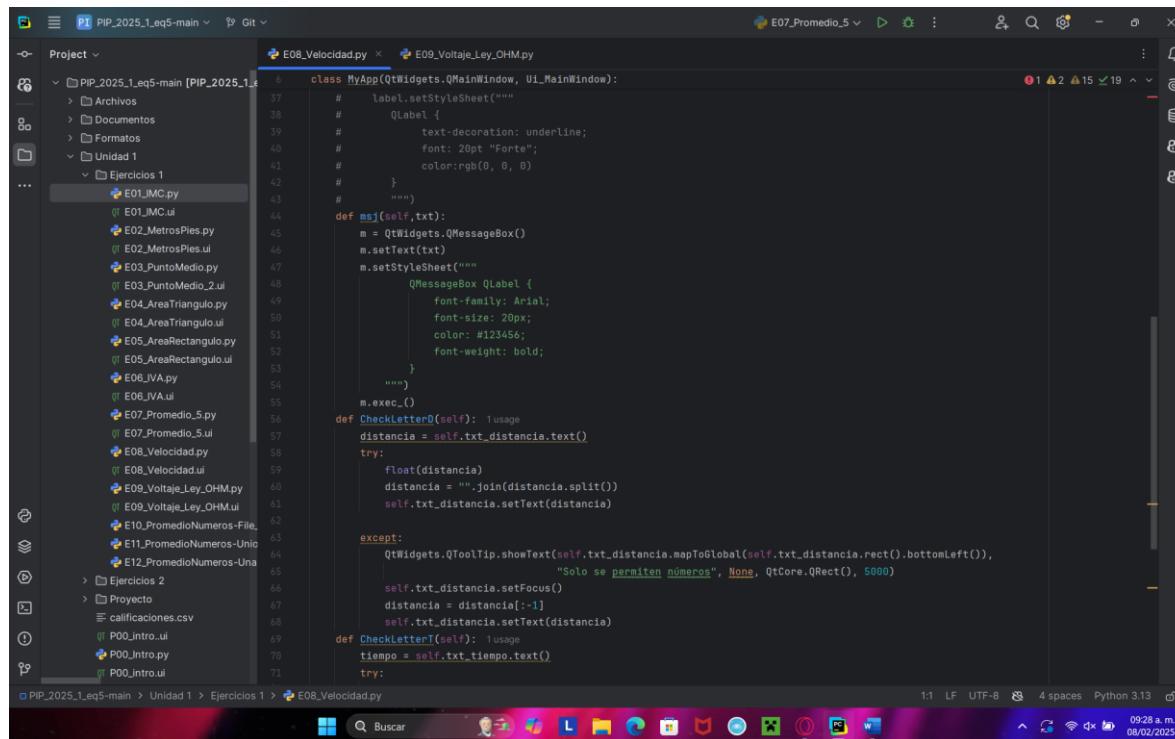
    # def changeStyleSheet(self, label):
```
- Bottom Status Bar:** Shows file encoding (LF), character set (UTF-8), and other system information.

Se asigno a `btn_Calcular` la función (`calcular`), a los 2 `text_edit` se les asigno una función para cuando cambie el valor de “text” perteneciente al suyo.

Calcular:

Lee ambos `text_Edit`, si falla esta por lo menos uno vacío, por lo que se le pide que rellene ambos.

Si la distancia es mas de 1000 mts, la cambia a Km para una mejor vista, calcula la velocidad con su formula, y se redondea a 2 cifras, para después asignarla al “`Lbl_Velocidad`”



The screenshot shows the PyCharm IDE interface with the following details:

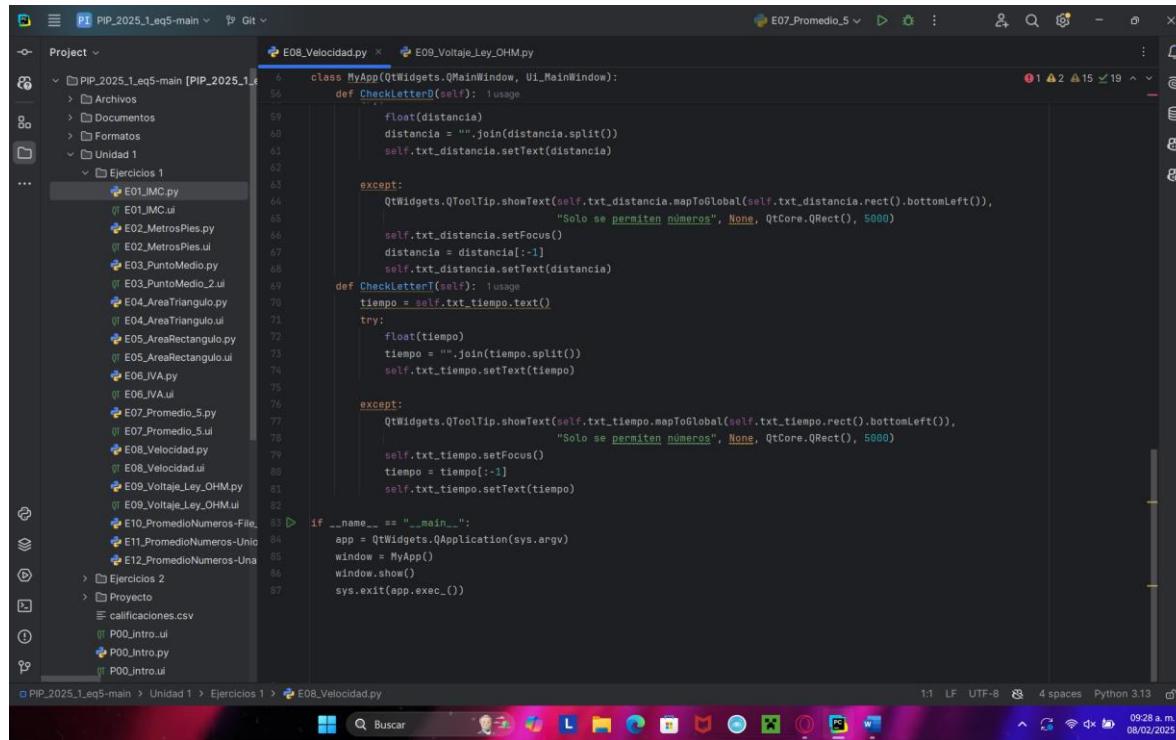
- Project Structure:** The project is named "PIP\_2025\_1\_eq5-main". It contains a folder "PIP\_2025\_1\_eq5-main [PIP\_2025\_1\_eq5-main]" which includes "Archivos", "Documentos", "Formatos", "Unidad 1", and "Ejercicios 1". "Ejercicios 1" contains files like E01\_IMC.py, E01\_IMC.ui, E02\_MetrosPies.py, E02\_MetrosPies.ui, E03\_PuntoMedio.py, E03\_PuntoMedio\_2.ui, E04\_AreaTriangulo.py, E04\_AreaTriangulo.ui, E05\_AreaRectangulo.py, E05\_AreaRectangulo.ui, E06\_IVA.py, E06\_IVA.ui, E07\_Promedio\_5.py, E07\_Promedio\_5.ui, E08\_Velocidad.py, E08\_Velocidad.ui, E09\_Voltaje\_Ley\_OHM.py, E09\_Voltaje\_Ley\_OHM.ui, E10\_PromedioNumeros.py, E11\_PromedioNumeros\_Unico.py, and E12\_PromedioNumeros\_Una.py.
- Code Editor:** The current file is "E08\_Velocidad.py". The code defines a class `MyApp` that inherits from `QtWidgets.QMainWindow` and `Ui\_MainWindow`. It includes methods for handling user input and displaying messages using `QMessageBox`.
- Status Bar:** Shows the file path "PIP\_2025\_1\_eq5-main > Unidad 1 > Ejercicios 1 > E08\_Velocidad.py", the line number "71", and the Python version "Python 3.13".
- System Tray:** Shows icons for battery, signal, and date/time "09:28 a.m. 08/02/2025".

Msj:

Muestra mensaje emergente.

CheckLetterD y CheckLetterT:

Toman el valor de su txt\_Edit respectivo (sea el de distancia o tiempo) revisan si el valor es un numero, si NO es un numero se Avisa que solo introduzca números y se le quita el ultimo valor agregado, si ES un numero, se asegura de quitar espacios y “enters”



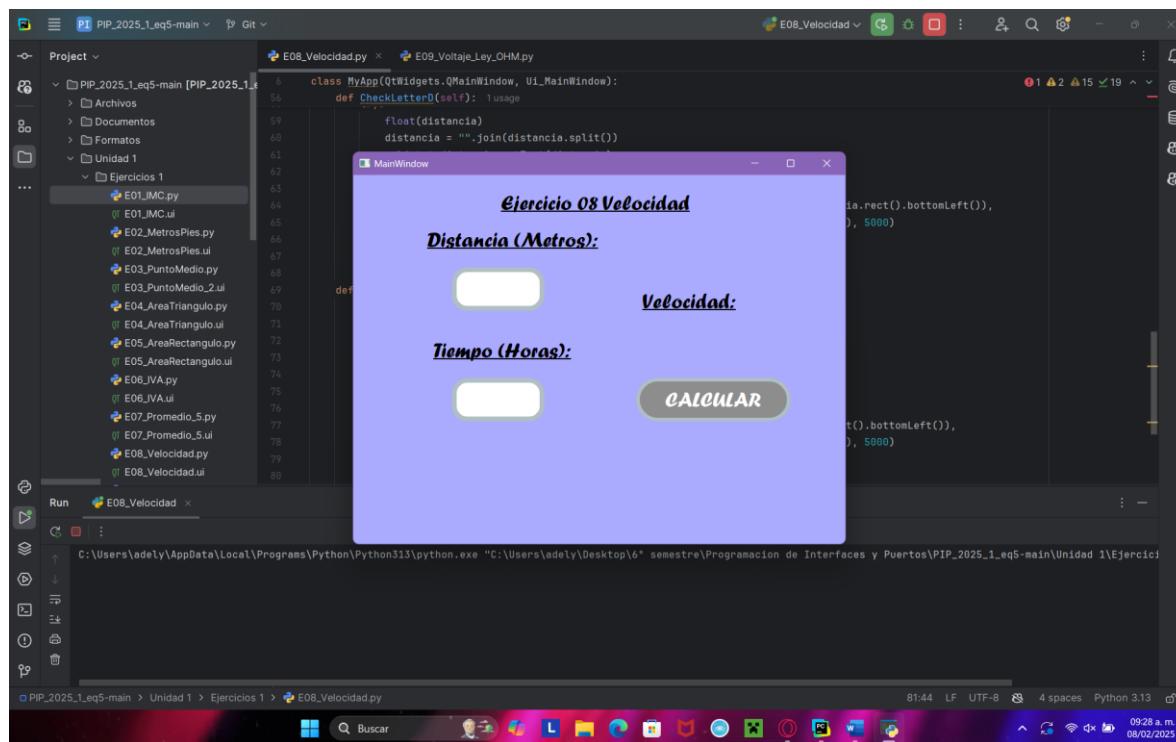
```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def CheckLetterD(self):
        usage
        float(distancia)
        distancia = ''.join(distancia.split())
        self.txt_distancia.setText(distancia)

    except:
        QtWidgets.QToolTip.showText(self.txt_distancia.mapToGlobal(self.txt_distancia.rect().bottomLeft()),
                                    "Solo se permiten números", None, QtCore.QRect(), 5000)
        self.txt_distancia.setFocus()
        distancia = distancia[:-1]
        self.txt_distancia.setText(distancia)

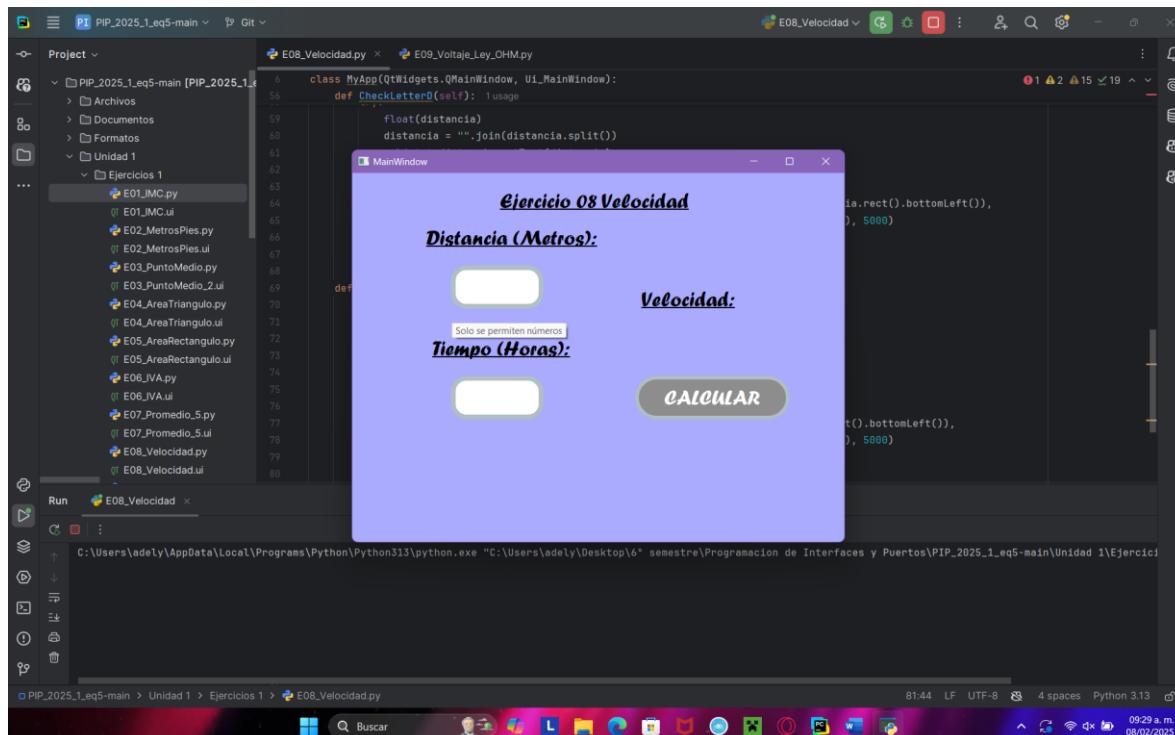
    def CheckLetterT(self):
        usage
        tiempo = self.txt_tiempo.text()
        try:
            float(tiempo)
            tiempo = ''.join(tiempo.split())
            self.txt_tiempo.setText(tiempo)

        except:
            QtWidgets.QToolTip.showText(self.txt_tiempo.mapToGlobal(self.txt_tiempo.rect().bottomLeft()),
                                        "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_tiempo.setFocus()
            tiempo = tiempo[:-1]
            self.txt_tiempo.setText(tiempo)

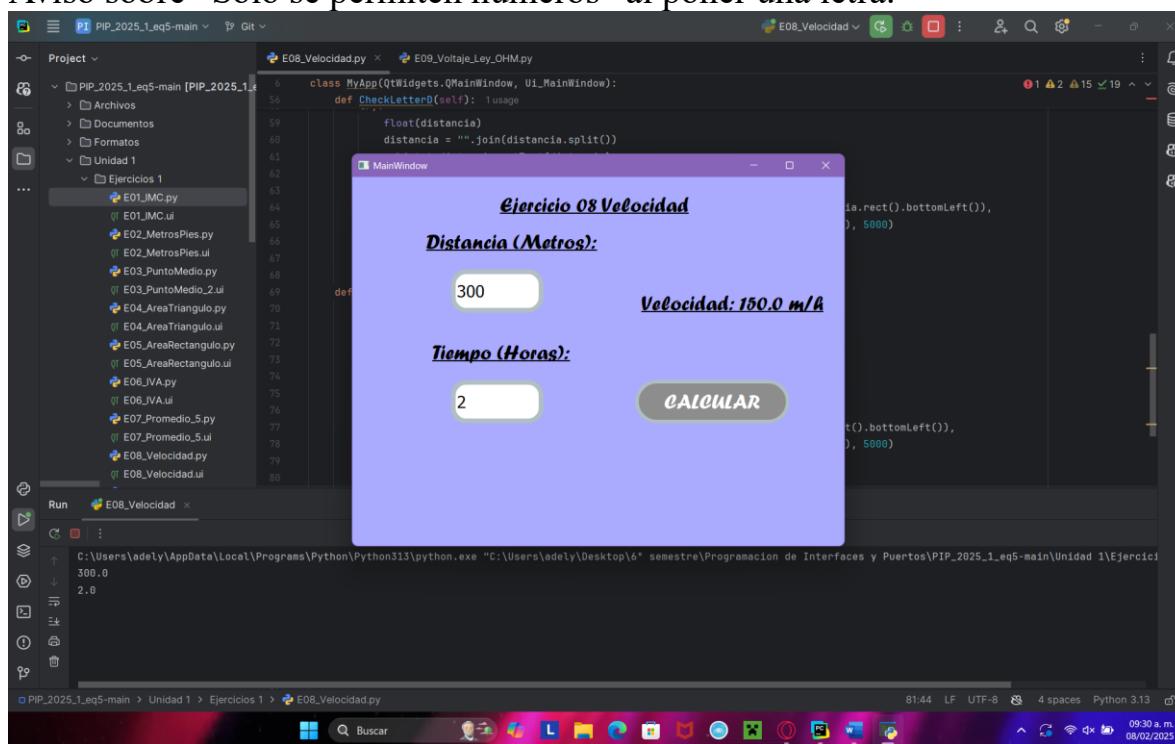
    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```



Vista principal del Programa



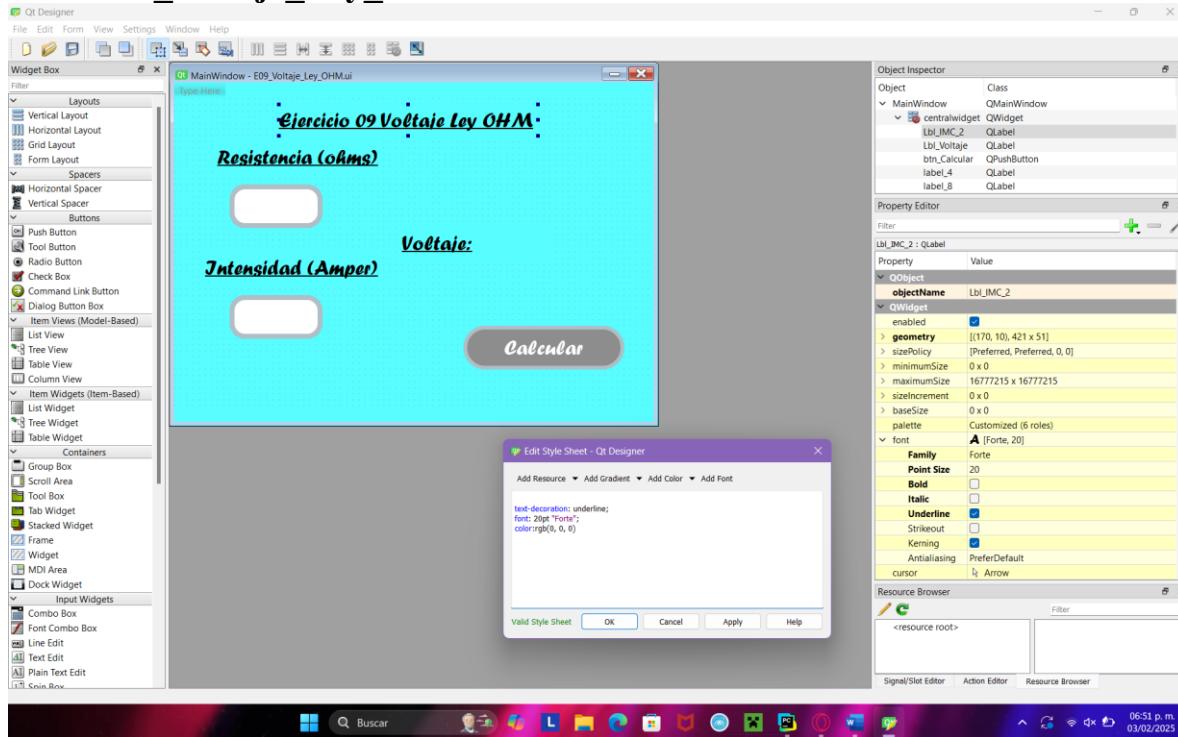
Aviso sobre “Solo se permiten números” al poner una letra.



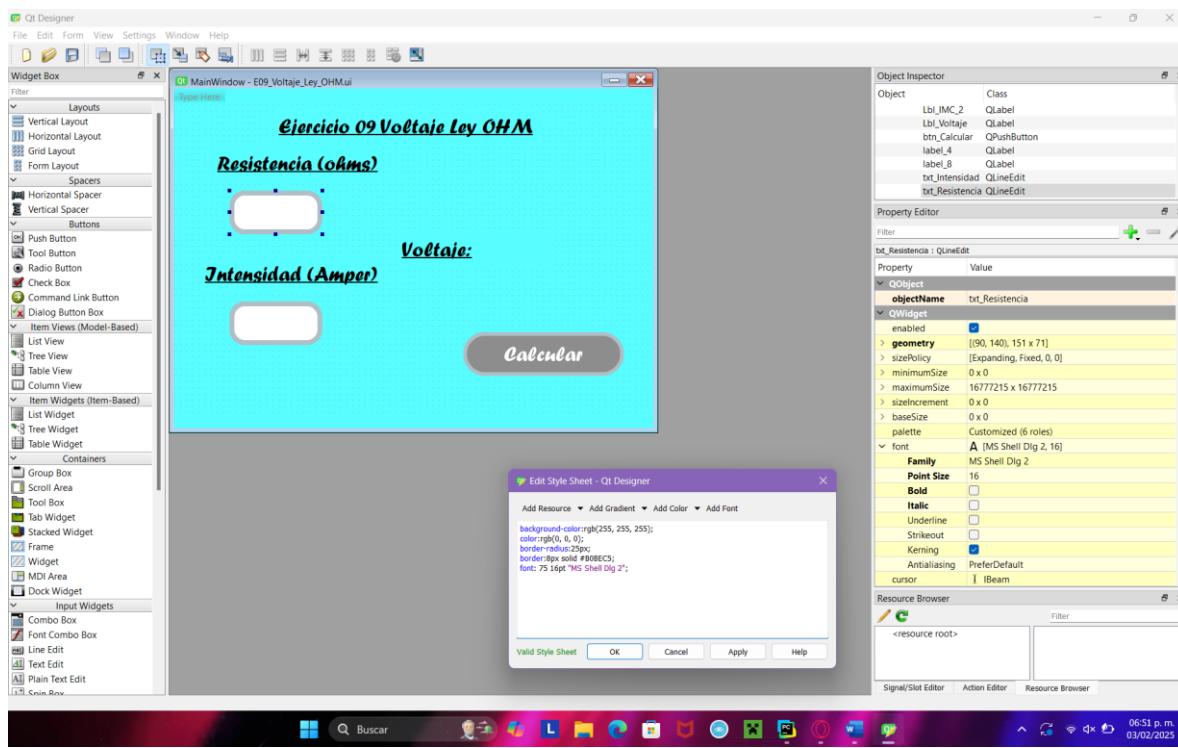
Acción después de presionar “Calcular” mostrando la velocidad



## C18. E09\_Voltaje\_Ley\_OHM

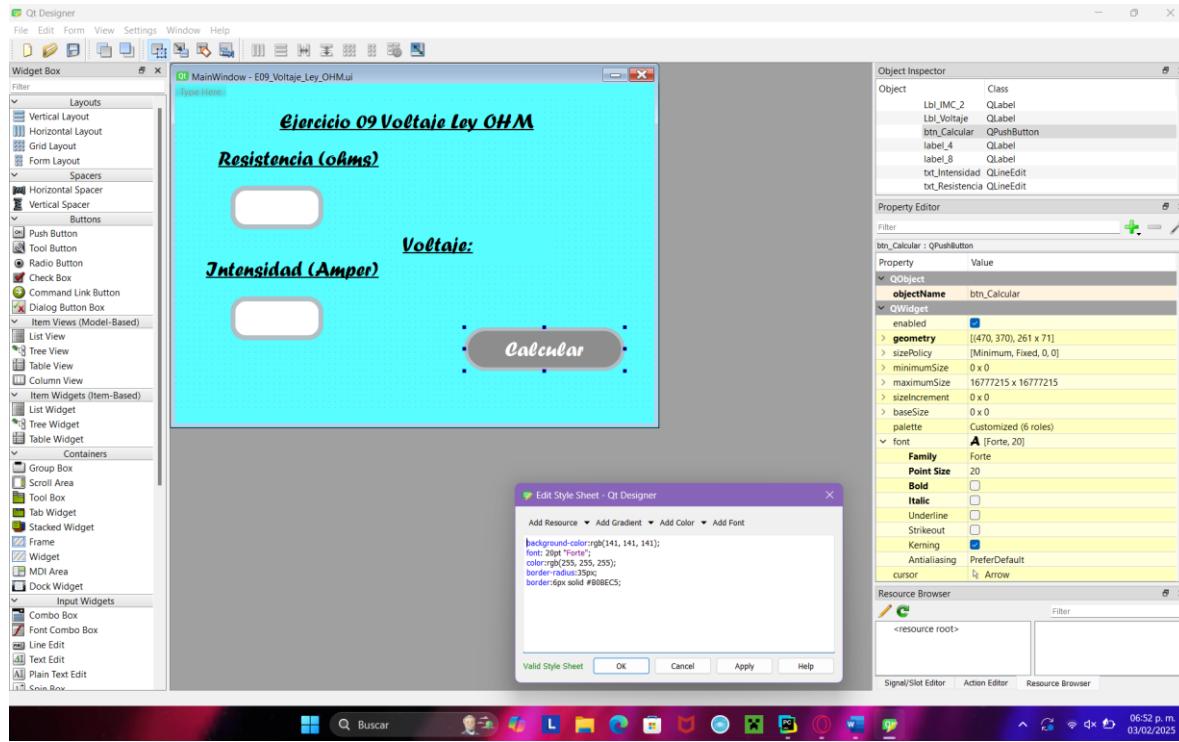


Estilo utilizado en las etiquetas, letra “forte” color blanco y subrayado

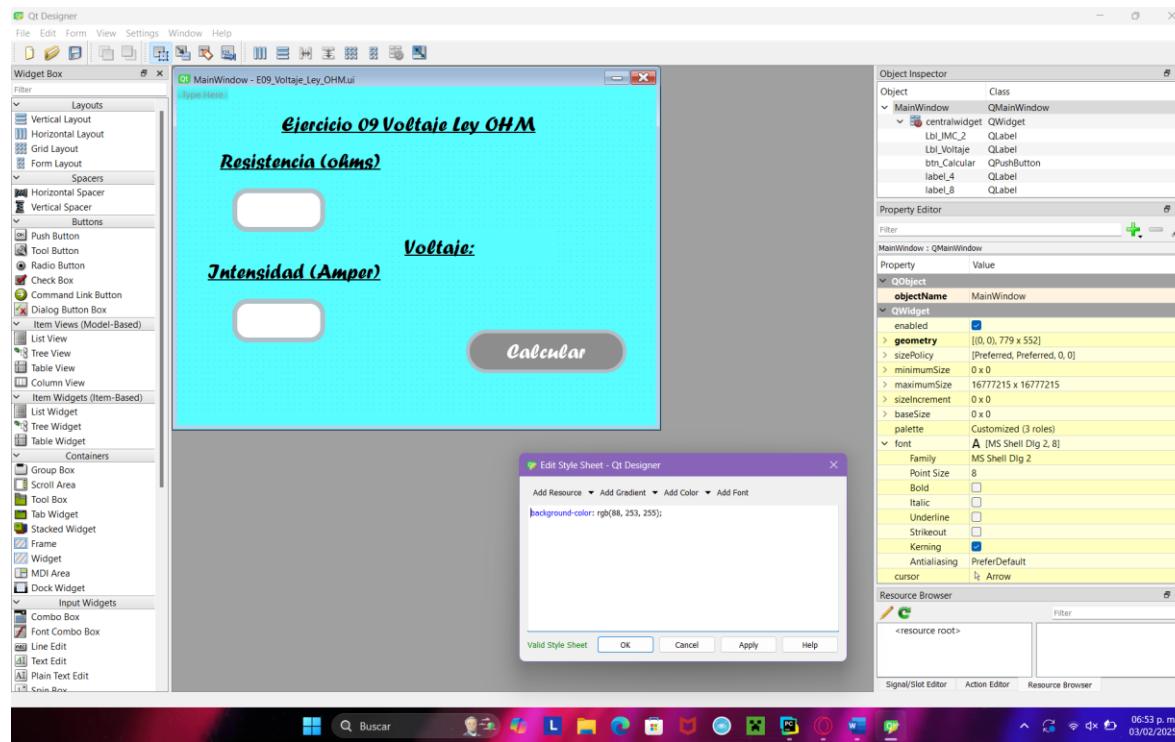




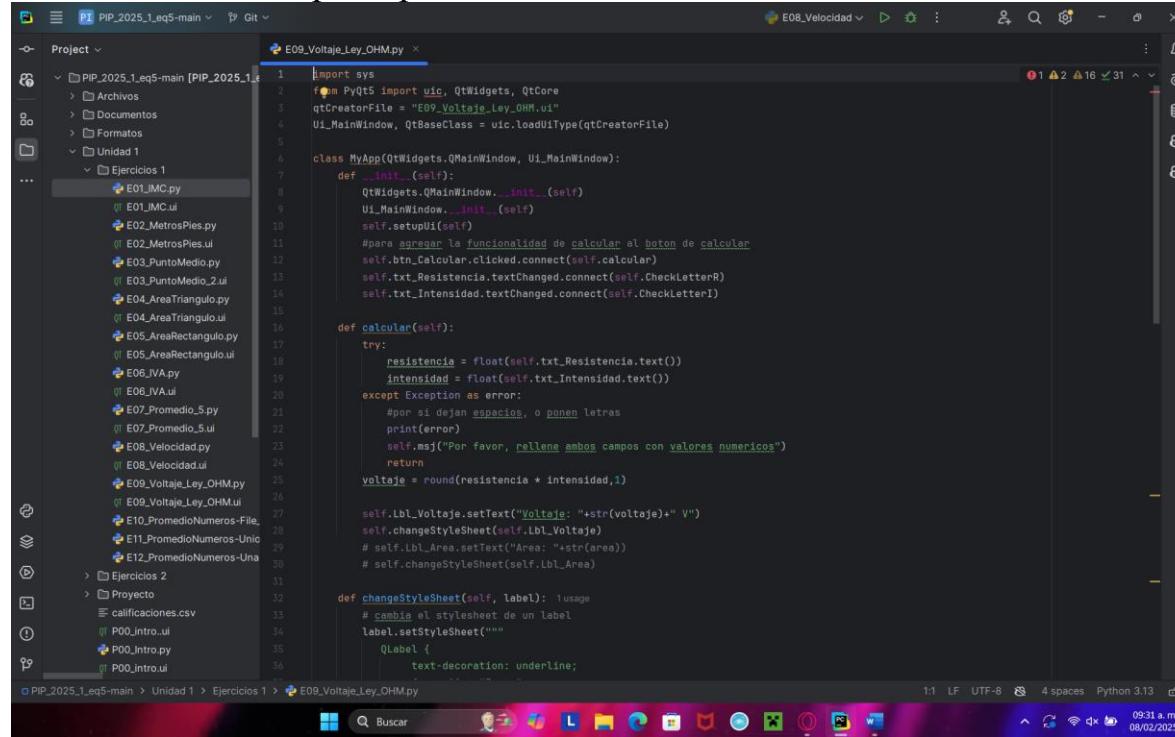
Estilo utilizado en text\_edit, color blanco de fondo, borde 8px a 25 de radio, letra blanca a 16px



Estilo de los botones, gris de fondo, con borde 6px y radio de 35px. Letra “forte” a 20pt



## Color de la ventana principal



```
1 import sys
2 from PyQt5 import uic, QtWidgets, QtCore
3 qtCreatorFile = "E09_Voltaje_Ley_OHM.ui"
4 Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
5
6
7 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
8     def __init__(self):
9         QtWidgets.QMainWindow.__init__(self)
10        Ui_MainWindow.__init__(self)
11        self.setupUi(self)
12        # para agregar la funcionalidad de calcular al boton de calcular
13        self.btn_Calcular.clicked.connect(self.calcular)
14        self.txt_Resistance.textChanged.connect(self.CheckLetterR)
15        self.txt_Intensidad.textChanged.connect(self.CheckLetterI)
16
17    def calcular(self):
18        try:
19            resistencia = float(self.txt_Resistance.text())
20            intensidad = float(self.txt_Intensidad.text())
21        except Exception as error:
22            # por si dejan espacios, o ponen letras
23            print(error)
24            self.msg("Por favor, rellene ambos campos con valores numéricos")
25            return
26        voltaje = round(resistencia * intensidad, 1)
27
28        self.lbl_Voltaje.setText("Voltaje: "+str(voltaje)+" V")
29        self.changeStyleSheet(self.lbl_Voltaje)
30        # self.lbl_Area.setText("Area: "+str(area))
31        # self.changeStyleSheet(self.lbl_Area)
32
33    def changeStyleSheet(self, label): # usage
34        # cambia el stylesheet de un label
35        label.setStyleSheet("""
36            QLabel {
37                text-decoration: underline;
38            }
39        """)
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
```

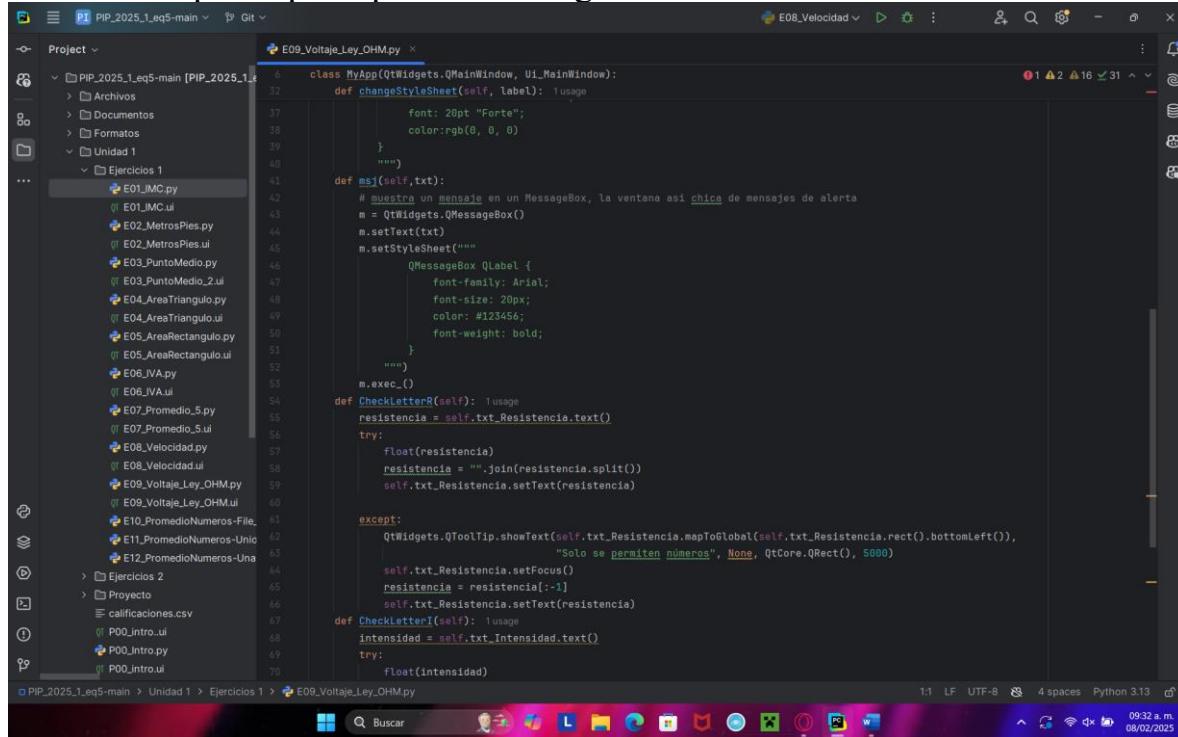
Asignacion al btn\_Calcular su función.

Asignacion de su respectivo método a los 2 txt\_Edit con su método (con textChanged, que hace que se llame al método cuando cambie el texto)



Calcular:

Revisa que ambos valores en los 2 txt\_edit contengan números validos, una vez son validos, calcula el voltaje, lo muestra en una etiqueta y le pone el Style Sheet de etiquetas para que se mantenga la vista

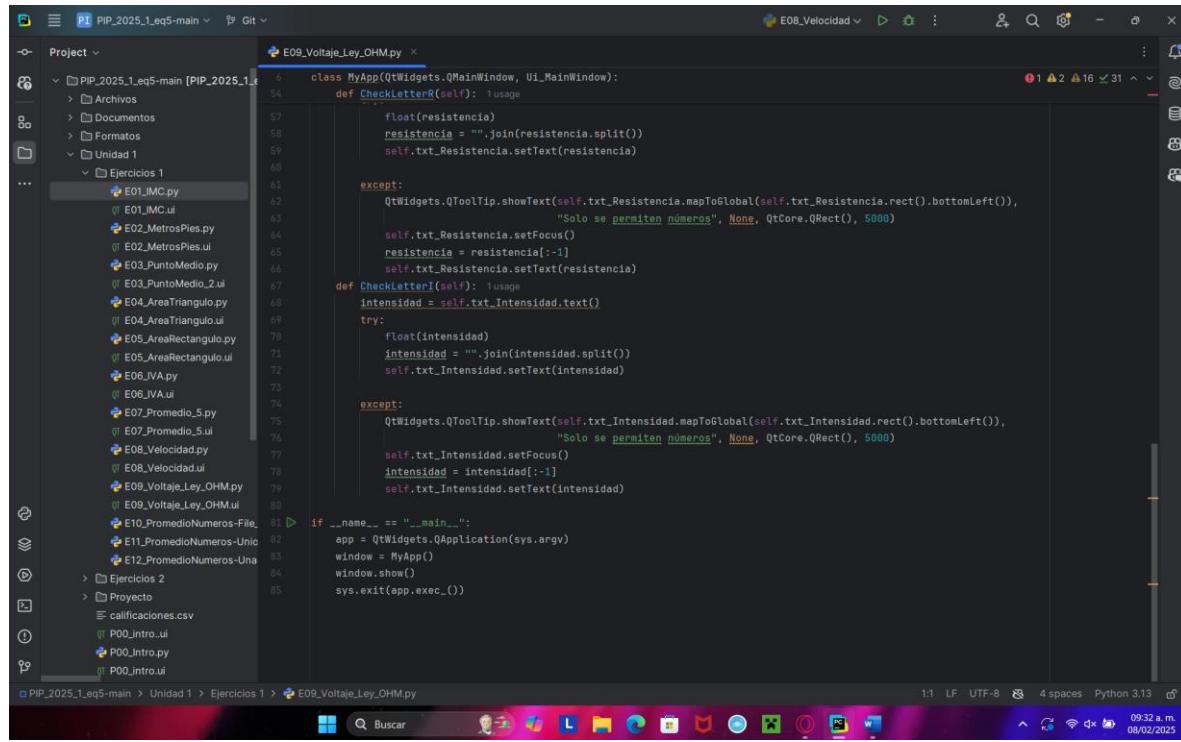


```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def changeStyleSheet(self, label):
        font = QFont("Forte");
        color=rgb(0, 0, 0)
        """
        """
    def msj(self,txt):
        # muestra un mensaje en un QMessageBox, la ventana así chice de mensajes de alerta
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.setStyleSheet("""
            QMessageBox QLabel {
                font-family: Arial;
                font-size: 20px;
                color: #123456;
                font-weight: bold;
            }
        """)
        m.exec_()
    def CheckLetterR(self): 1 usage
        resistencia = self.txt_Resistance.text()
        try:
            float(resistencia)
            resistencia = "".join(resistencia.split())
            self.txt_Resistance.setText(resistencia)
        except:
            QToolTip.showText(self.txt_Resistance.mapToGlobal(self.txt_Resistance.rect().bottomLeft()),
                              "Solo se permiten números", None, Qtcore.QRect(), 5000)
            self.txt_Resistance.setFocus()
            resistencia = resistencia[:-1]
            self.txt_Resistance.setText(resistencia)
    def CheckLetterI(self): 1 usage
        intensidad = self.txt_Intensidad.text()
        try:
            float(intensidad)
```

Msj: Mensaje emergente que muestra el atributo recibido en “txt”

CheckLetterR y CheckLetterI:

De los text\_Edit de Resistencia e Intensidad, guardan el valor que hay, revisan si es un numero, si lo es, quitan los espacios y \n, y lo ponen de nuevo ahí. Si no es numero, muestra alerta de que “solo se permiten números”, regresa el “focus” al txt respectivo, quita el ultimo valor escrito en el text\_edit y lo pone de texto en el mismo



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)

    def CheckLetter(self):
        usage

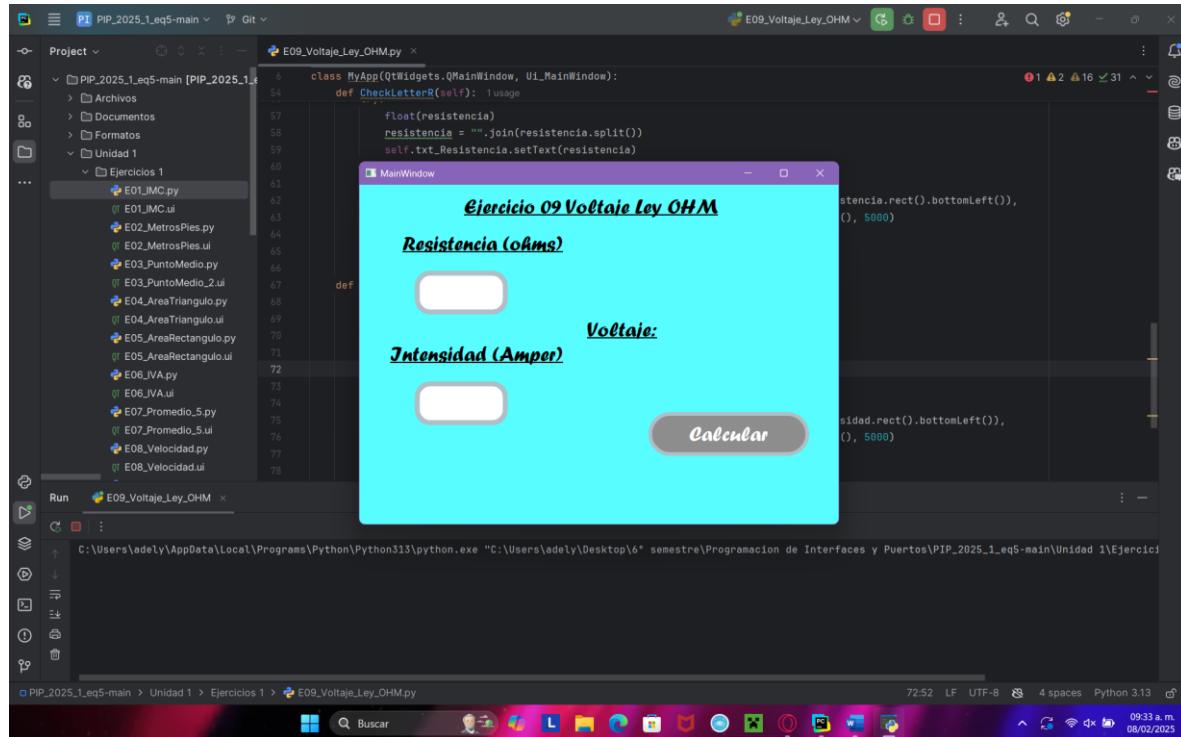
        float(resistencia)
        resistencia = ''.join(resistencia.split())
        self.txt_Resistencia.setText(resistencia)

    except:
        QtWidgets.QToolTip.showText(self.txt_Resistencia.mapToGlobal(self.txt_Resistencia.rect().bottomLeft()),
                                    "Solo se permiten números", None, QtCore.QRect(), 5000)
        self.txt_Resistencia.setFocus()
        resistencia = resistencia[:-1]
        self.txt_Resistencia.setText(resistencia)

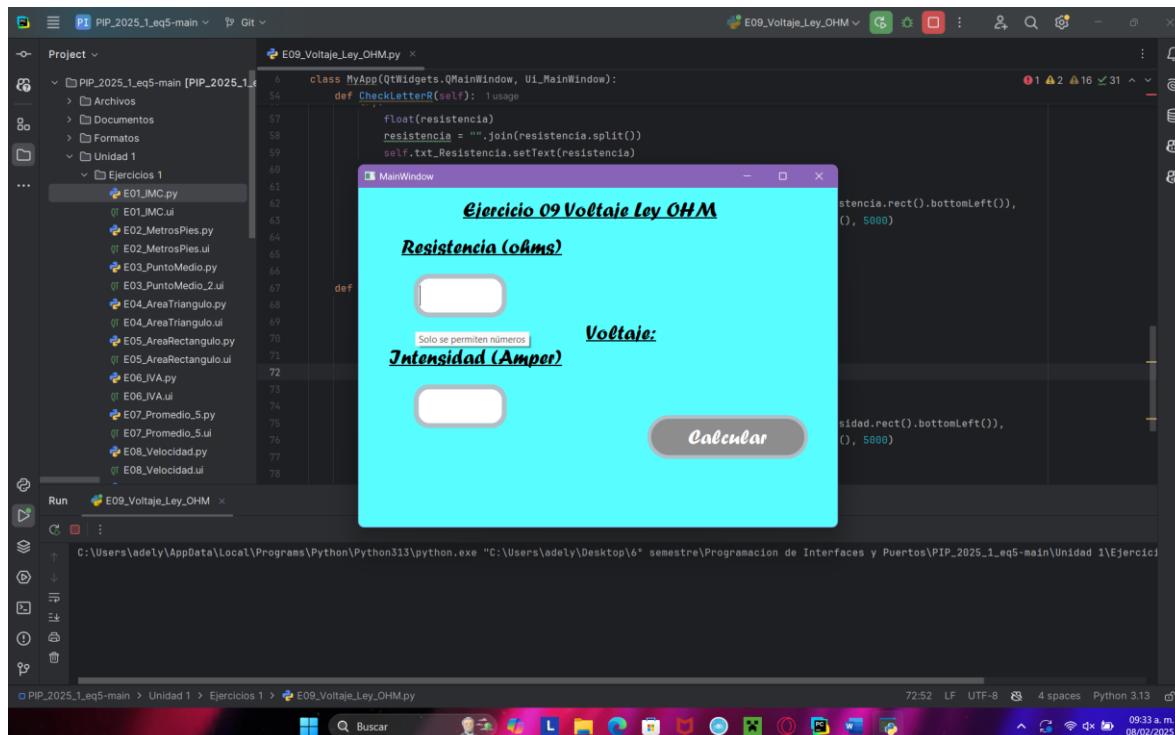
    def CheckLetter(self):
        usage
        intensidad = self.txt_Intensidad.text()
        try:
            float(intensidad)
            intensidad = ''.join(intensidad.split())
            self.txt_Intensidad.setText(intensidad)

        except:
            QtWidgets.QToolTip.showText(self.txt_Intensidad.mapToGlobal(self.txt_Intensidad.rect().bottomLeft()),
                                        "Solo se permiten números", None, QtCore.QRect(), 5000)
            self.txt_Intensidad.setFocus()
            intensidad = intensidad[:-1]
            self.txt_Intensidad.setText(intensidad)

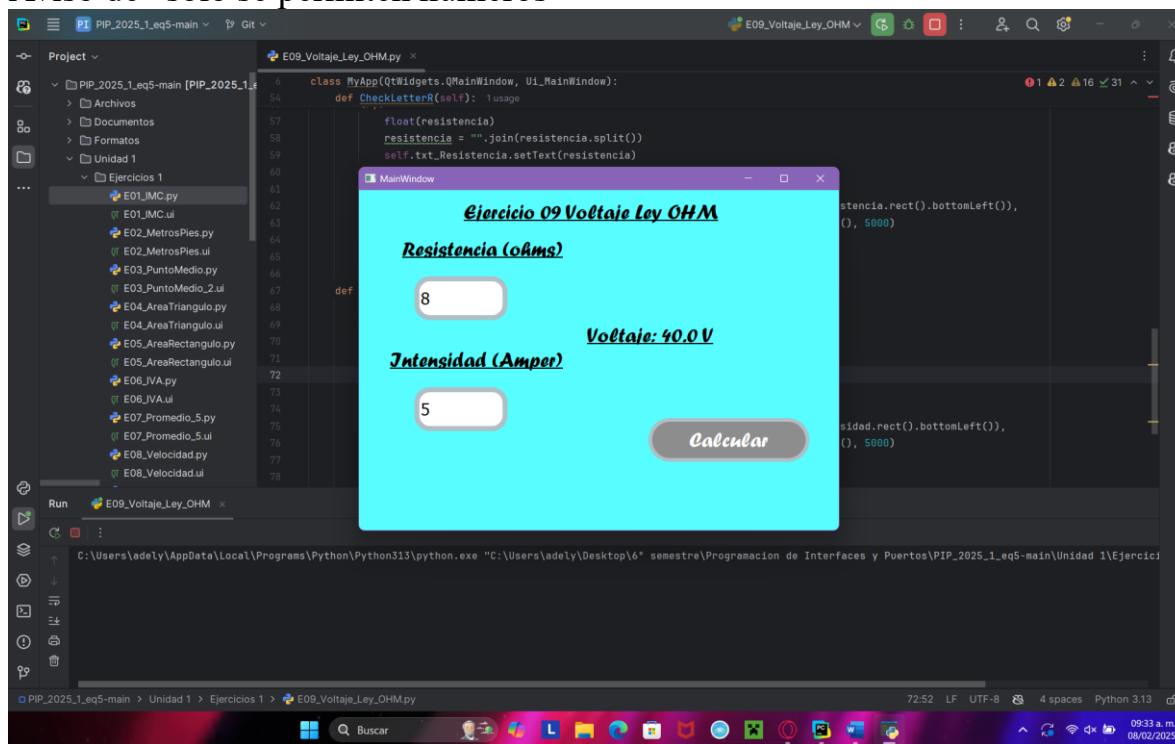
    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```



Vista principal del programa



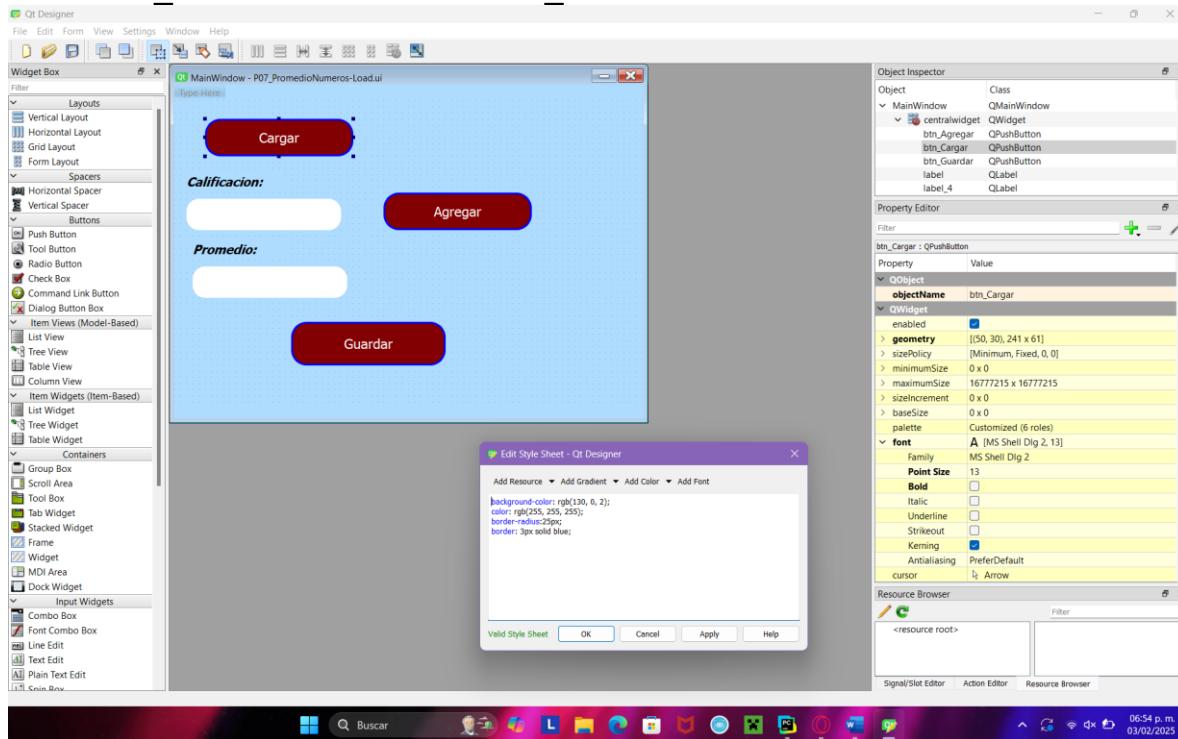
Aviso de “solo se permiten numeros”



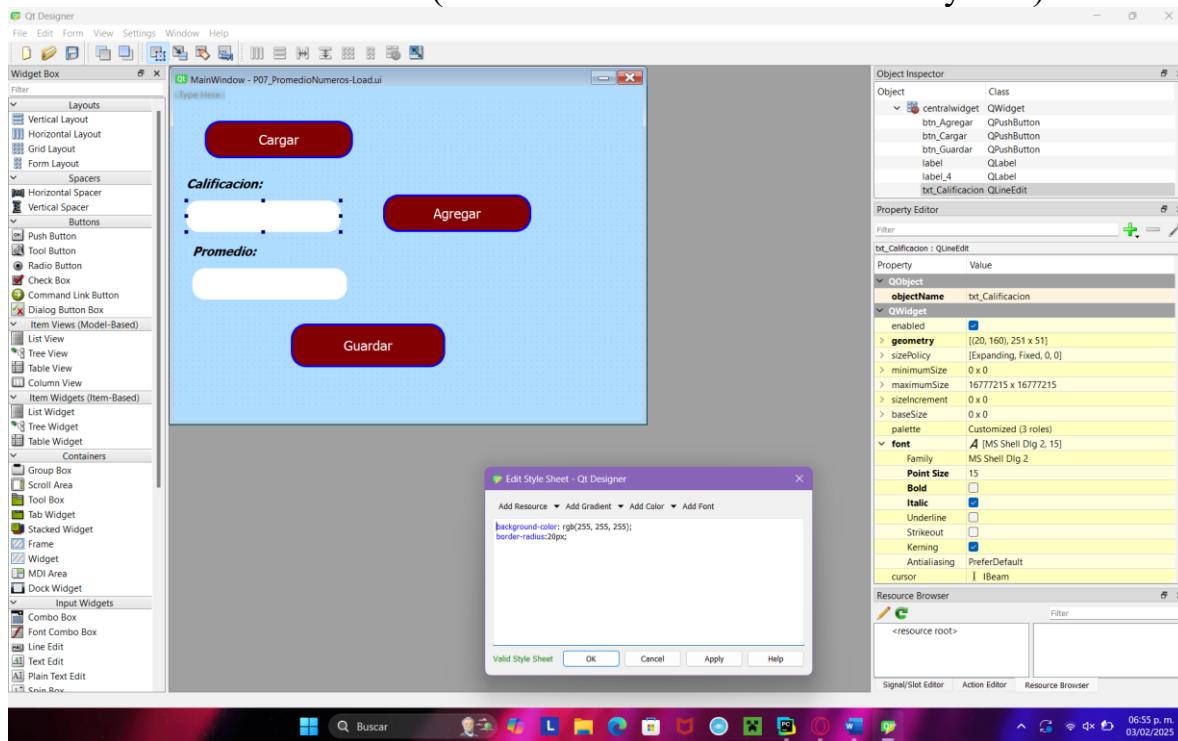
Calculo de el voltaje



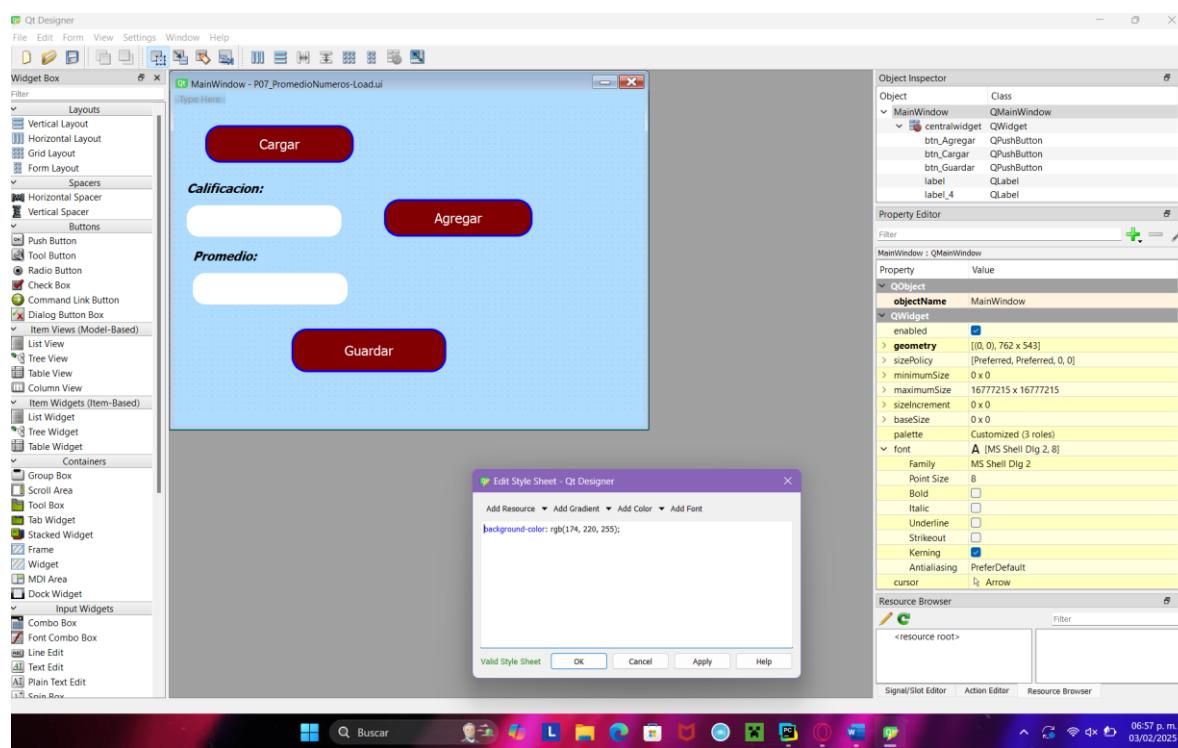
## C19. E10\_PromedioNumeros-File\_check



Estilo utilizado en Botones (Se utilizó el mismo estilo en E11 y E12)



Estilo utilizado en Txt\_edit (Se utilizó el mismo estilo en E11 y E12)



Estilo de la ventana principal (Se utilizo el mismo estilo en E11 y E12)

The screenshot shows a PyCharm IDE interface with several tabs open. The tabs include:

- PIP\_2025\_1\_eq5-main
- Git
- E09\_Voltaje\_Ley\_OHM.py
- E09\_Voltaje\_Ley\_OHMu
- E10\_PromedioNumeros-File
- E11\_PromedioNumeros-Union de proms.py
- E12\_PromedioNumeros-UnaCarga.py
- E01\_JMC.py
- E01\_JMC.ui
- E02\_MetrosPies.py
- E02\_MetrosPies.ui
- E03\_PuntoMedio.py
- E03\_PuntoMedio.ui
- E04\_AreaTriangulo.py
- E04\_AreaTriangulo.ui
- E05\_AreaRectangulo.py
- E05\_AreaRectangulo.ui
- E06\_IVA.py
- E06\_IVA.ui
- E07\_Promedio\_5.py
- E07\_Promedio\_5.ui
- E08\_Velocidad.py
- E08\_Velocidad.ui
- E09\_Voltaje\_Ley\_OHM.py
- E09\_Voltaje\_Ley\_OHMu
- E10\_PromedioNumeros-File
- E11\_PromedioNumeros-Union de proms.py
- E12\_PromedioNumeros-UnaCarga.py

The code editor displays the file `E10_PromedioNumeros-File_check.py` with the following content:

```
1 import sys
2 from PyQt5 import uic, QtWidgets, QtCore
3 qtCreatorFile = "./PO0_PromedioNumeros_Load.ui" #Cambiar el nombre a PO0_o sea el numero 0, no el 0 XD
4 ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
5 class MyApp(QtWidgets.QMainWindow, ui_MainWindow):
6     def __init__(self):
7         QtWidgets.QMainWindow.__init__(self)
8         ui_MainWindow.__init__(self)
9         self.setupUi(self)
10        self.btnAdd_Agregar.clicked.connect(self.Agregar)
11        self.btnAdd_Guardar.clicked.connect(self.Guardar)
12        self.btnAdd_Cargar.clicked.connect(self.Cargar)
13        self.txtCalificacion.textChanged.connect(self.CheckLetterC)
14        self.calificaciones = []
15
16    def Cargar(self): #usase
17        #Comprobar si el archivo existe?
18        #Copiarlo, hacer el ejercicio 10, esto sera el ejercicio 10, lo de revisar que existe
19
20        # archivo = open("../Archivos/calificaciones.csv")
21        try:
22            archivo = open("../..../Archivos/calificaciones.csv")
23        except Exception as error:
24            print(error)
25            self.msj("El archivo no existe")
26            return
27
28        contenido = archivo.readlines()
29        print(contenido)
30        datos = [float(x) for x in contenido]
31        print(datos)
32        #Ejercicio 11
33        #En lugar de Sobreescribir, Concatenar
34        self.calificaciones = datos
35        self.promedio()
36        #Tareas 12
```

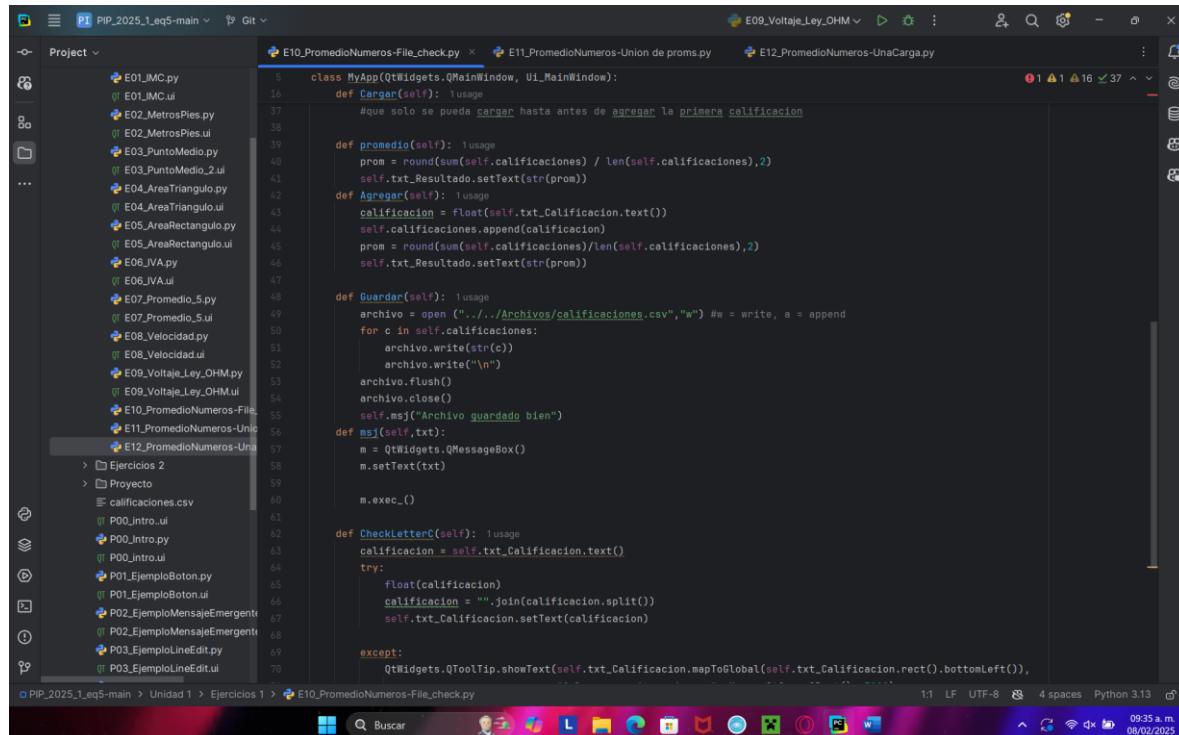
Se le asigno a btn\_agregar, btn\_guardar y btn\_cargar su respectiva función.



Se asigno a txt\_Calificaciones un “textChanged” para cuando se ingrese un valor este lo revise.

Cargar:

Revisa que el archivo exista, esto mediante un try except con un “open”, si al hacer el open, da una excepción, el archivo no existe, por lo que se le avisara al usuario y regresara a partir de ese punto.



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.setupUi(self)
        self.txt_Calificacion.textChanged.connect(self.Cargar)

    def Cargar(self):
        promedio = round(sum(self.calificaciones) / len(self.calificaciones), 2)
        self.txt_Resultado.setText(str(promedio))

    def Agregar(self):
        calificacion = float(self.txt_Calificacion.text())
        self.calificaciones.append(calificacion)
        prom = round(sum(self.calificaciones)/len(self.calificaciones), 2)
        self.txt_Resultado.setText(str(prom))

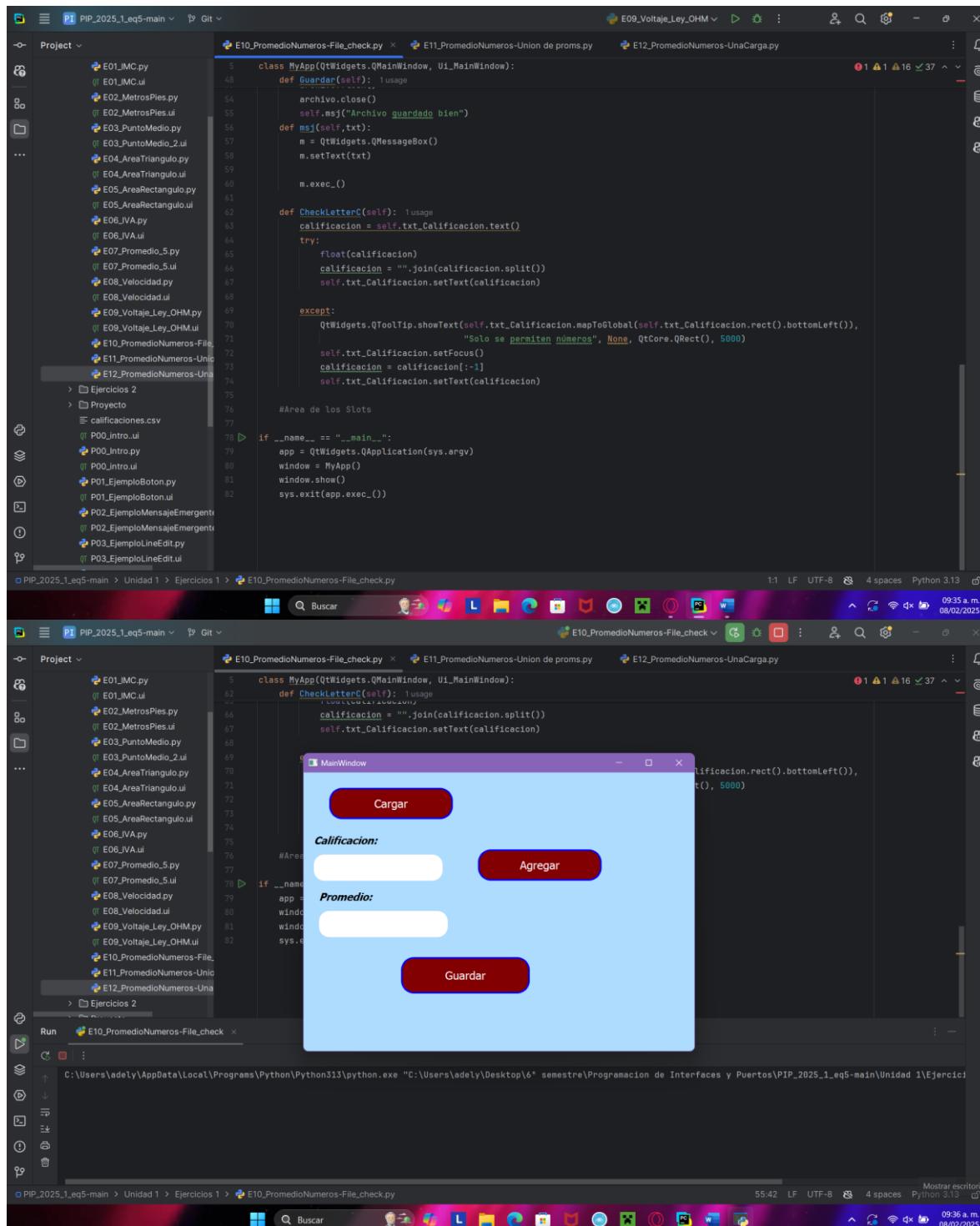
    def Guardar(self):
        archivo = open("../Archivos/calificaciones.csv", "w") # w = write, a = append
        for c in self.calificaciones:
            archivo.write(str(c))
            archivo.write("\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo guardado bien")

    def msj(self, txt):
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

    def CheckLetterC(self):
        calificacion = self.txt_Calificacion.text()
        try:
            float(calificacion)
            calificacion = ''.join(calificacion.split())
            self.txt_Calificacion.setText(calificacion)
        except:
            QtWidgets.QToolTip.showText(self.txt_Calificacion.mapToGlobal(self.txt_Calificacion.rect().bottomLeft()),
```

CheckLetterC:

Se llama cuando cambia una letra en el text\_Edit txt\_Calificacion, revisa que sea un numero, si no lo es, muestra una alerta de que solo se permiten números y borra el ultimo carácter escrito.



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def Guardar(self):
        archivo.close()
        self.msj("Archivo guardado bien")
    def msj(self,txt):
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()

    def CheckLetterC(self):
        if ...name__ == "__main__":
            app = QtWidgets.QApplication(sys.argv)
            window = MyApp()
            window.show()
            sys.exit(app.exec_())

class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Calificación:")
        self.setGeometry(300, 200, 400, 300)
        self.setCentralWidget(self.tabWidget)
        self.tabWidget.setCurrentIndex(0)
        self.tabWidget.currentChanged.connect(self.on_tab_change)

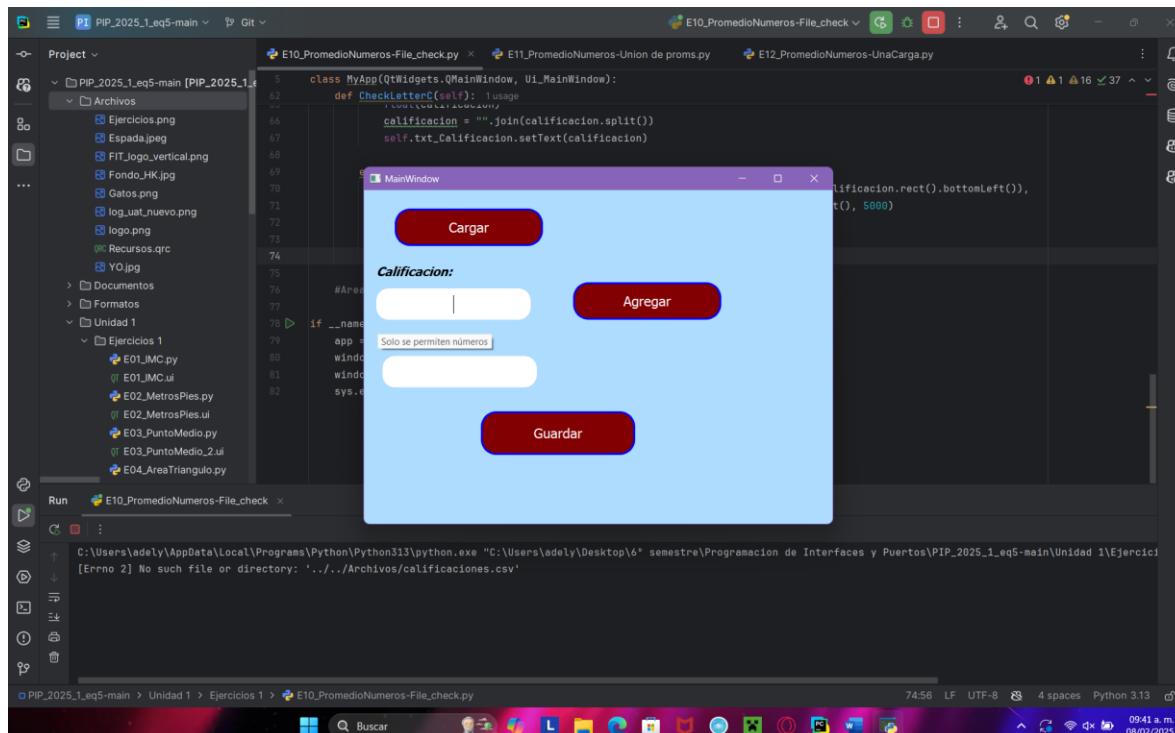
    def on_tab_change(self, index):
        if index == 0:
            self.load_file()
        elif index == 1:
            self.add_file()
        elif index == 2:
            self.save_file()

    def load_file(self):
        file_name = QFileDialog.getOpenFileName(self, "Abrir Archivo", "", "Archivos de texto (*.txt);;Todos los archivos (*.*)")
        if file_name[0]:
            with open(file_name[0], "r") as f:
                data = f.read()
            self.txt_Calificacion.setText(data)

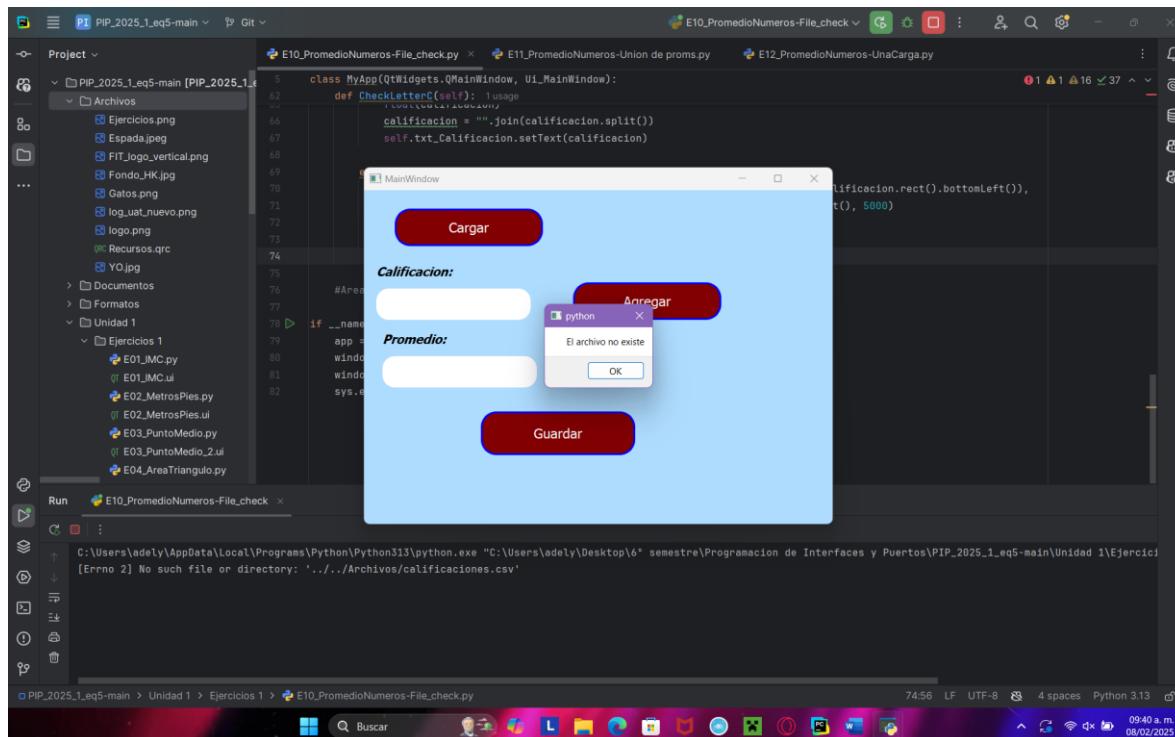
    def add_file(self):
        file_name = QFileDialog.getSaveFileName(self, "Guardar Archivo", "", "Archivos de texto (*.txt);;Todos los archivos (*.*)")
        if file_name[0]:
            with open(file_name[0], "w") as f:
                f.write(self.txt_Calificacion.toPlainText())
            self.msj("Archivo guardado bien")

    def save_file(self):
        file_name = QFileDialog.getSaveFileName(self, "Guardar Archivo", "", "Archivos de texto (*.txt);;Todos los archivos (*.*)")
        if file_name[0]:
            with open(file_name[0], "w") as f:
                f.write(self.txt_Calificacion.toPlainText())
            self.msj("Archivo guardado bien")
```

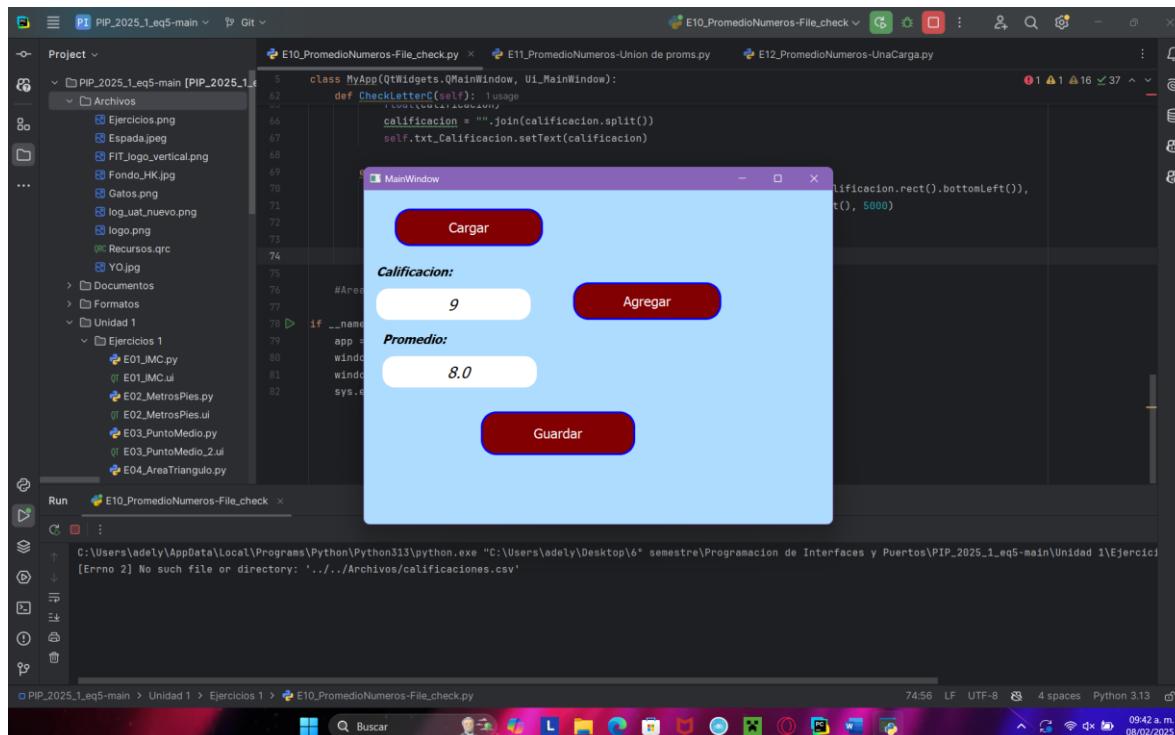
Vista principal del programa



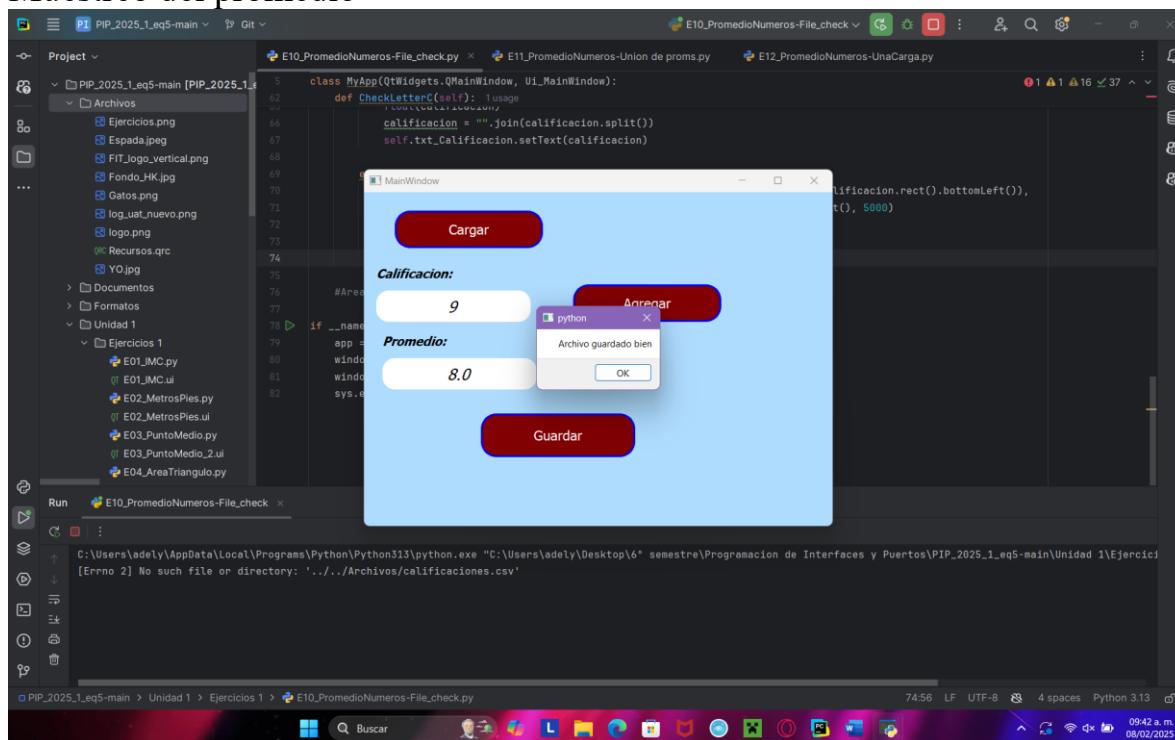
Alerta de cuando se intenta introducir un valor no numérico



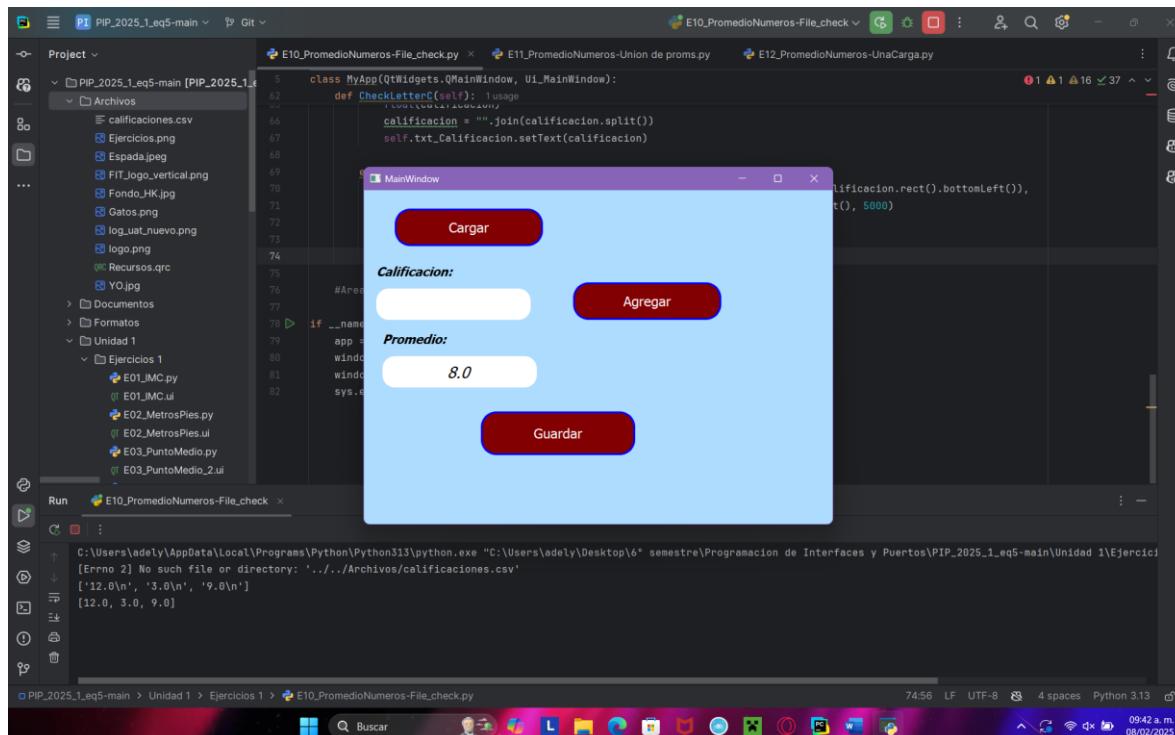
Alerta cuando no existe el archivo de calificaciones



Muestreo del promedio



Guardado del archivo



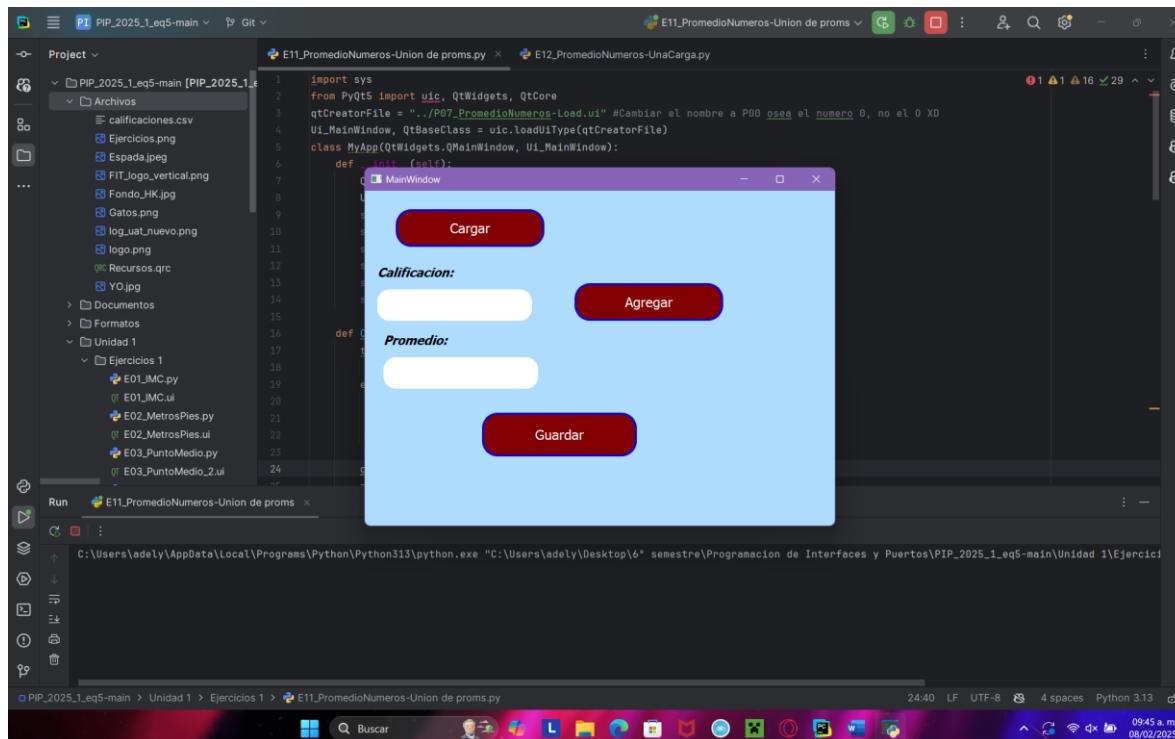
Carga de un archivo



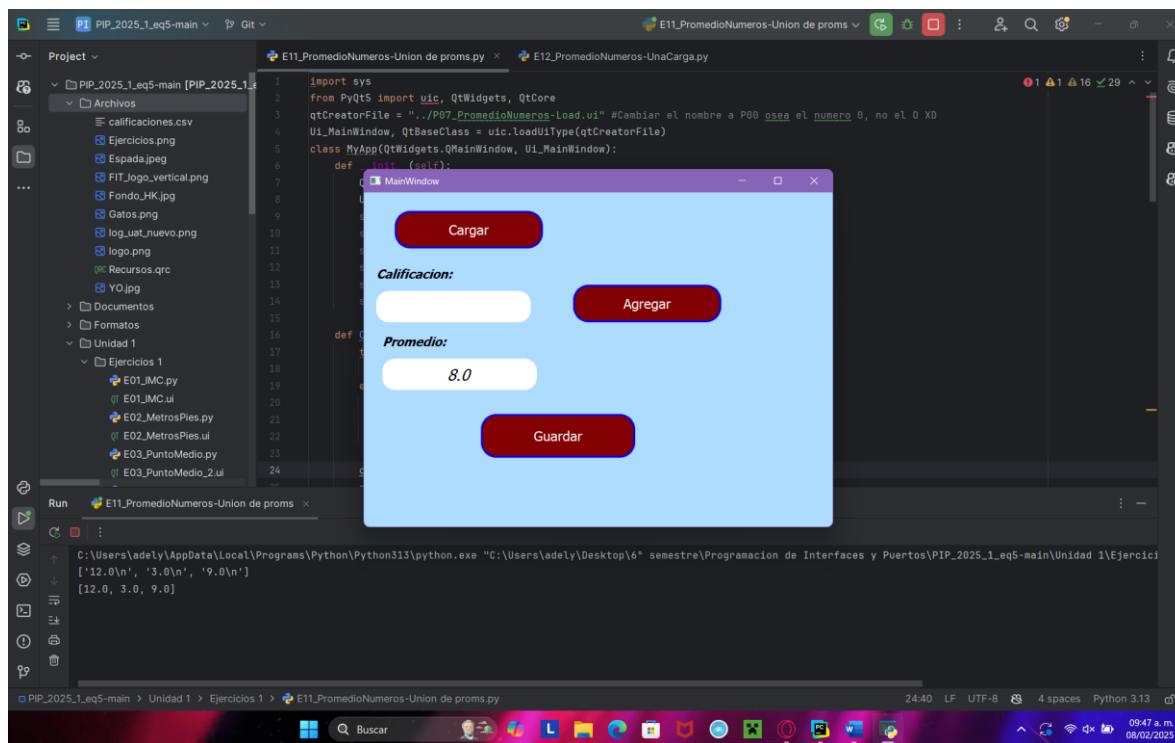
## C20. E11\_PromedioNumeros-Union de proms

Se cambio en la línea 30, de self.calificaciones = datos, a:  
self.calificaciones += datos, logrando así que solo se CONCATENEN a los que ya se tenían, sin hacer que se sobre escriban sobre los que estaban anteriormente

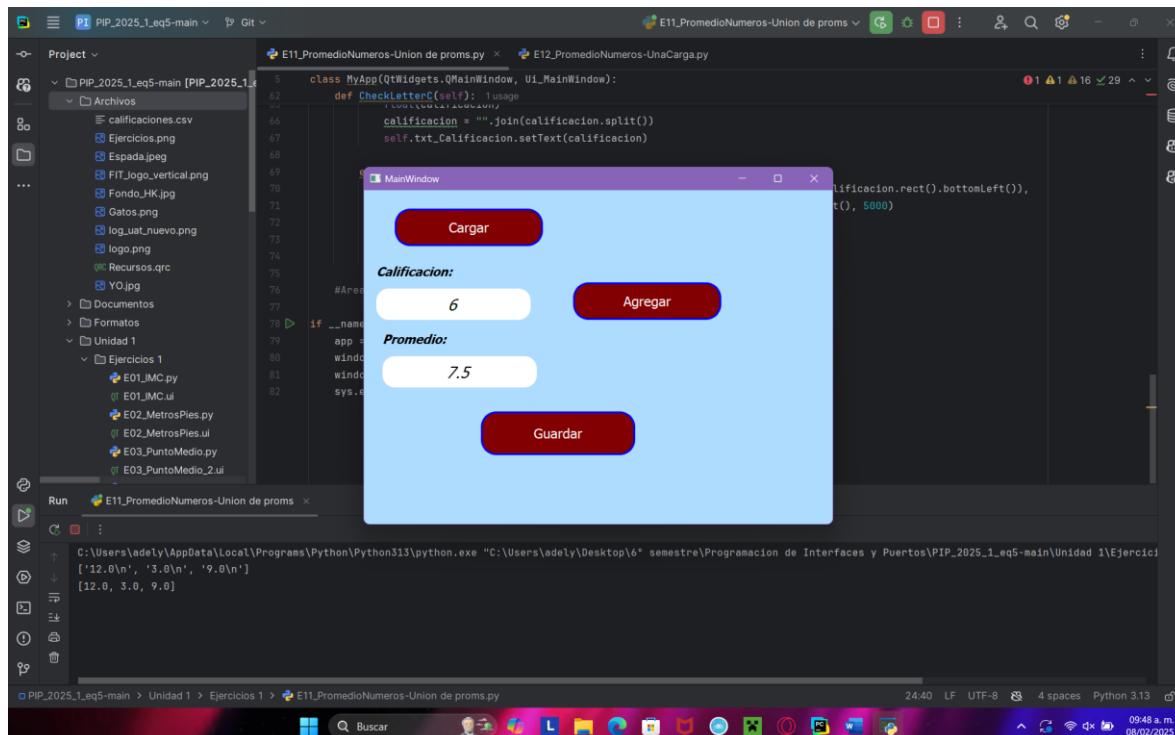
```
16     def Cargar(self): 1usage
17         try:
18             archivo = open("../Archivos/calificaciones.csv")
19         except Exception as error:
20             print(error)
21             self.msj("El archivo no existe")
22             return
23
24             contenido = archivo.readlines()
25             print(contenido)
26             datos = [float(x) for x in contenido]
27             print(datos)
28             #Ejercicio 11
29             #En lugar de Sobreescribir, Concatenar
30             self.calificaciones += datos
31             self.promedio()
32             #Tarea 12
33             #que solo se pueda cargar hasta antes de agregar la primera calificacion
34
```



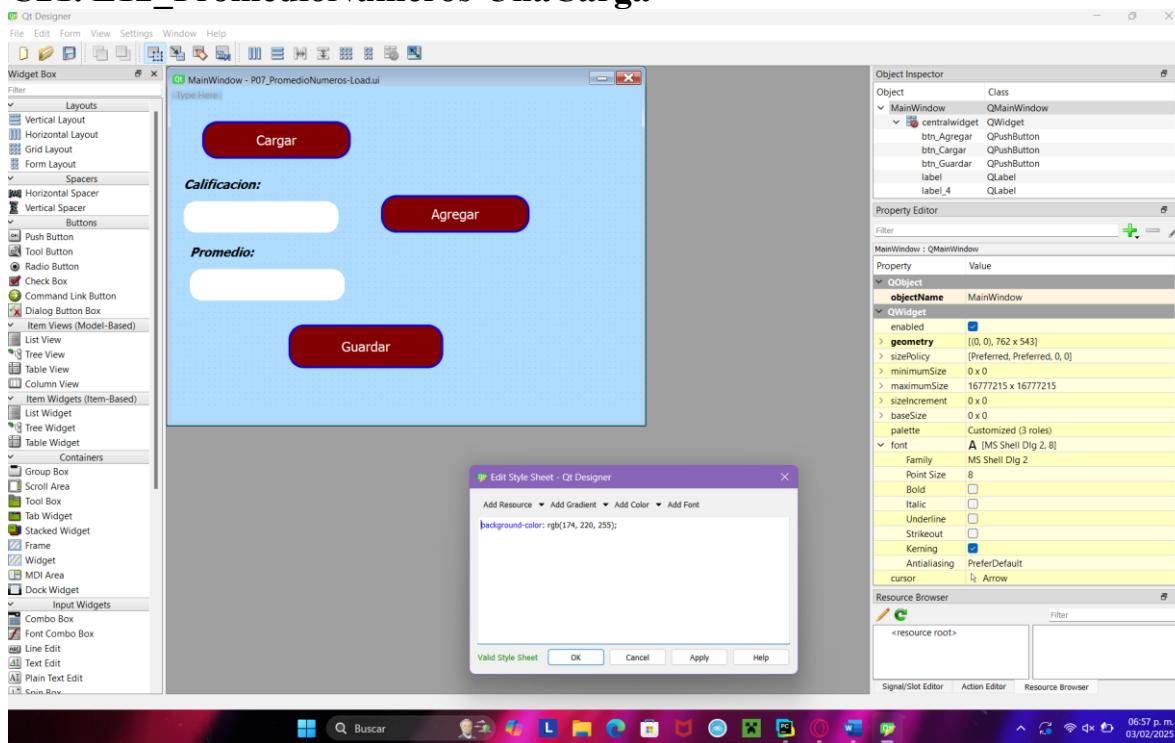
Vista principal del programa

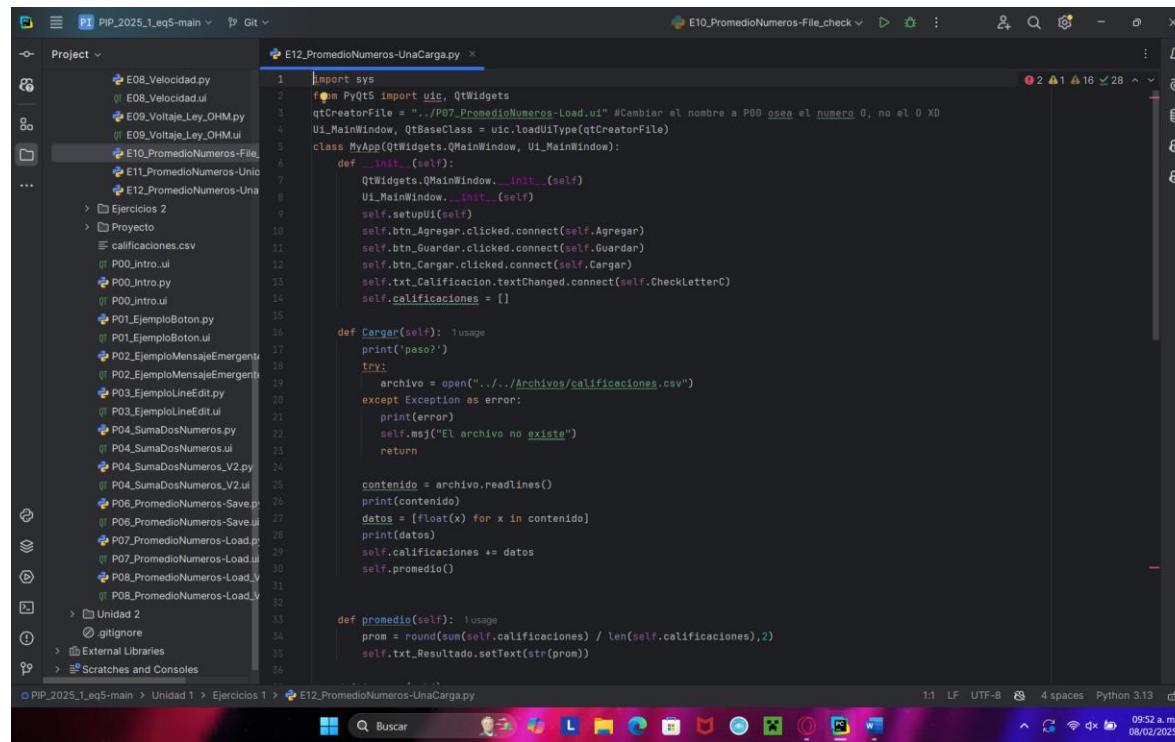


Carga de valores a self.calificaciones



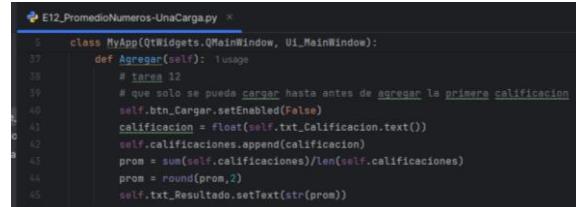
Agregar un dato y calcular el promedio teniendo en cuenta los valores cargados  
**C21. E12\_PromedioNumeros-UnaCarga**





```
Project : PIP_2025_1_eq5-main
File : E12_PromedioNumeros-UnaCarga.py
```

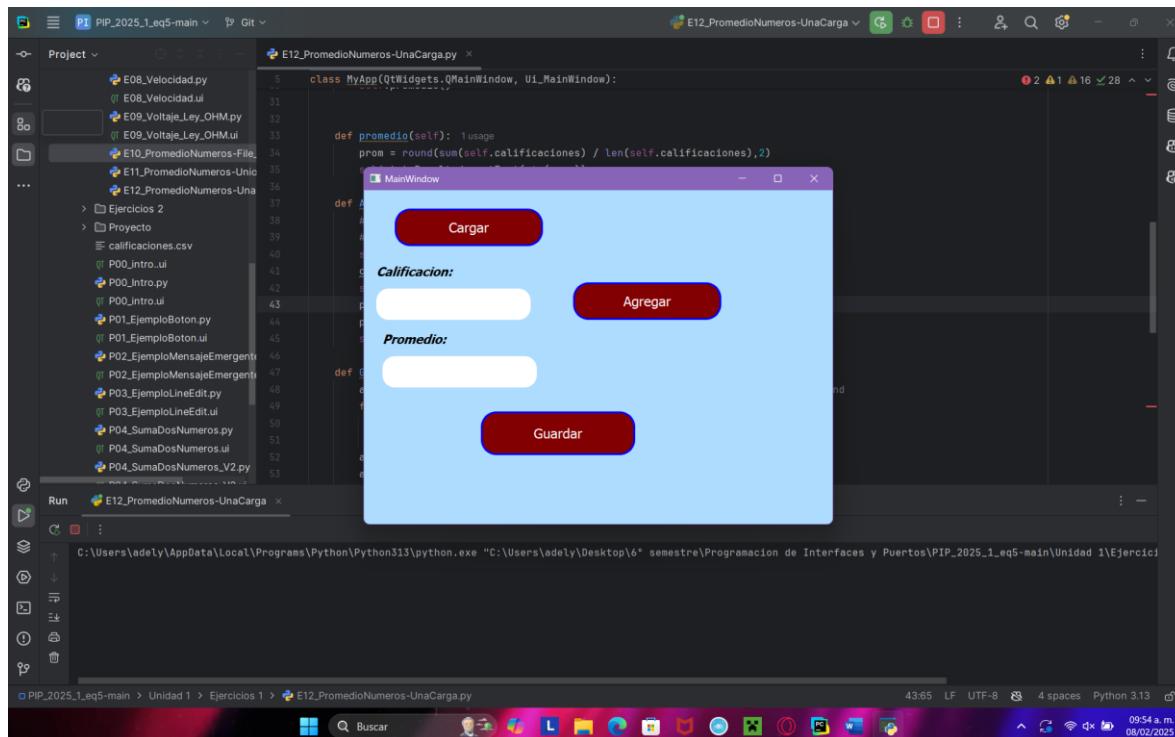
```
1 import sys
2 from PyQt5 import uic, QtWidgets
3 qtCreatorFile = ".../P07_PromedioNumeros-Load.ui" #Cambiar el nombre a P00 _cseas el numero 0, no el 0 XD
4 UI_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
5 class MyApp(QtWidgets.QMainWindow, UI_MainWindow):
6     def __init__(self):
7         QtWidgets.QMainWindow.__init__(self)
8         UI_MainWindow.__init__(self)
9         self.setupUi(self)
10        self.btn_Agregar.clicked.connect(self.Agregar)
11        self.btn_Guardar.clicked.connect(self.Guardar)
12        self.btn_Cargar.clicked.connect(self.Cargar)
13        self.txt_Calificacion.textChanged.connect(self.CheckLetterC)
14        self.calificaciones = []
15
16    def Cargar(self): usage
17        print('paso?')
18        try:
19            archivo = open("../Archivos/calificaciones.csv")
20        except Exception as error:
21            print(error)
22            self.msj("El archivo no existe")
23            return
24
25            contenido = archivo.readlines()
26            print(contenido)
27            datos = [float(x) for x in contenido]
28            print(datos)
29            self.calificaciones += datos
30            self.promedio()
31
32
33    def promedio(self): usage
34        prom = round(sum(self.calificaciones) / len(self.calificaciones),2)
35        self.txt_Resultado.setText(str(prom))
36
37
38
39
40
41
42
43
44
45
```



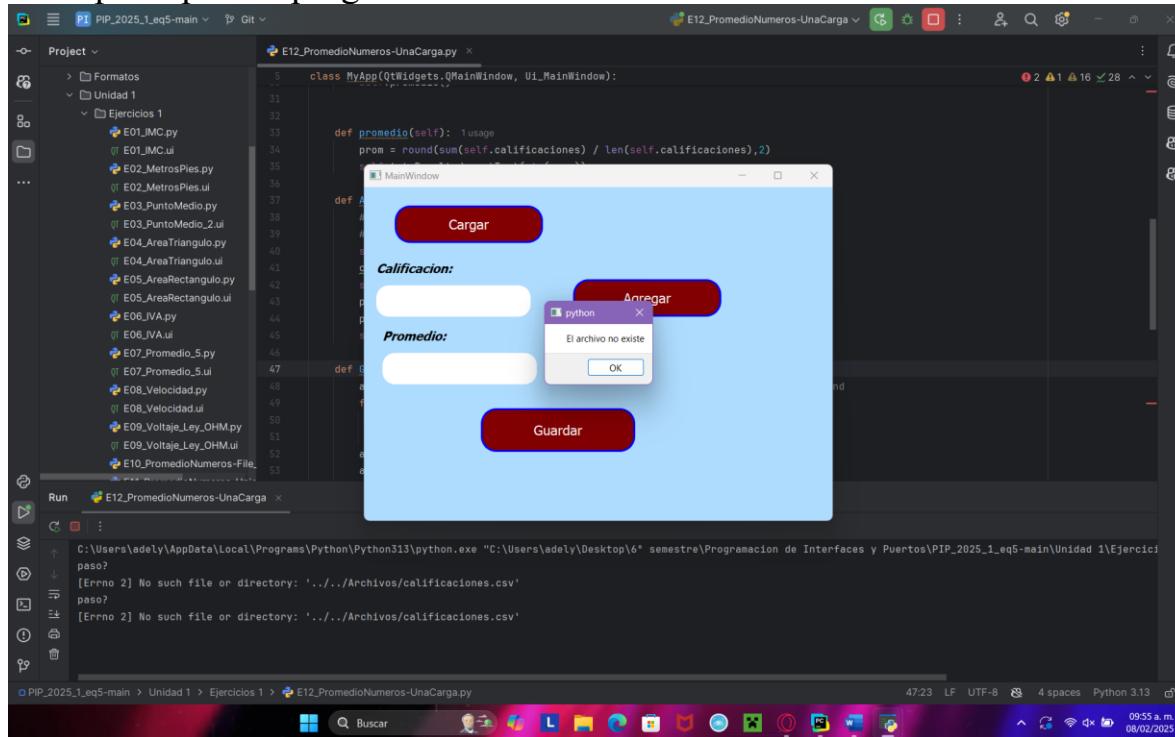
```
Project : PIP_2025_1_eq5-main
File : E12_PromedioNumeros-UnaCarga.py
```

```
1 class MyApp(QtWidgets.QMainWindow, UI_MainWindow):
2     def Agregar(self): usage
3         # tareas 12
4         # que solo se pueda cargar hasta antes de agregar la primera calificación
5         self.btn_Cargar.setEnabled(False)
6         calificacion = float(self.txt_Calificacion.text())
7         self.calificaciones.append(calificacion)
8         prom = sum(self.calificaciones)/len(self.calificaciones)
9         prom = round(prom,2)
10        self.txt_Resultado.setText(str(prom))
```

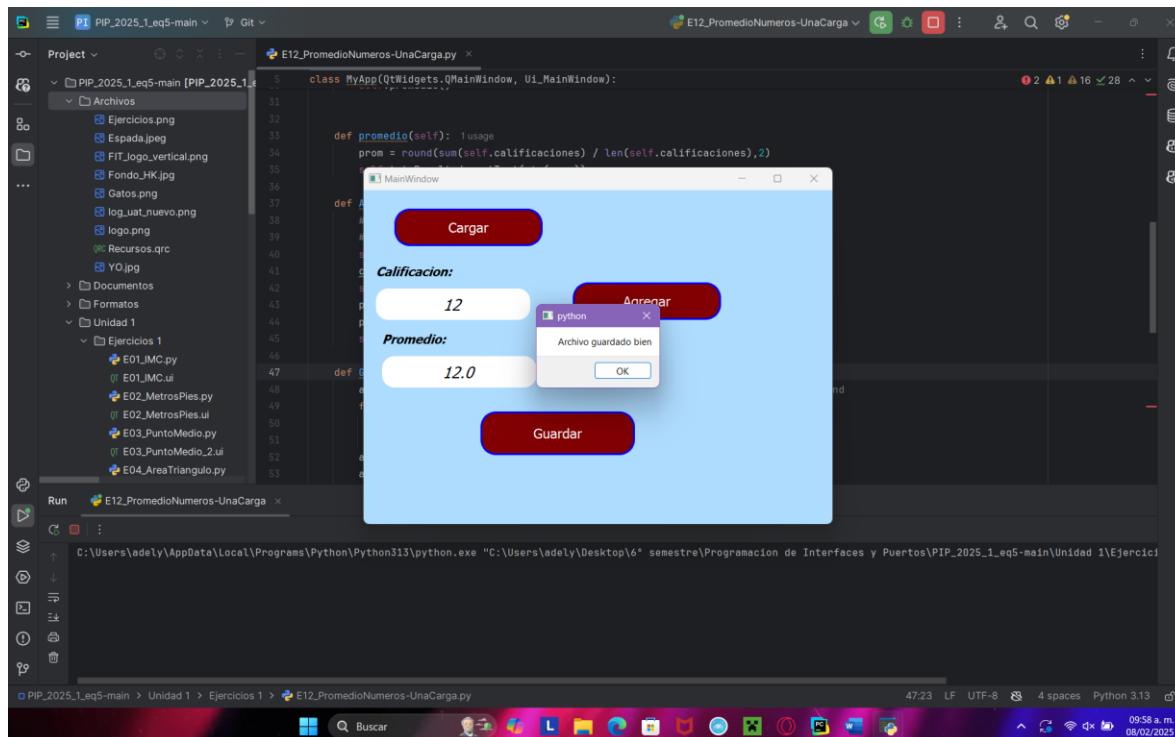
Al agregar el primer dato a la lista para sacar el promedio, se deshabilita el botón de “btn\_cargar”, usando el método de setEnable e introduciéndole False



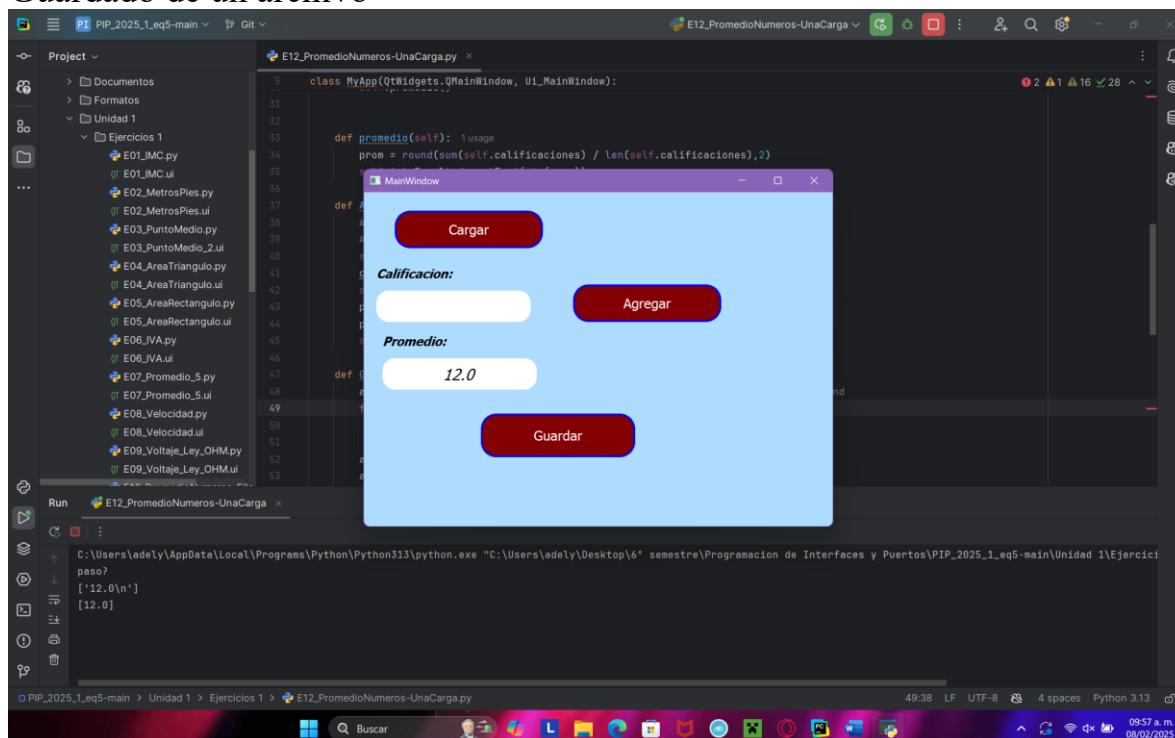
Vista principal del programa



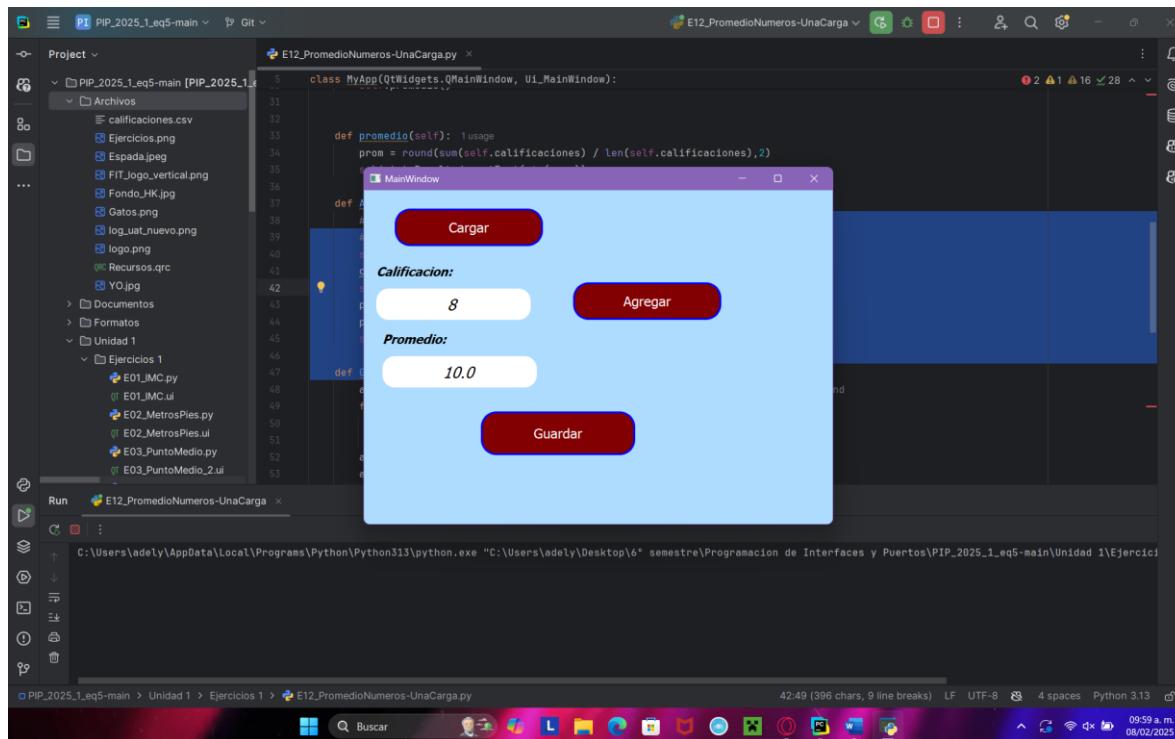
Intento de carga de un archivo cuando este no existe.



## Guardado de un archivo



## Carga de un archivo



No se ve, pero al Presionar Guardar, se deshabilita el botón de cargar, por lo que aquí esta presionado pero este ya no esta habilitado