

# **Universidad Autónoma de Tamaulipas**

## **Facultad de Ingeniería Tampico**



### **ASIGNATURA**

### **PROGRAMACION DE INTERFACES Y**

### **PUERTOS**

**6. Semestre – Grupo “I”**  
**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>
Izaguirre Cortes Emanuel	33.33%
Turrubiates Mejia Gilberto	33.33%
García Salas Yahir Misael	33.33%
Total:	100%



## INDICE

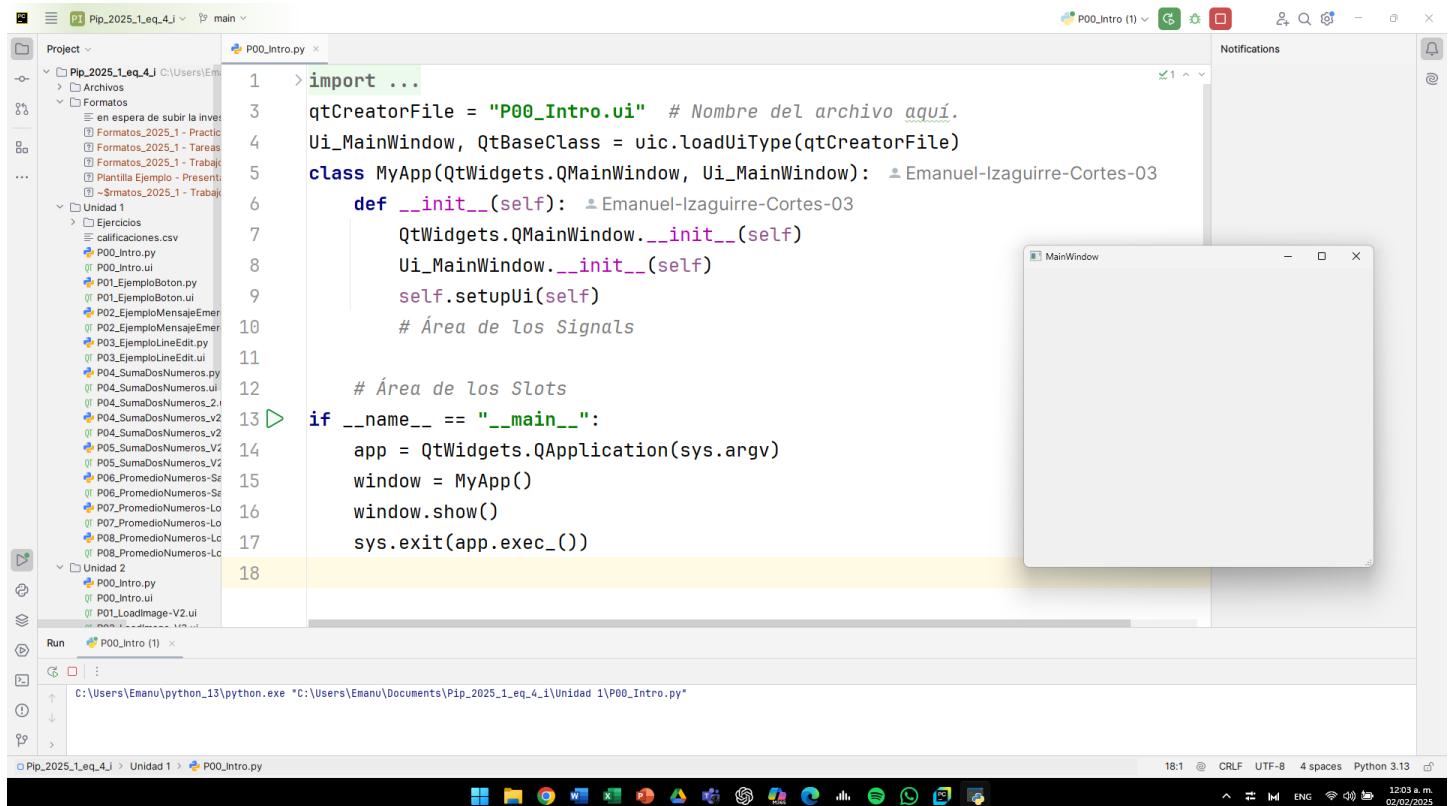
REPOSITORIO(S) .....	2
P00_Intro.....	2
P01_EJEMPLOBOTON .....	3
P02_EJEMPLOMENSAJEEMERGENTE .....	4
P03_EJEMPLOLINEEDIT .....	5
P04_SUMADOSNUMEROS.....	6
P05_SumaDosNumeros_V2.....	7
P06_PromedioNumeros-Save .....	9
P07_PromedioNumeros-Load .....	11
P08_PromedioNumeros-Load_V2 .....	14
EJERCICIOS.....	17
E01_Distancia de un punto a otro punto .....	17
E02_Comprobar si un número es par o impar.....	20
E03_Area de un cuadrado .....	22
E04_Áreas de un pentágono.....	24
E05_Comprobar si es mayor de edad una persona.....	27
E06_Imprimir en consola/MessageBox una tabla de multiplicar.....	29
E07_cantidad de caracteres de una cadena.....	32
E08_Cuántas horas le restan al día para terminarse .....	35
E09_Calcular la factorial de un número.....	37



## Repositorio(s)

Actividad	Repository
Todas	<a href="#">Emanuel-Izaguirre-Cortes-03/Pip_2025_1_eq_4_i: Trabajos en clase , ejercicios , formatos</a>

## P00\_Intro



```
Project: Pip_2025_1_eq_4_i | main | P00_Intro (1) | G | A | D | Notifications | @ | Emanu

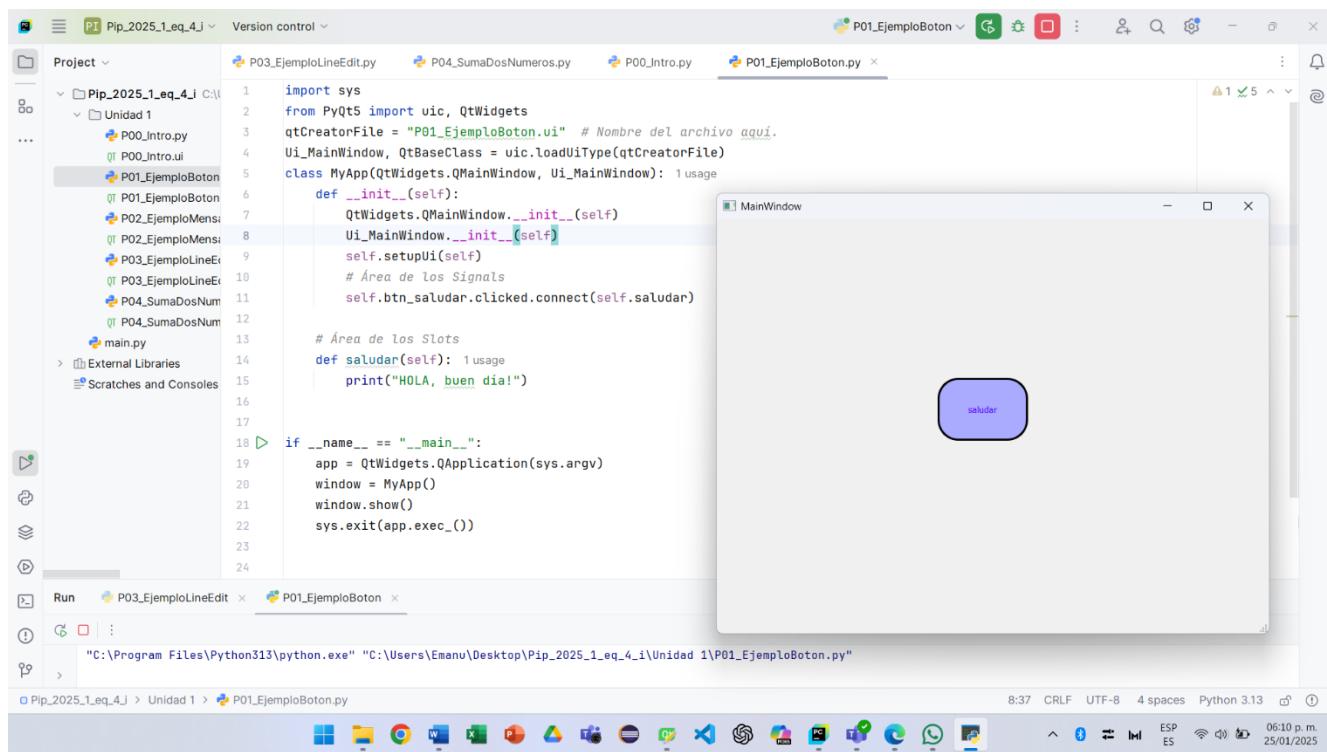
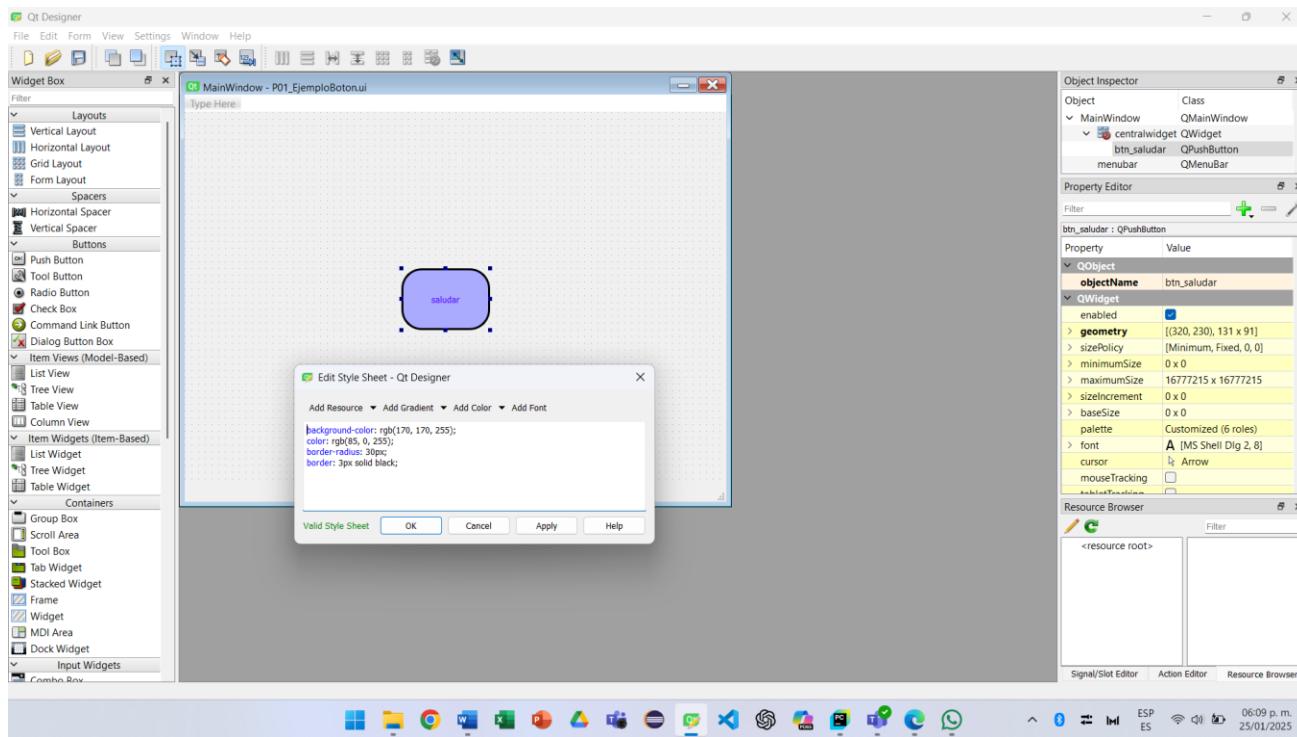
Pip_2025_1_eq_4_i C:\Users\Emanu\Pip_2025_1_eq_4_i\main\

1 > import ...
2
3 qtCreatorFile = "P00_Intro.ui" # Nombre del archivo aquí.
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        # Área de los Signals
11
12        # Área de los Slots
13    if __name__ == "__main__":
14        app = QtWidgets.QApplication(sys.argv)
15        window = MyApp()
16        window.show()
17        sys.exit(app.exec_())
18

Run: P00_Intro (1) | C:\Users\Emanu\python_13\python.exe *C:\Users\Emanu\Documents\Pip_2025_1_eq_4_i\Unidad 1\P00_Intro.py | 18:1 | CRLF | UTF-8 | 4 spaces | Python 3.13 | 12/03 a.m. | 02/02/2025
```



## P01\_EjemploBoton

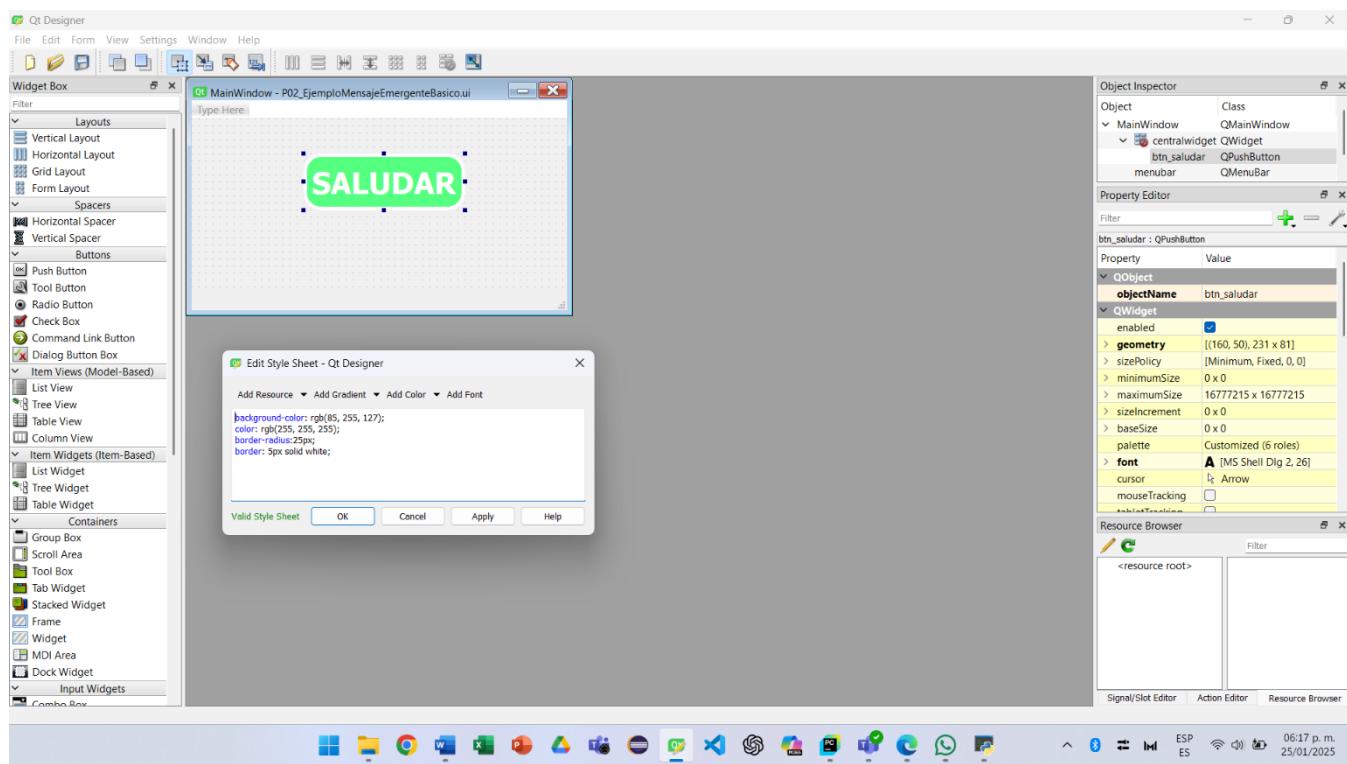




## P02\_EjemploMensajeEmergente

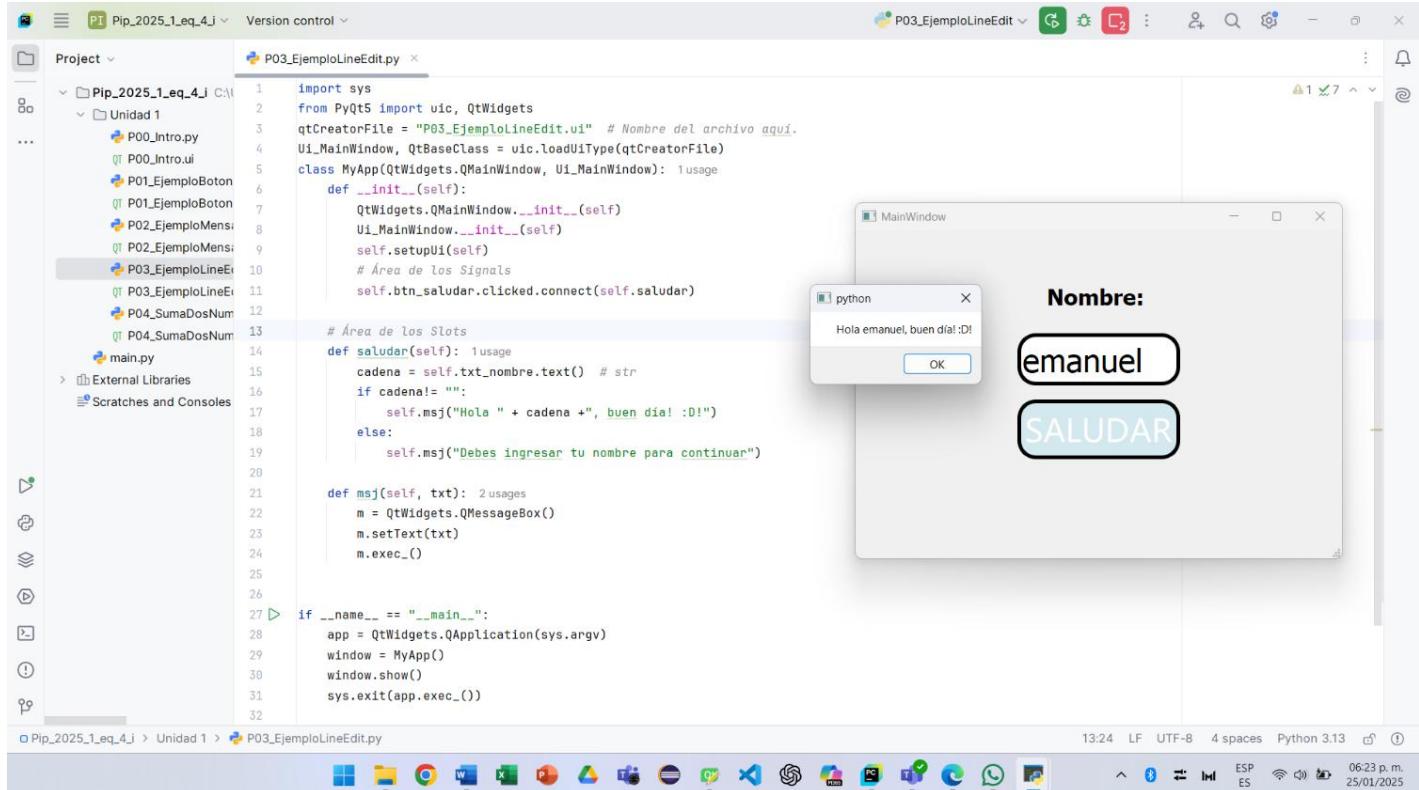
The screenshot shows the PyCharm IDE interface. On the left is the project tree for 'Pip\_2025\_1\_eq\_4\_j' containing files like 'P00\_Intro.py', 'P00\_Intro.ui', 'P01\_EjemploBoton', 'P01\_EjemploBoton', 'P02\_EjemploMensajeEmergente.py', 'P02\_EjemploMensajeEmergente.ui', 'P03\_EjemploLineEdit', 'P03\_EjemploLineEdit', 'P04\_SumaDosNumeros.py', and 'P04\_SumaDosNumeros.ui'. The main editor window displays the Python code for 'P02\_EjemploMensajeEmergente.py'. A message box window titled 'python' is shown with the text 'Hola, buen dia :D!', with an 'OK' button. The status bar at the bottom indicates the time as 16:11 and the date as 25/01/2025.

```
1 import sys
2 from PyQt5 import uic, QtWidgets
3 qtCreatorFile = "P02_EjemploBoton.ui" # Nombre del archivo aqui.
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        # Área de los Signals
11        self.btn_saludar.clicked.connect(self.saludar)
12
13        # Área de los Slots
14    def saludar(self):
15        self.msj("Hola, buen dia! :D!")
16
17    def msj(self, txt):
18        m = QtWidgets.QMessageBox()
19        m.setText(txt)
20        m.exec_()
21
22
23 if __name__ == "__main__":
24     app = QtWidgets.QApplication(sys.argv)
25     window = MyApp()
26     window.show()
27     sys.exit(app.exec_())
```





### P03\_EjemploLineEdit

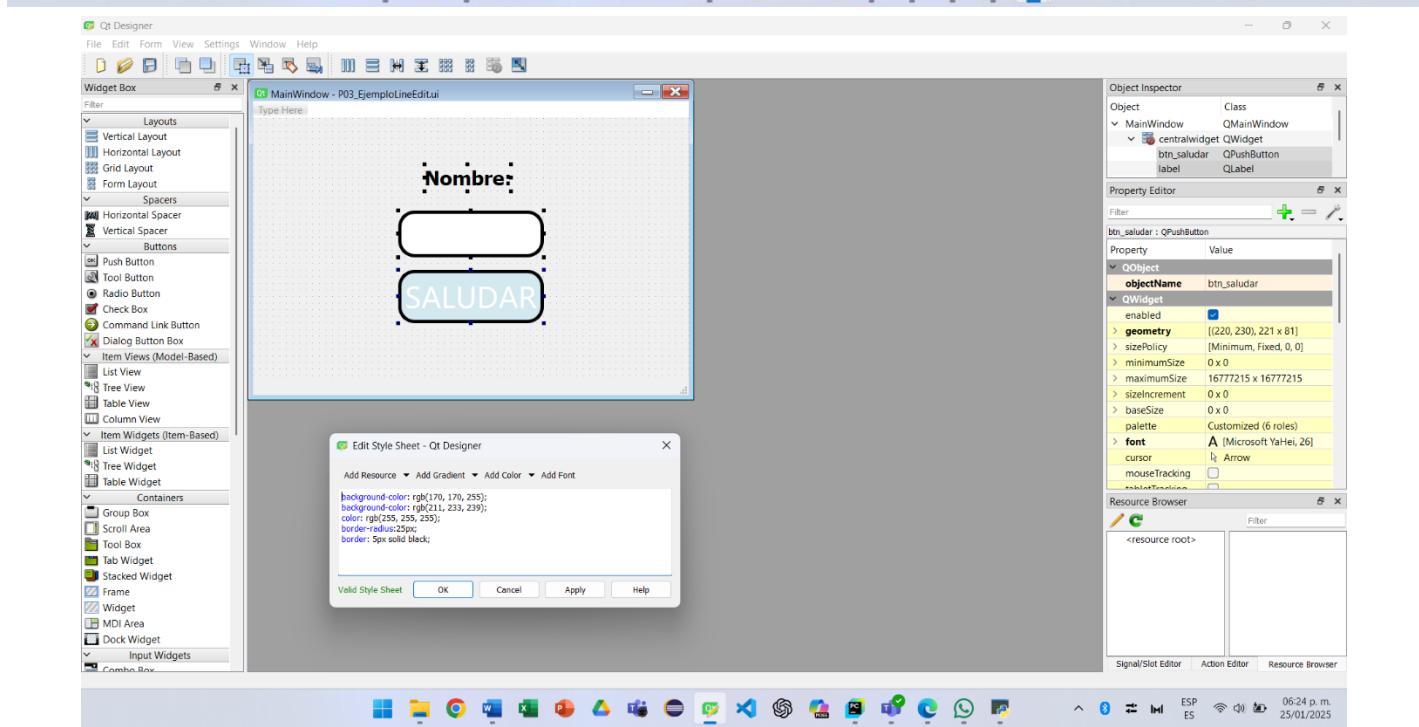


```
Project Pip_2025_1_eq_4_i Version control P03_EjemploLineEdit.py

Pip_2025_1_eq_4_i C:\
  Unidad 1
    P00_Intro.py
    P00_Intro.ui
    P01_EjemploBoton
    P01_EjemploBoton
    P02_EjemploMensi...
    P02_EjemploMensi...
    P03_EjemploLineEd...
    P03_EjemploLineEd...
    P04_SumaDosNum...
    P04_SumaDosNum...
    main.py
External Libraries
Scratches and Consoles

1 import sys
2 from PyQt5 import uic, QtWidgets
3 qtCreatorFile = "P03_EjemploLineEdit.ui" # Nombre del archivo aqui.
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        # Área de los Signals
11        self.btn_saludar.clicked.connect(self.saludar)
12
13        # Área de los Slots
14    def saludar(self): 1 usage
15        cadena = self.txt_nombre.text() # str
16        if cadena!="":
17            self.msg("Hola " + cadena + ", buen dia! :D!")
18        else:
19            self.msg("Debes ingresar tu nombre para continuar")
20
21    def msg(self, txt): 2 usages
22        m = QtWidgets.QMessageBox()
23        m.setText(txt)
24        m.exec_()
25
26
27 if __name__ == "__main__":
28     app = QtWidgets.QApplication(sys.argv)
29     window = MyApp()
30     window.show()
31     sys.exit(app.exec_())
32

13:24 LF UTF-8 4 spaces Python 3.13 06:23 p.m. 25/01/2025
```





## P04\_SumaDosNumeros

The screenshot shows the PyCharm IDE interface. On the left is the project tree for 'Pip\_2025\_1\_eq\_4\_j' containing 'Unidad 1' which includes files like P00\_Intro.py, P01\_EjemploBoton.py, P02\_EjemploMensaje.py, P03\_EjemploLineEditor.py, P04\_SumaDosNumeros.py, and main.py. The right pane displays the Python code for 'P04\_SumaDosNumeros.py'. A message box in the foreground shows 'La suma es: 11.0'. To the right is a screenshot of the application window titled 'MainWindow' with two input fields labeled 'A:' and 'B:' containing '5' and '6' respectively, and a large button labeled 'SUMAR'.

```
import sys
from PyQt5 import uic, QtWidgets
qtCreatorFile = "P04_SumaDosNumeros.ui" # Nombre del archivo aquí.
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)
        # Área de los Signals
        self.btn_sumar.clicked.connect(self.sumar)

    # Área de los Slots
    def sumar(self):
        try:
            a = float(self.txt_A.text())
            b = float(self.txt_B.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_())
```

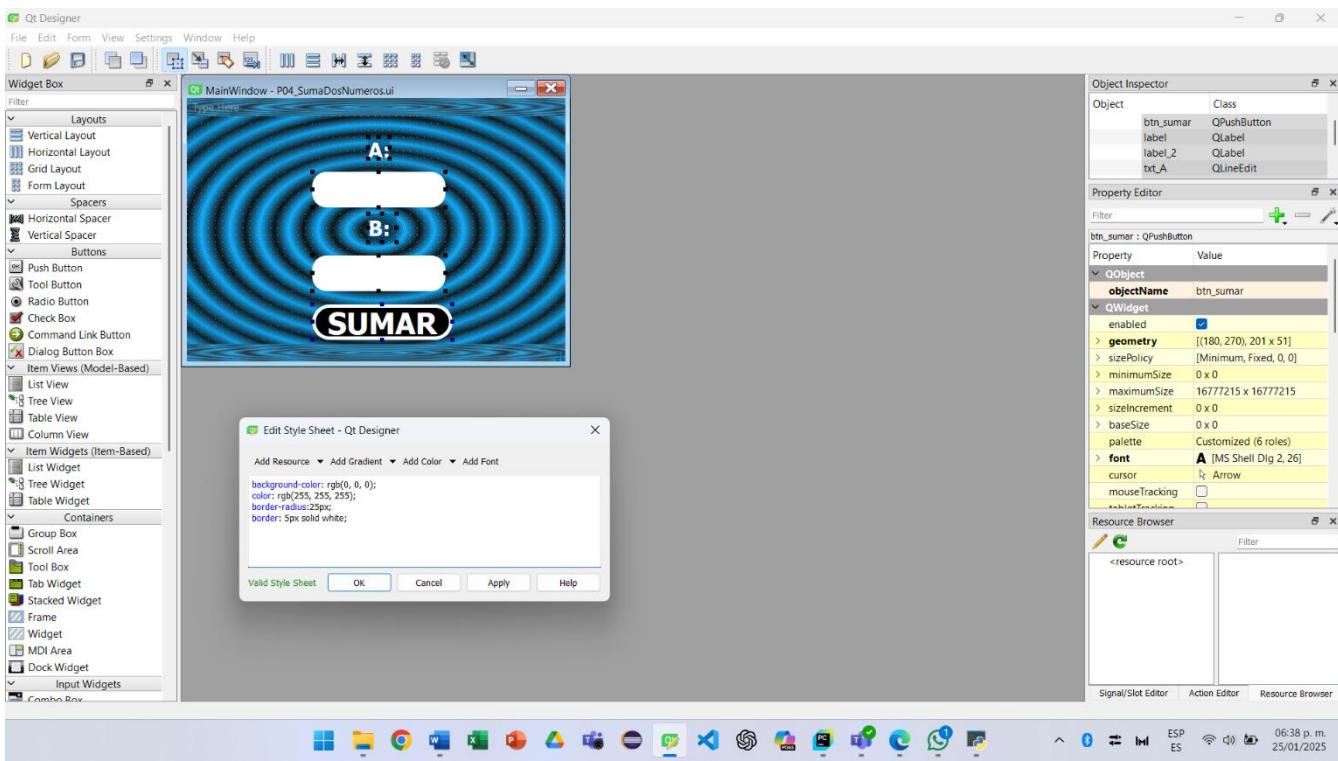
The screenshot shows the PyCharm IDE interface with the same project structure and code as the previous screenshot. The code for 'P04\_SumaDosNumeros.py' is identical, showing the addition of two numbers and displaying the result in a message box. The application window is not visible in this specific screenshot, but the code is identical to the one shown in the first screenshot.

```
import sys
from PyQt5 import uic, QtWidgets
qtCreatorFile = "P04_SumaDosNumeros.ui" # Nombre del archivo aquí.
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)
        # Área de los Signals
        self.btn_sumar.clicked.connect(self.sumar)

    # Área de los Slots
    def sumar(self):
        try:
            a = float(self.txt_A.text())
            b = float(self.txt_B.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_())
```

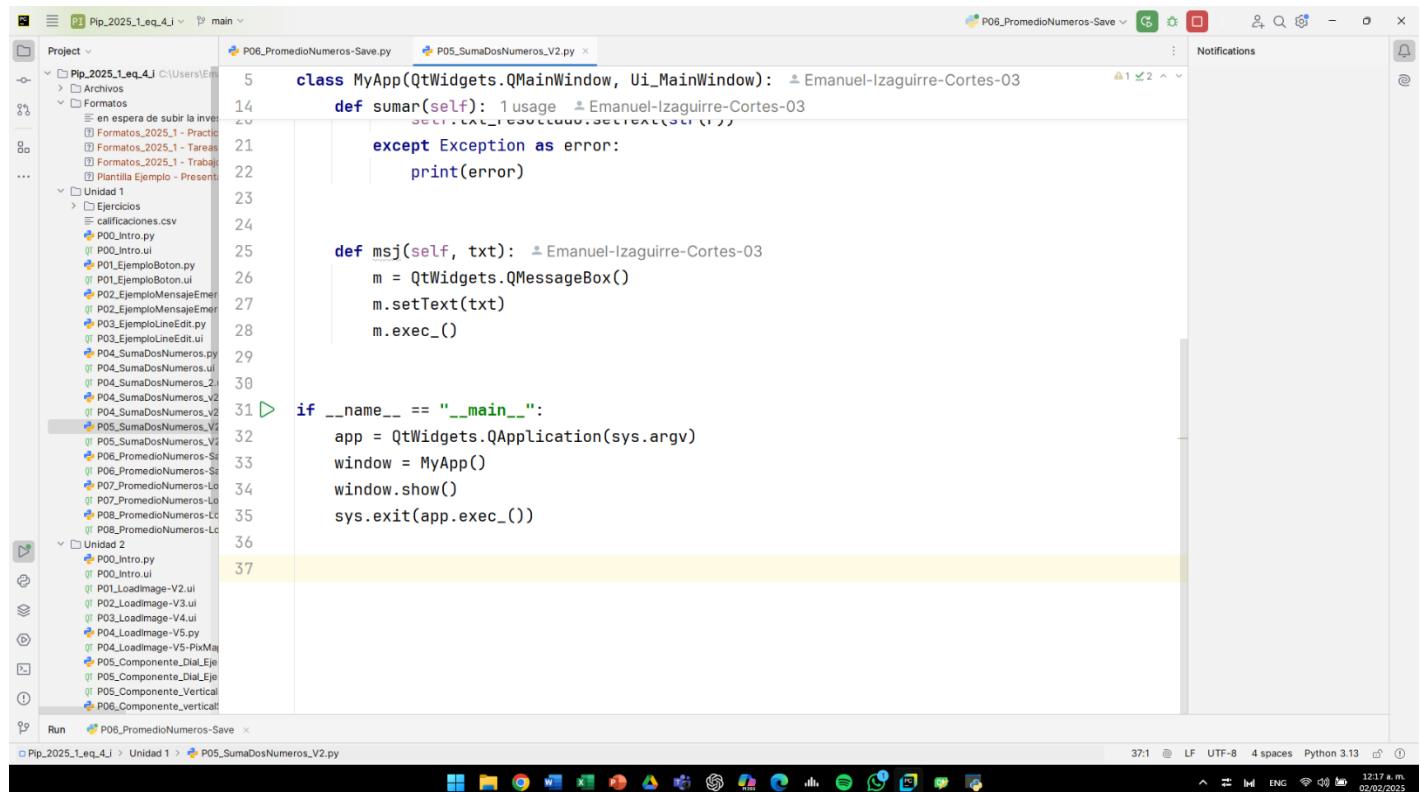


## P05\_SumaDosNumeros\_V2

The screenshot shows the PyCharm IDE interface with two code editors open. The left editor contains `P06_PromedioNumeros_Save.py`, which includes imports, UI loading, class definitions, and various methods like `sumar` and `msj`. The right editor contains `P05_SumaDosNumeros_V2.py`, which defines a UI class `MyApp` with an `__init__` method and a slot `btn_sumar.clicked.connect(self.sumar)`. Both files are associated with the `QtWidgets.QMainWindow` base class.

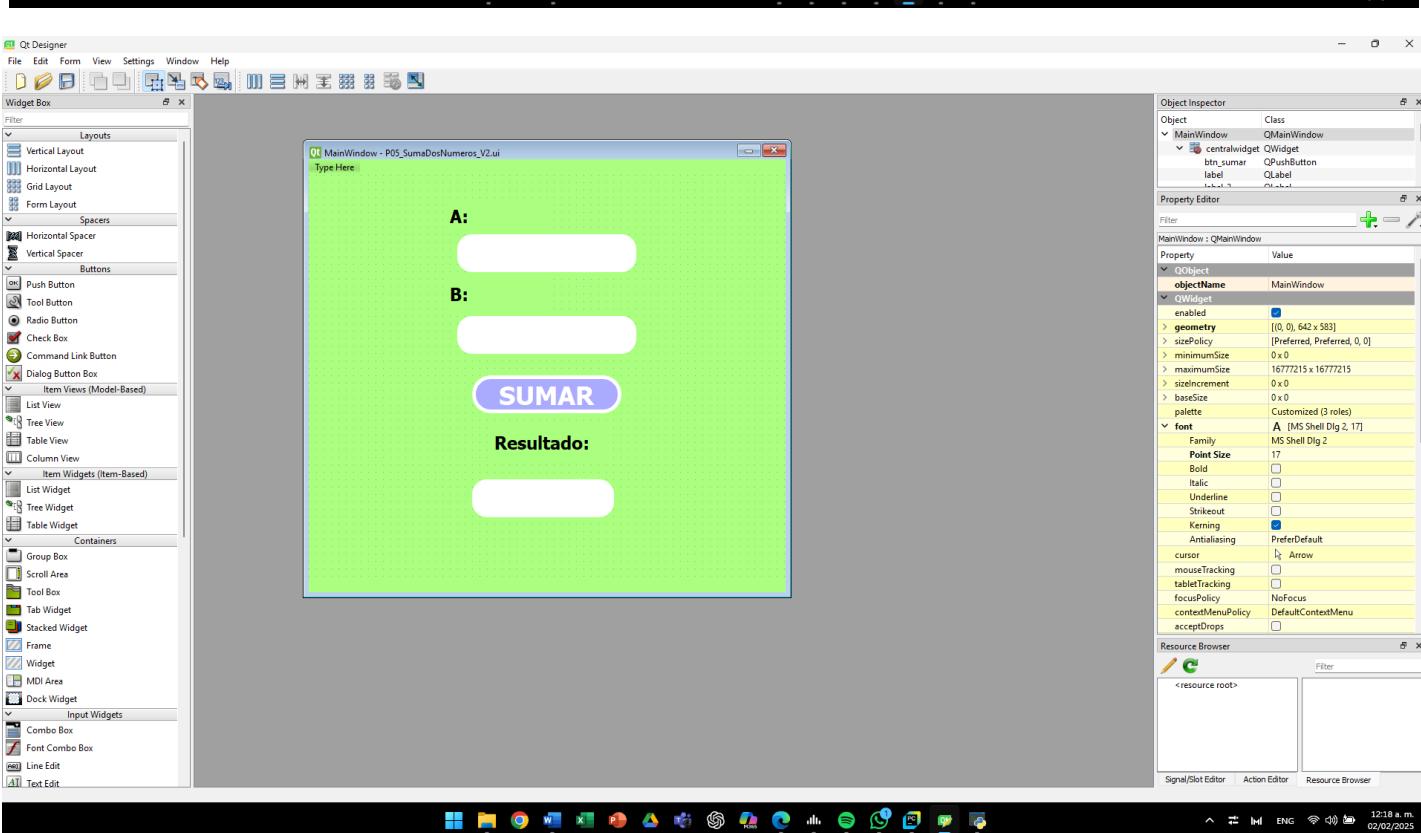
```
Project: Pip_2025_1_eq_4_J > main > P06_PromedioNumeros_Save > P05_SumaDosNumeros_V2.py

1 > import ...
2 qtCreatorFile = "P05_SumaDosNumeros_V2.ui" # Nombre del archivo aquí.
3 Uি MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
4 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):  Emanuel-Izaguirre-Cortes-03
5     def __init__(self):  Emanuel-Izaguirre-Cortes-03
6         QtWidgets.QMainWindow.__init__(self)
7         Ui_MainWindow.__init__(self)
8         self.setupUi(self)
9         # Área de los Signals
10        self.btn_sumar.clicked.connect(self.sumar)
11
12
13 # Área de los Slots
14 def sumar(self):  1 usage Emanuel-Izaguirre-Cortes-03
15     try:
16         a = float(self.txt_A.text())
17         b = float(self.txt_B.text())
18         r = a+b
19         #self.msg("La suma es: " + str(r))
20         self.txt_resultado.setText(str(r))
21     except Exception as error:
22         print(error)
23
24
25 def msj(self, txt):  Emanuel-Izaguirre-Cortes-03
26     # - Utilizado en QMessageBox()
```



The screenshot shows the PyCharm IDE interface. The code editor displays a Python script named P05\_SumaDosNumeros\_V2.py. The script contains a class MyApp that inherits from QMainWindow. It includes methods for summing numbers and displaying messages. The \_\_main\_\_ block initializes the QApplication and shows the window. The project structure on the left shows multiple files and folders related to the application development.

```
Project: Pip_2025_1_eq_4J
File: P05_SumaDosNumeros_V2.py
5      class MyApp(QtWidgets.QMainWindow, Ui_MainWindow): Emanuel-Izaguirre-Cortes-03
14          def sumar(self): Emanuel-Izaguirre-Cortes-03
20              except Exception as error: Emanuel-Izaguirre-Cortes-03
21                  print(error)
22
23
24
25          def msj(self, txt): Emanuel-Izaguirre-Cortes-03
26              m = QtWidgets.QMessageBox()
27              m.setText(txt)
28              m.exec_()
29
30
31 if __name__ == "__main__":
32     app = QtWidgets.QApplication(sys.argv)
33     window = MyApp()
34     window.show()
35     sys.exit(app.exec_())
36
37
```





The screenshot shows the PyCharm IDE interface with the file `P05_SumaDosNumeros_V2.py` open. The code implements a simple addition application. It loads a UI file named `P05_SumaDosNumeros_V2.ui`. The `sumar` slot adds the values from two text inputs (A and B) and displays the result in a third text input (Resultado). A screenshot of the application window is displayed, showing two input fields labeled 'A' and 'B' each containing '5', a button labeled 'SUMAR', and a result field labeled 'Resultado' containing '10.0'.

```
Project Pip_2025_1.eq_4 J main
P06_PromedioNumeros-Save.py P05_SumaDosNumeros_V2.py
1 > import ...
2 qtCreatorFile = "P05_SumaDosNumeros_V2.ui" # Nombre del archivo aquí.
3 Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
4 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
5     def __init__(self): Emanuel-Izaguirre-Cortes-03
6         QtWidgets.QMainWindow.__init__(self)
7         self.setupUi(self)
8         # Área de los Signals
9         self.btn_sumar.clicked.connect(self.sumar)
10
11     # Área de los Slots
12     def sumar(self): 1 usage Emanuel-Izaguirre-Cortes-03
13         try:
14             a = float(self.txt_A.text())
15             b = float(self.txt_B.text())
16             r = a+b
17             #self.msj("La suma es: " + str(r))
18             self.txt_resultado.setText(str(r))
19         except Exception as error:
20             print(error)
21
22     def msj(self, txt): Emanuel-Izaguirre-Cortes-03
23         QMessageBox.information(self, "Mensaje", txt)
24
25
26 Run P05_SumaDosNumeros_V2
Pip_2025_1.eq_4 J Unidad 1 P05_SumaDosNumeros_V2.py
13:24 ⌂ LF UTF-8 4 spaces Python 3.13 ⓘ
12:19 a.m. 02/02/2025

```

## P06\_PromedioNumeros-Save

The screenshot shows the PyCharm IDE interface with the file `P06_PromedioNumeros-Save.py` open. The code implements a grade entry application. It loads a UI file named `P06_PromedioNumeros-Save.ui`. The `agregar` slot adds a grade value to a list of grades. The `guardar` slot writes the grades to a CSV file named `calificaciones.csv`. A screenshot of the application window is partially visible.

```
Project Pip_2025_1.eq_4 J main
P06_PromedioNumeros-Save.py
1 > import ...
2 qtCreatorFile = "P06_PromedioNumeros-Save.ui" # Nombre del archivo aquí.
3 Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
4 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
5     def __init__(self): Emanuel-Izaguirre-Cortes-03
6         QtWidgets.QMainWindow.__init__(self)
7         self.setupUi(self)
8         self.calificaciones = []
9
10     # Área de los Signals
11     def agregar(self): 1 usage Emanuel-Izaguirre-Cortes-03
12         calificacion = int(self.txt_calificacion.text())
13         self.calificaciones.append(calificacion)
14         prom = sum(self.calificaciones)/len(self.calificaciones)
15         self.txt_promedio.setText(str(prom))
16
17     def guardar(self): 1 usage Emanuel-Izaguirre-Cortes-03
18         archivo = open("../Archivos/calificaciones.csv", "w") # w = write --- a = app
19         archivo.write(str(self.calificaciones))
20         archivo.close()
21
22
23 Run P06_Intro (1)
Pip_2025_1.eq_4 J Unidad 1 P06_PromedioNumeros-Save.py
33:18 ⌂ LF UTF-8 4 spaces Python 3.13 ⓘ
12:09 a.m. 02/02/2025

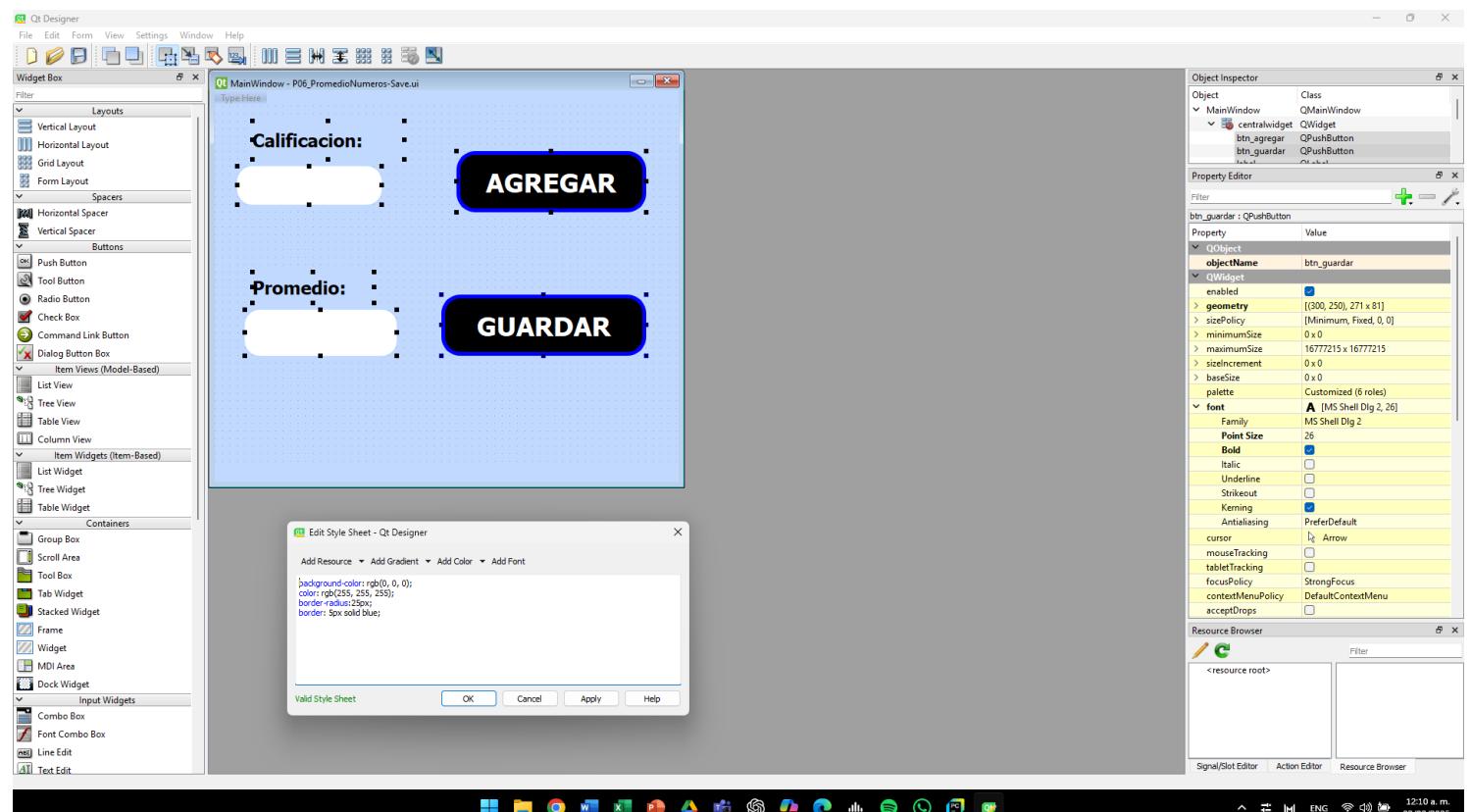
```



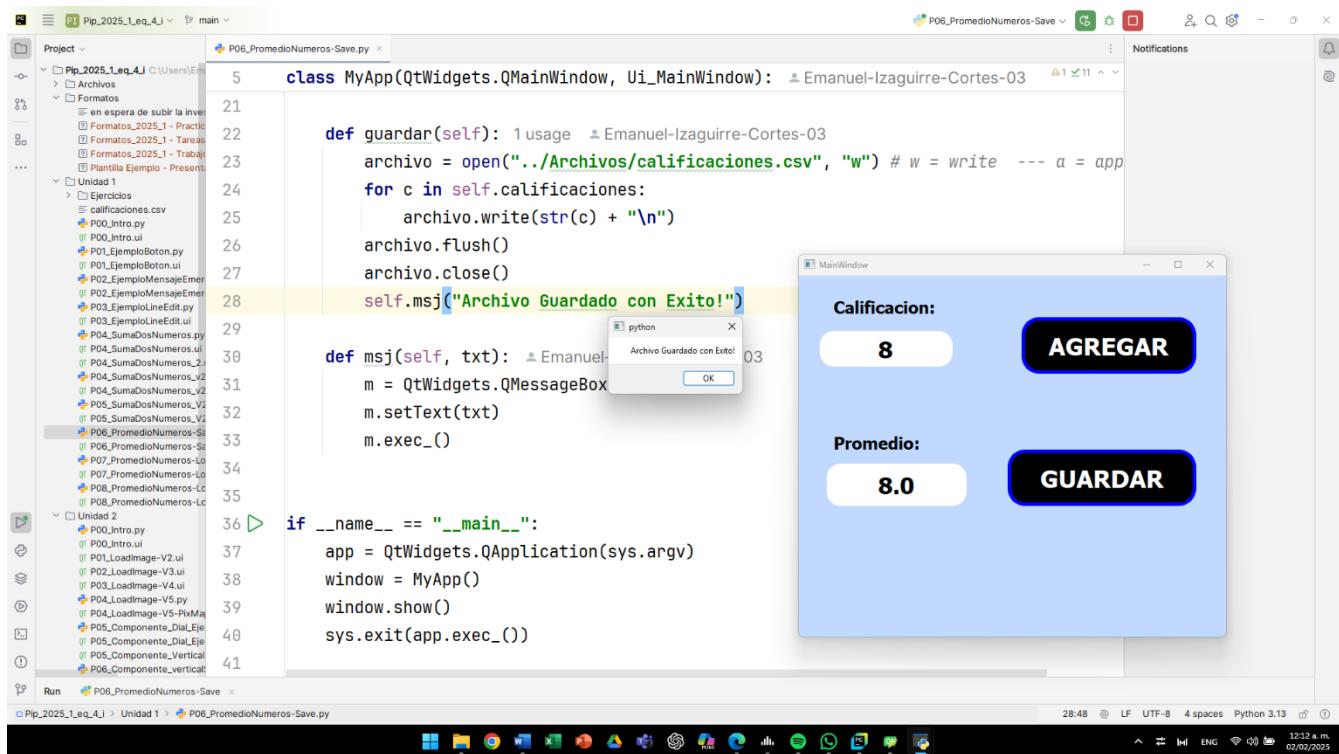
The screenshot shows the PyCharm IDE interface with the following details:

- Project:** Pip\_2025\_1\_eq\_4\_j
- File:** P06\_PromedioNumeros-Save.py
- Code Content:**

```
5 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow): Emanuel-Izaguirre-Cortes-03
6
7     def guardar(self): 1 usage Emanuel-Izaguirre-Cortes-03
8         archivo = open("../Archivos/calificaciones.csv", "w") # w = write --- a = app
9         for c in self.calificaciones:
10             archivo.write(str(c) + "\n")
11         archivo.flush()
12         archivo.close()
13         self.msj("Archivo Guardado con Exito!")
14
15     def msj(self, txt): Emanuel-Izaguirre-Cortes-03
16         m = QtWidgets.QMessageBox()
17         m.setText(txt)
18         m.exec_()
19
20
21     if __name__ == "__main__":
22         app = QtWidgets.QApplication(sys.argv)
23         window = MyApp()
24         window.show()
25         sys.exit(app.exec_())
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```
- Toolbars and Status Bar:** The status bar at the bottom shows the file path (Pip\_2025\_1\_eq\_4\_j > Unidad 1 > P06\_PromedioNumeros-Save.py), and the bottom right corner shows the time (33:18), date (12/09/2023), and system status.



10



Project: Pip\_2025\_1\_eq\_4\_J

File: P06\_PromedioNumeros-Save.py

```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):    # Emanuel-Izaguirre-Cortes-03
    def guardar(self): 1 usage  ± Emanuel-Izaguirre-Cortes-03
        archivo = open("../Archivos/calificaciones.csv", "w") # W = Write --- a = app
        for c in self.calificaciones:
            archivo.write(str(c) + "\n")
        archivo.flush()
        archivo.close()
        self.msj("Archivo Guardado con Exito!")
    def msj(self, txt): ± Emanuel-Izaguirre-Cortes-03
        m = QtWidgets.QMessageBox()
        m.setText(txt)
        m.exec_()
    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        window = MyApp()
        window.show()
        sys.exit(app.exec_())
```

Notifications

python x Archivo Guardado con Exito!

MainWindow

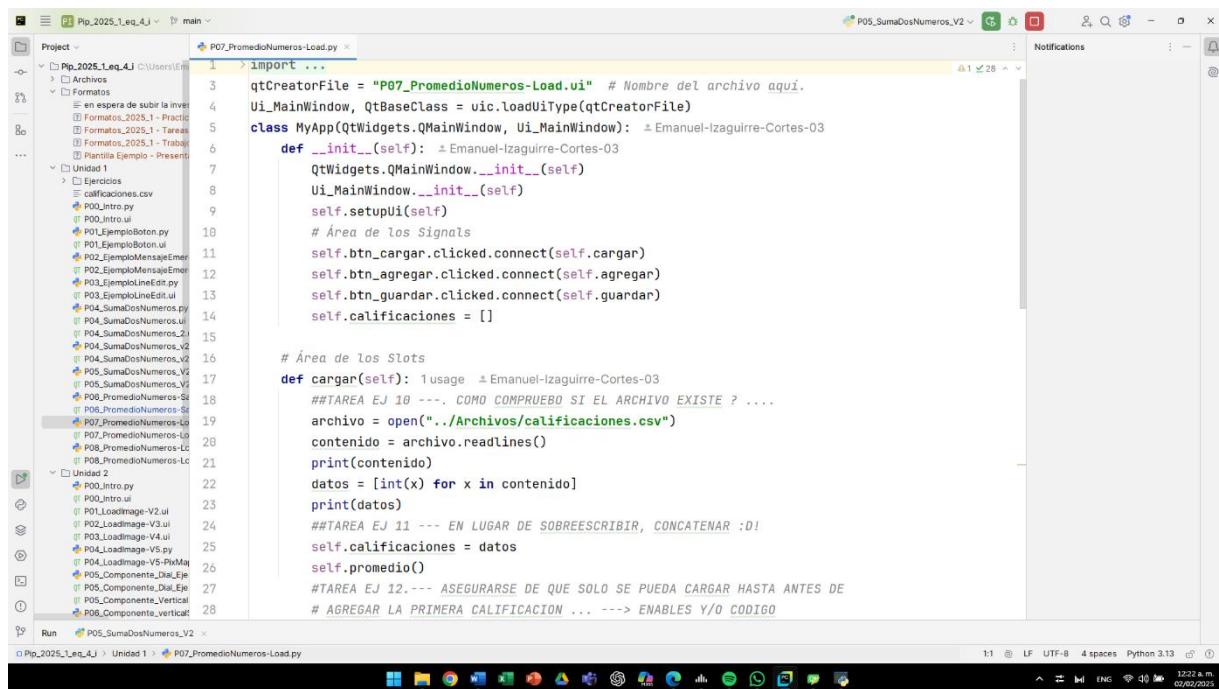
Calificación: 8 AGREGAR

Promedio: 8.0 GUARDAR

Run P06\_PromedioNumeros-Save

28:48 LF UTF-8 4 spaces Python 3.13 12:52 a.m. C2/02/2025

### P07\_PromedioNumeros-Load



Project: Pip\_2025\_1\_eq\_4\_J

File: P07\_PromedioNumeros-Load.py

```
> import ...
qtCreatorFile = "P07_PromedioNumeros-Load.ui" # Nombre del archivo aquí.
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):    # Emanuel-Izaguirre-Cortes-03
    def __init__(self):    # Emanuel-Izaguirre-Cortes-03
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        # Área de los Signals
        self.btn_cargar.clicked.connect(self.cargar)
        self.btn_agregar.clicked.connect(self.agregar)
        self.btn_guardar.clicked.connect(self.guardar)
        self.calificaciones = []
    def cargar(self): 1 usage  ± Emanuel-Izaguirre-Cortes-03
        #TAREA EJ 10 --- COMO COMPROUEBO SI EL ARCHIVO EXISTE ? ....
        archivo = open("../Archivos/calificaciones.csv")
        contenido = archivo.readlines()
        print(contenido)
        datos = [int(x) for x in contenido]
        print(datos)
        #TAREA EJ 11 --- EN LUGAR DE SOBREESCRIBIR, CONCATENAR :D!
        self.calificaciones = datos
        self.promedio()
        #TAREA EJ 12.--- ASEGURARSE DE QUE SOLO SE PUEDA CARGAR HASTA ANTES DE
        # AGREGAR LA PRIMERA CALIFICACION ... --> ENABLES Y/O CODIGO
```

Notifications

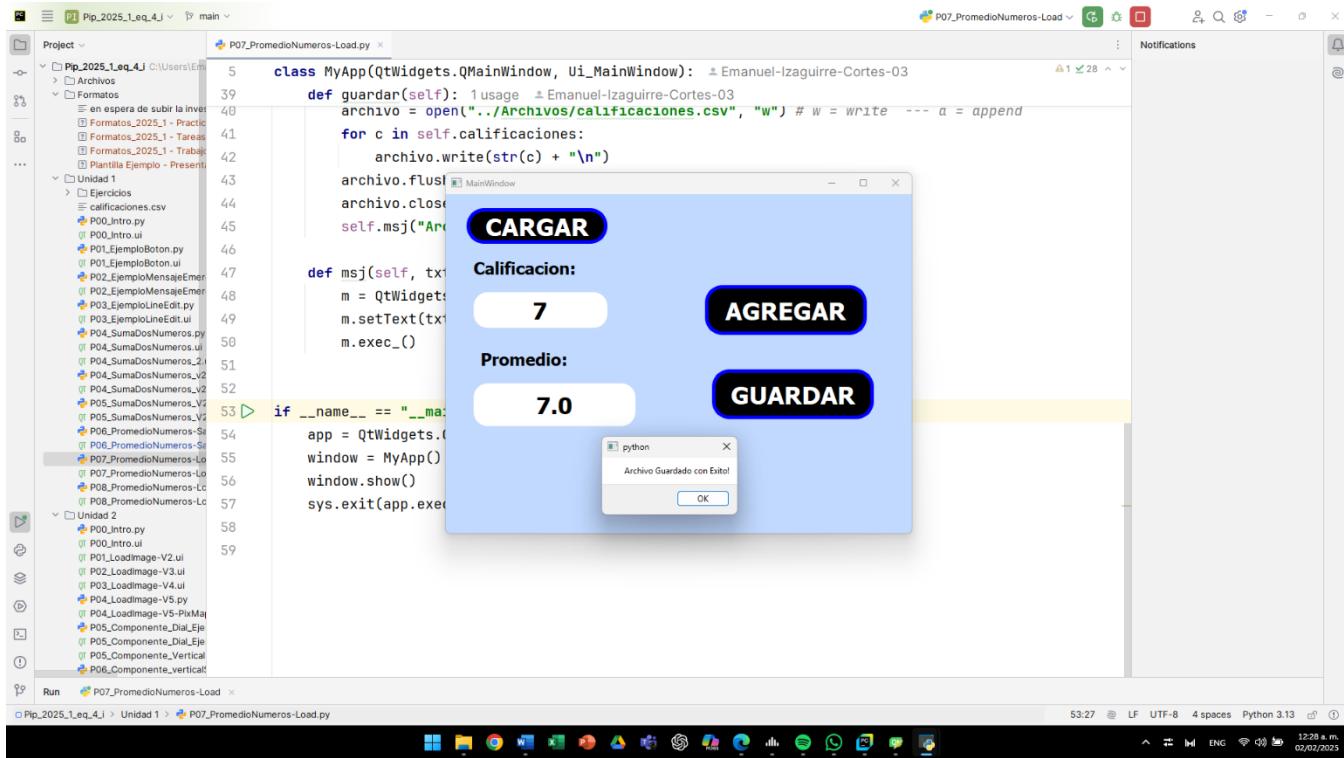
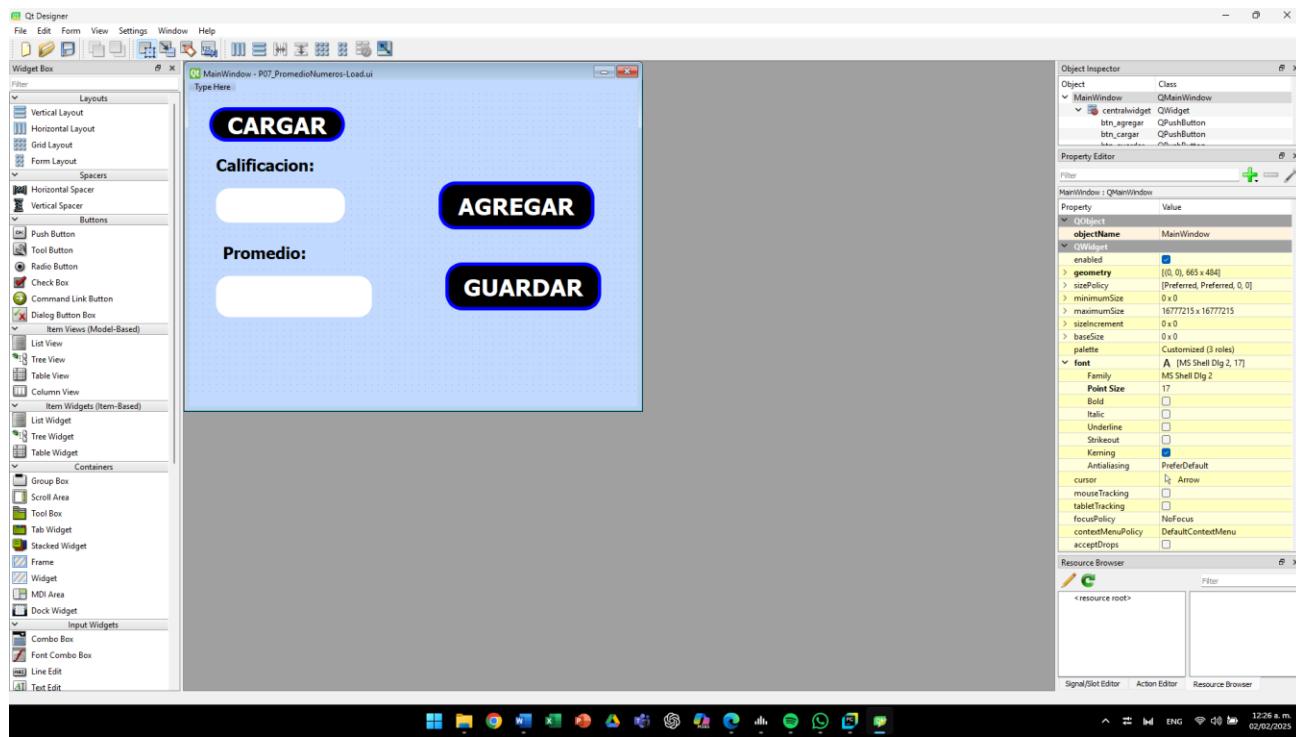
Run P07\_PromedioNumeros-Load

1:1 LF UTF-8 4 spaces Python 3.13 12:52 a.m. C2/02/2025



```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow): ...  
    def agregar(self): ...  
    def promedio(self): ...  
    def guardar(self): ...  
    def msj(self, txt): ...  
    if __name__ == "__main__":  
        app = QtWidgets.QApplication(sys.argv)  
        window = MyApp()  
        window.show()  
        sys.exit(app.exec_())
```

```
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow): ...  
    def guardar(self): ...  
    def msj(self, txt): ...  
    if __name__ == "__main__":  
        app = QtWidgets.QApplication(sys.argv)  
        window = MyApp()  
        window.show()  
        sys.exit(app.exec_())
```





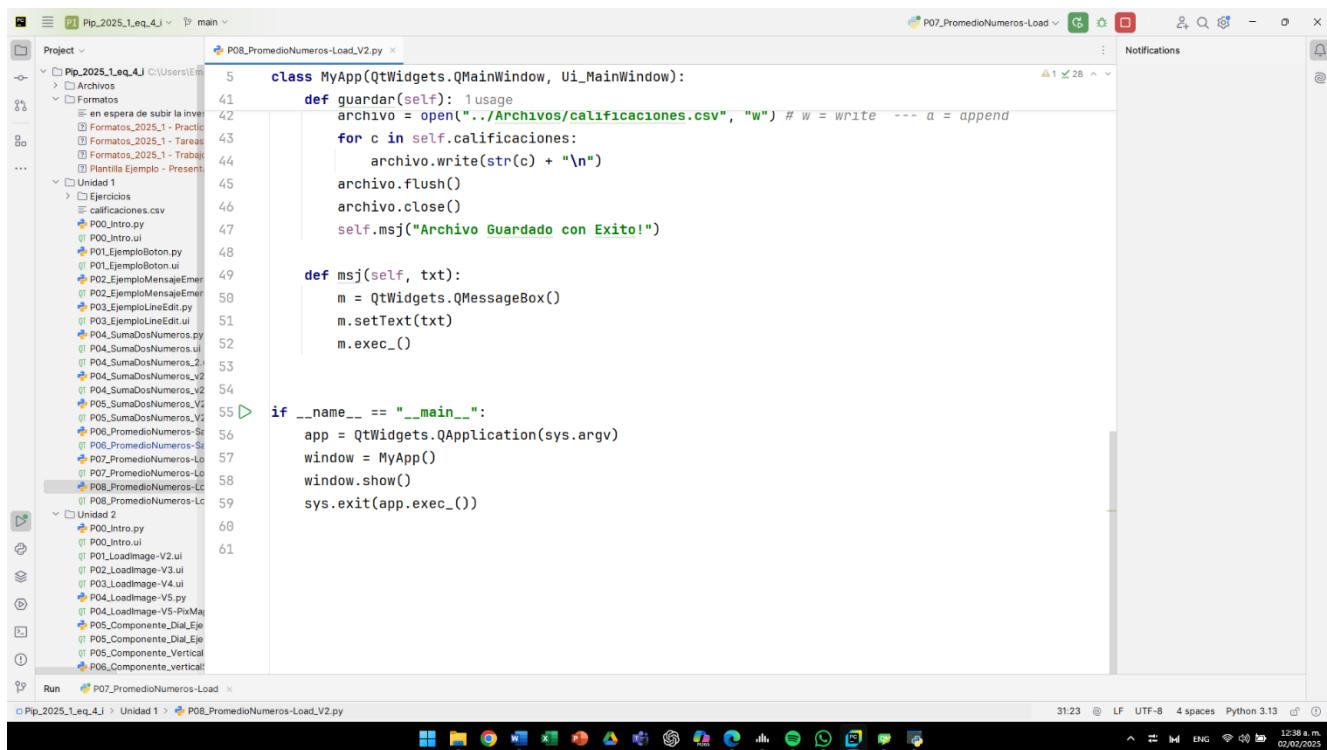
## P08\_PromedioNumeros-Load\_V2

```
Project Pip_2025_1_leq_4 J:\Users\Emerson\PycharmProjects\P08_PromedioNumeros-Load_V2.py main Notifications

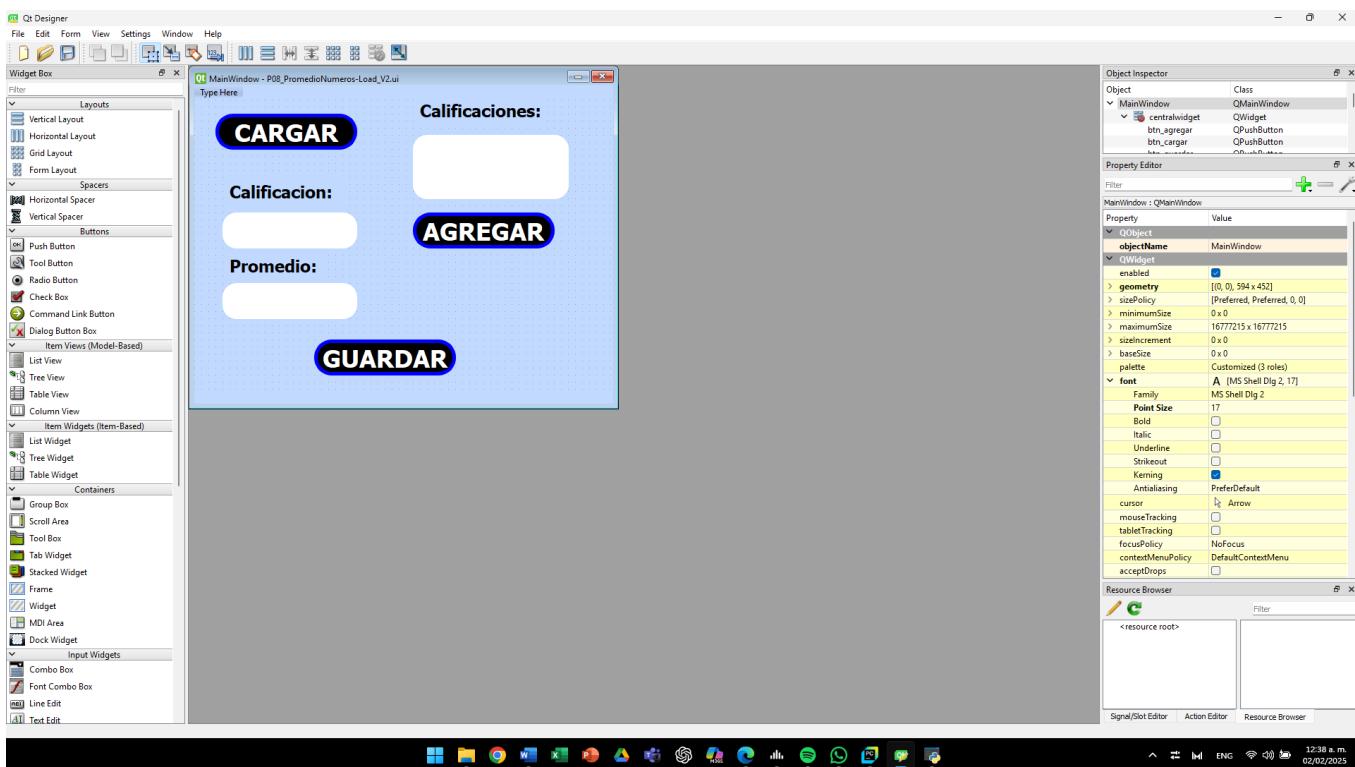
1 > import ...
2
3 qtCreatorFile = "P08_PromedioNumeros-Load_V2.ui" # Nombre del archivo aquí.
4
5 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
6     def __init__(self):
7         QtWidgets.QMainWindow.__init__(self)
8         Ui_MainWindow.__init__(self)
9         self.setupUi(self)
10        # Área de los Signals
11        self.btn_cargar.clicked.connect(self.cargar)
12        self.btn_agregar.clicked.connect(self.agregar)
13        self.btn_guardar.clicked.connect(self.guardar)
14        self.calificaciones = []
15
16        # Área de los Slots
17    def cargar(self): 1 usage
18        ##TAREA EJ 10 ---. COMO COMPROUEBO SI EL ARCHIVO EXISTE ? ....
19        archivo = open("../Archivos/calificaciones.csv")
20        contenido = archivo.readlines()
21        print(contenido)
22        datos = [int(x) for x in contenido]
23        print(datos)
24        ##TAREA EJ 11 --- EN LUGAR DE SOBREESCRIBIR, CONCATENAR :D
25        self.calificaciones = datos
26        self.promedio()
27        ##TAREA EJ 12.--- ASEGURARSE DE QUE SOLO SE PUEDA CARGAR HASTA ANTES DE
28        # AGREGAR LA PRIMERA CALIFICACION ... --> ENABLES Y/O CODIGO
29
30
Run P07_PromedioNumeros-Load
31:23 @ LF UTF-8 4 spaces Python 3.13 ⓘ
Pip_2025_1_leq_4 J:\Users\Emerson\PycharmProjects\P08_PromedioNumeros-Load_V2.py 12:37 a. m. 02/02/2025
```

```
Project Pip_2025_1_leq_4 J:\Users\Emerson\PycharmProjects\P08_PromedioNumeros-Load_V2.py main Notifications

5 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
6     def cargar(self): 1 usage
7         # AGREGAR LA PRIMERA CALIFICACION ... --> ENABLES Y/O CODIGO
8         self.txt_lista_calificaciones.setText(str(self.calificaciones))
9
10    def agregar(self): 1 usage
11        calificacion = int(self.txt_calificacion.text())
12        self.calificaciones.append(calificacion)
13        self.promedio()
14        self.txt_lista_calificaciones.setText(str(self.calificaciones))
15
16    def promedio(self): 2 usages
17        prom = sum(self.calificaciones) / len(self.calificaciones)
18        self.txt_promedio.setText(str(prom))
19
20    def guardar(self): 1 usage
21        archivo = open("../Archivos/calificaciones.csv", "w") # w = write --- a = append
22        for c in self.calificaciones:
23            archivo.write(str(c) + "\n")
24        archivo.flush()
25        archivo.close()
26        self.msj("Archivo Guardado con Exito!")
27
28    def msj(self, txt):
29        m = QtWidgets.QMessageBox()
30        m.setText(txt)
31        m.exec_()
32
Run P07_PromedioNumeros-Load
31:23 @ LF UTF-8 4 spaces Python 3.13 ⓘ
Pip_2025_1_leq_4 J:\Users\Emerson\PycharmProjects\P08_PromedioNumeros-Load_V2.py 12:38 a. m. 02/02/2025
```



```
Project Pip_2025_1_eq_4_J main
P08_PromedioNumeros_Load_V2.py
5 class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
41     def guardar(self): 1 usage
42         archivo = open("../Archivos/calificaciones.csv", "w") # w = write --- a = append
43
44         for c in self.calificaciones:
45             archivo.write(str(c) + "\n")
46
47         archivo.flush()
48         archivo.close()
49         self.msj("Archivo Guardado con Exito!")
50
51     def msj(self, txt):
52         m = QtWidgets.QMessageBox()
53         m.setText(txt)
54         m.exec_()
55
56 if __name__ == "__main__":
57     app = QtWidgets.QApplication(sys.argv)
58     window = MyApp()
59     window.show()
60     sys.exit(app.exec_())
61
```





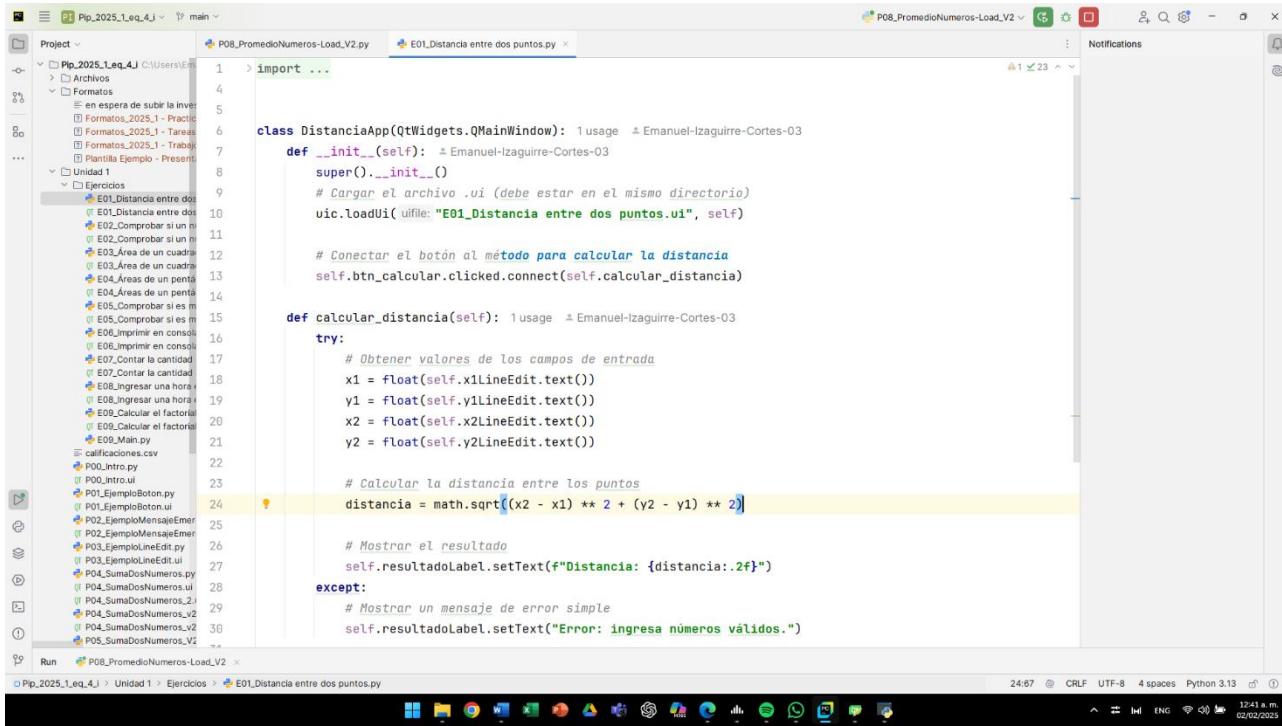
The screenshot shows the PyCharm IDE interface. On the left is the project tree for 'Pip\_2025\_1\_eq\_4' containing files like P08\_PromedioNumeros-Load\_V2.py, P08\_PromedioNumeros-L2, and various UI files. The main editor shows the code for P08\_PromedioNumeros-Load\_V2.py:

```
Project: Pip_2025_1_eq_4
File: P08_PromedioNumeros-Load_V2.py
5   class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
41      def guardar(self): 1 usage
42          archivo = open("../Archivos/calificaciones.csv", "w") # w = write --- a = append
43          for c in self.calificaciones:
44              archivo.write(str(c) + "\n")
45          archivo.flush()
46          archivo.close()
47          self.msj("Archivo guardado exitosamente")
48
49      def msj(self, txt):
50          m = QtWidgets.QMessageBox()
51          m.setText(txt)
52          m.exec_()
53
54  if __name__ == "__main__":
55      app = QtWidgets.QApplication(sys.argv)
56      window = MyApp()
57      window.show()
58      sys.exit(app.exec_())
59
60
61
```

A modal dialog box titled 'Calificaciones:' is displayed in the center. It has a 'CARGAR' button at the top. Below it is a text input field containing '8'. To the right of the input field is a 'AGREGAR' button. Below the input field is the text 'Promedio:'. Underneath the input field is a 'GUARDAR' button.

## Ejercicios

### E01\_Distancia de un punto a otro punto



```

import ...

class DistanciaApp(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        # Cargar el archivo .ui (debe estar en el mismo directorio)
        uic.loadUi("E01_Distancia entre dos puntos.ui", self)

        # Conectar el botón al método para calcular la distancia
        self.btn_calcular.clicked.connect(self.calcular_distancia)

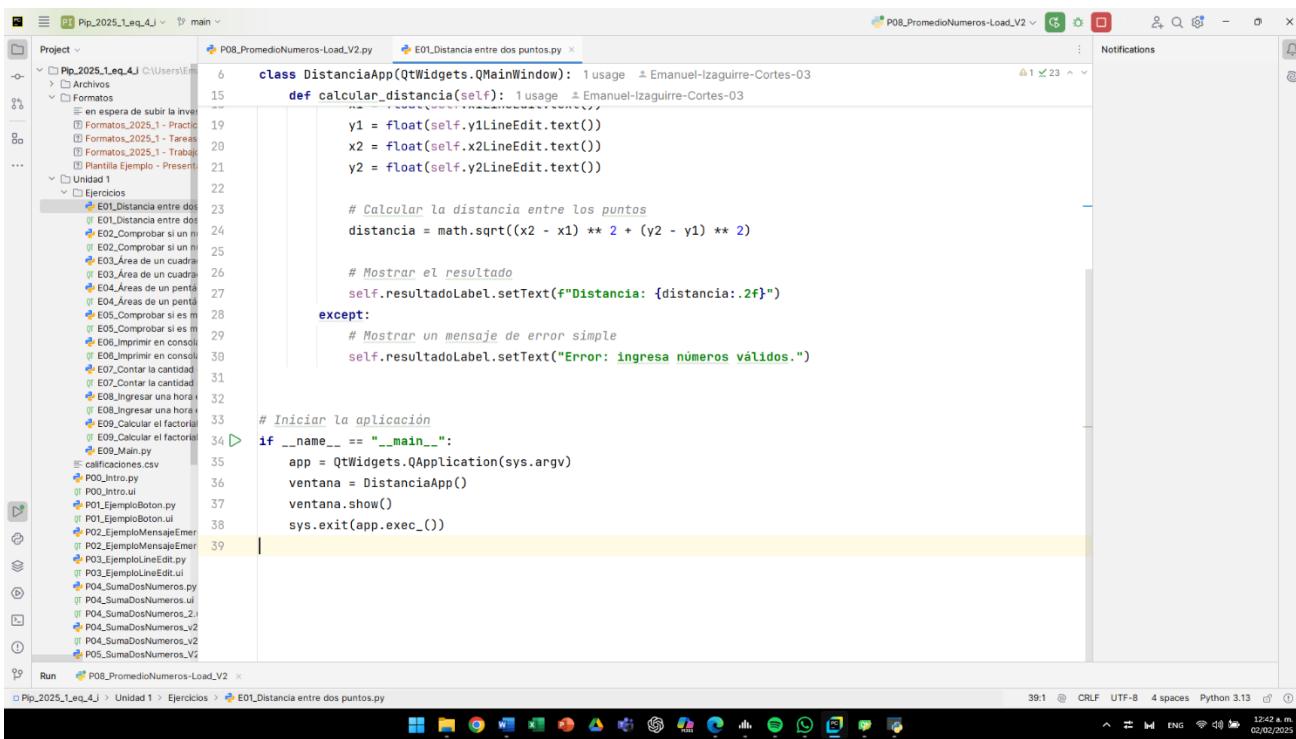
    def calcular_distancia(self):
        try:
            # Obtener valores de los campos de entrada
            x1 = float(self.xLineEdit.text())
            y1 = float(self.yLineEdit.text())
            x2 = float(self.x2LineEdit.text())
            y2 = float(self.y2LineEdit.text())

            # Calcular la distancia entre los puntos
            distancia = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

            # Mostrar el resultado
            self.resultadoLabel.setText(f"Distancia: {distancia:.2f}")
        except:
            # Mostrar un mensaje de error simple
            self.resultadoLabel.setText("Error: ingresa números válidos.")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    ventana = DistanciaApp()
    ventana.show()
    sys.exit(app.exec_())

```



```

import ...

class DistanciaApp(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        # Cargar el archivo .ui (debe estar en el mismo directorio)
        uic.loadUi("E01_Distancia entre dos puntos.ui", self)

        # Conectar el botón al método para calcular la distancia
        self.btn_calcular.clicked.connect(self.calcular_distancia)

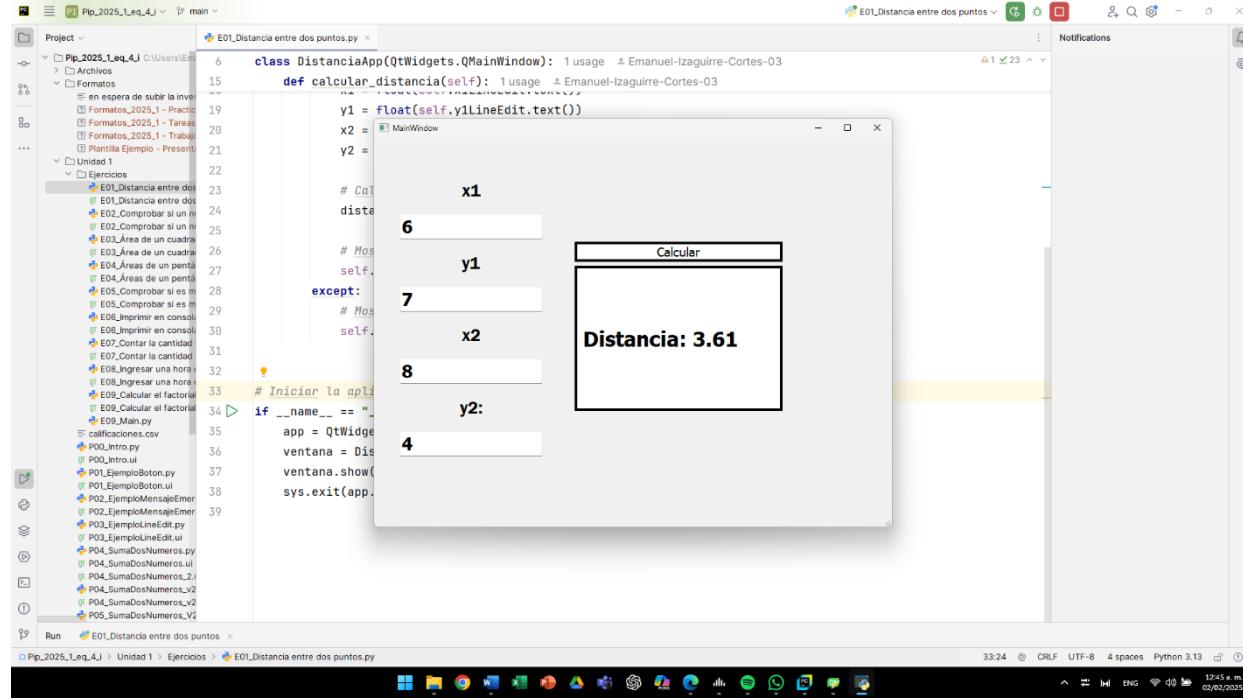
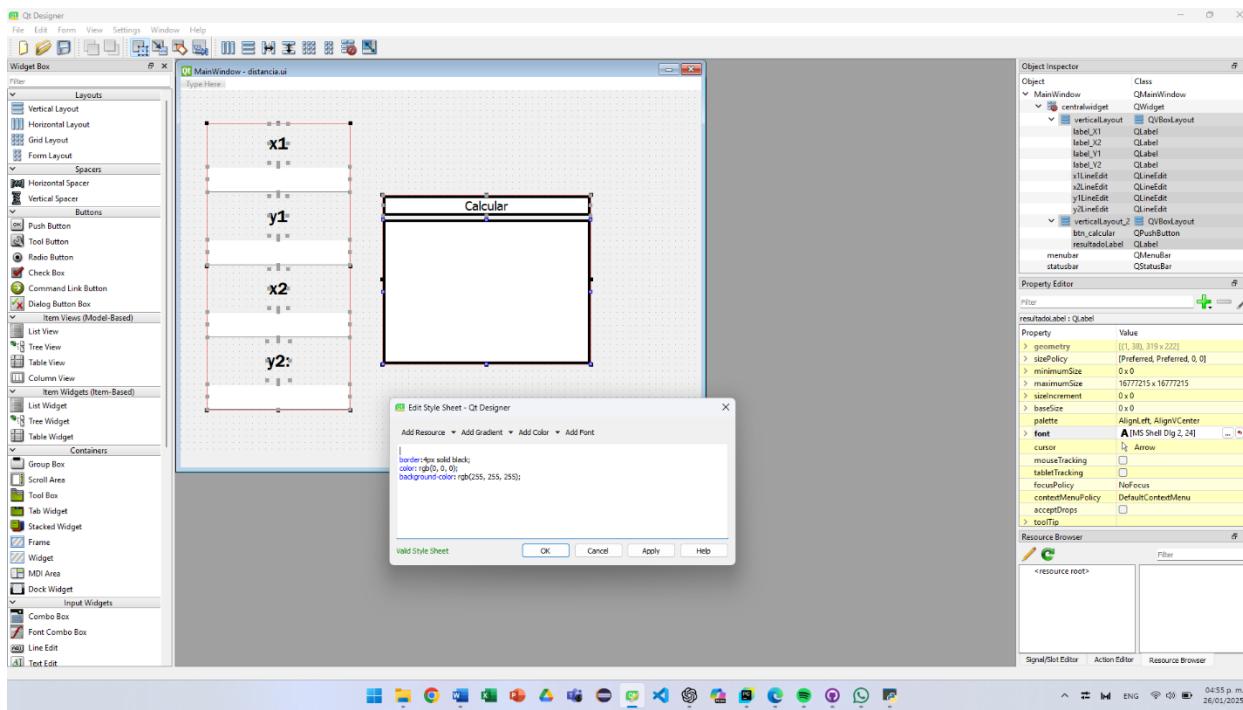
    def calcular_distancia(self):
        try:
            # Obtener valores de los campos de entrada
            x1 = float(self.xLineEdit.text())
            y1 = float(self.yLineEdit.text())
            x2 = float(self.x2LineEdit.text())
            y2 = float(self.y2LineEdit.text())

            # Calcular la distancia entre los puntos
            distancia = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

            # Mostrar el resultado
            self.resultadoLabel.setText(f"Distancia: {distancia:.2f}")
        except:
            # Mostrar un mensaje de error simple
            self.resultadoLabel.setText("Error: ingresa números válidos.")

    # Iniciar la aplicación
    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        ventana = DistanciaApp()
        ventana.show()
        sys.exit(app.exec_())

```



Este código es una aplicación en Python utilizando la biblioteca PyQt5 para crear una interfaz gráfica que permite calcular la distancia entre dos puntos en un plano cartesiano. La interfaz es cargada desde un archivo .ui, lo que facilita la creación y diseño de los elementos visuales sin necesidad de codificarlos manualmente.



La aplicación funciona a través de una clase llamada DistanciaApp, que hereda de QMainWindow, lo que significa que es una ventana principal en la que se integran distintos widgets. En el constructor de esta clase (`__init__`), se carga el diseño de la interfaz y se establece la conexión entre un botón (`btn_calcular`) y una función (`calcular_distancia`). Esto quiere decir que, cuando el usuario presiona el botón, se ejecuta la función que realiza el cálculo de la distancia.

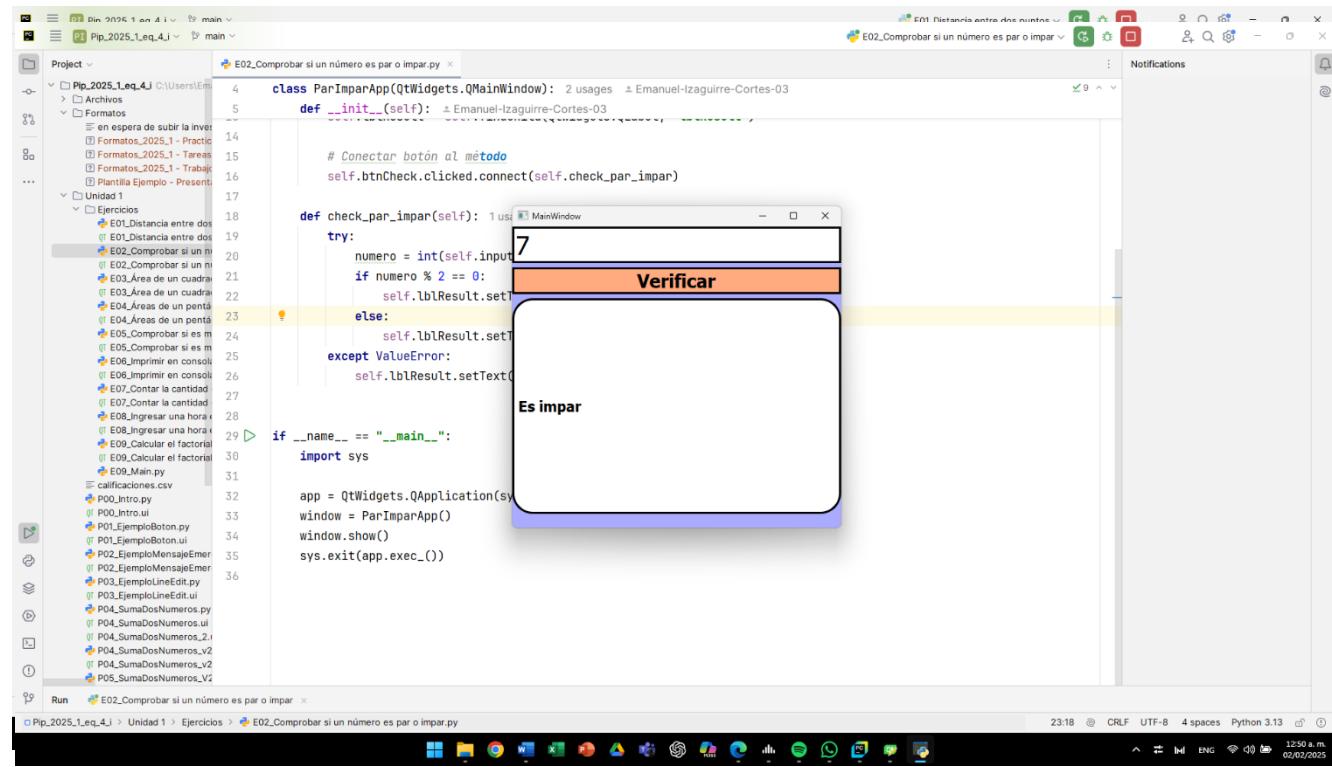
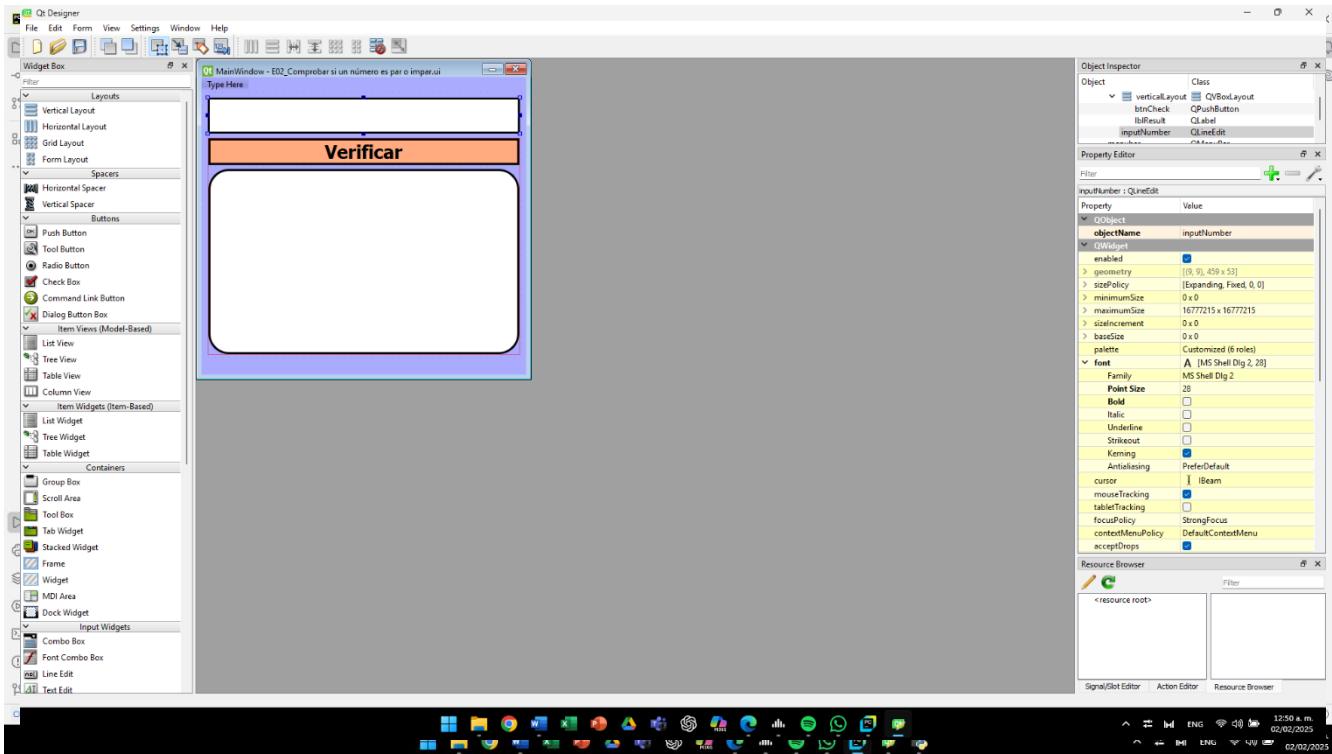
La función `calcular_distancia` obtiene los valores ingresados en los campos de texto, los convierte en números y aplica la fórmula de la distancia euclídea entre dos puntos, que es la raíz cuadrada de la suma de los cuadrados de las diferencias de las coordenadas. Luego, el resultado es mostrado en una etiqueta (QLabel). Si los datos ingresados no son válidos, se captura la excepción y se muestra un mensaje de error en la interfaz.

Finalmente, en la parte principal del código, se crea una instancia de QApplication, que es necesaria para ejecutar cualquier aplicación de PyQt5. Luego, se crea una ventana de DistanciaApp, se muestra en pantalla y se inicia el bucle de eventos con `app.exec_()`, lo que permite que la interfaz sea interactiva hasta que el usuario la cierre.

En resumen, este código permite a un usuario ingresar coordenadas de dos puntos en un plano, calcular la distancia entre ellos y ver el resultado en la pantalla.



## E02\_Comprobar si un número es par o impar





Este código es una aplicación en Python con PyQt5 que permite comprobar si un número ingresado por el usuario es par o impar. Para ello, utiliza una interfaz gráfica definida en un archivo .ui, la cual se carga dinámicamente al ejecutar la aplicación.

La aplicación está estructurada en una clase llamada ParImparApp, que hereda de QMainWindow, lo que significa que es una ventana principal en la que se colocan los distintos elementos gráficos. Dentro del constructor (`__init__`), se carga la interfaz y se vinculan los widgets a variables que permiten su manipulación. En este caso, QLineEdit se usa para capturar el número ingresado, QPushButton actúa como el botón que activa la verificación, y QLabel muestra el resultado en pantalla.

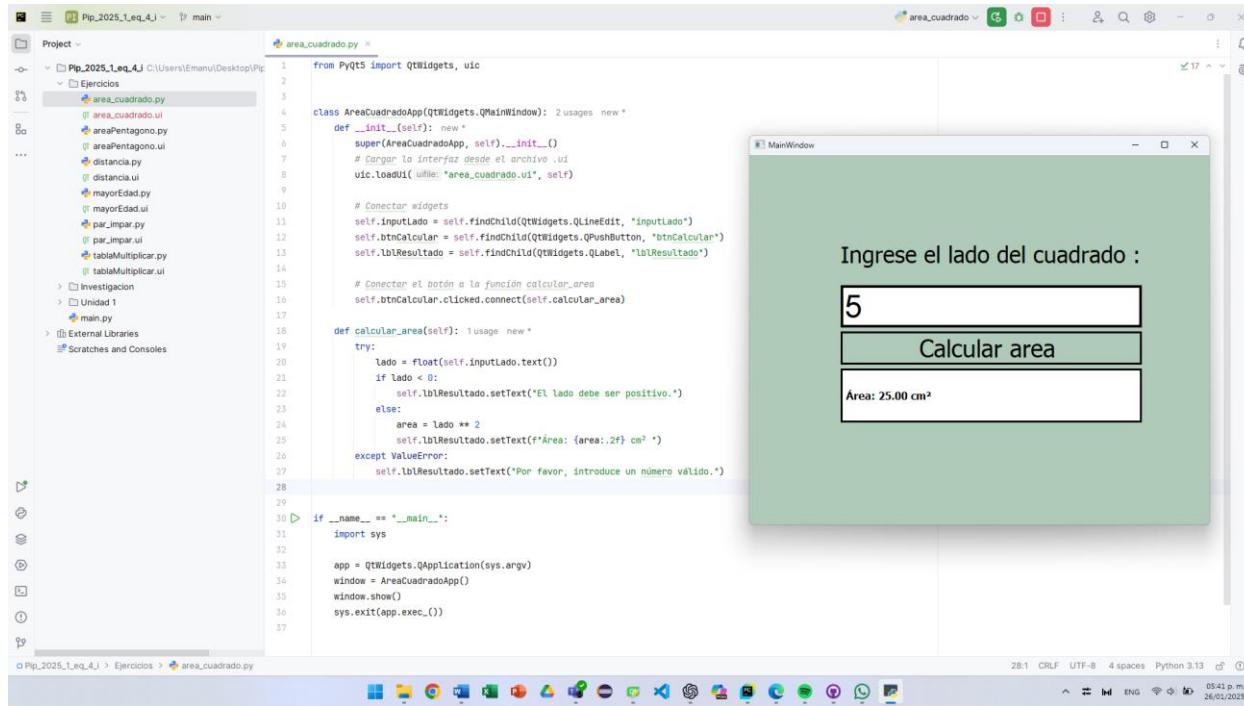
Cuando el usuario introduce un número y presiona el botón, la función `check_par_impar()` se ejecuta. Esta función obtiene el valor del campo de texto y lo convierte en un número entero. Luego, aplica una operación matemática sencilla: verifica si el número es divisible por 2 (`numero % 2 == 0`). Si la condición se cumple, se muestra el mensaje "Es par" en la etiqueta, de lo contrario, se muestra "Es impar". En caso de que el usuario ingrese un valor no numérico, el programa captura el error y muestra un mensaje indicando que se debe ingresar un número válido.

Para ejecutar la aplicación, el código principal crea una instancia de QApplication, que es necesaria para manejar los eventos de PyQt5. Luego, se instancia ParImparApp, se muestra la ventana y se ejecuta el bucle de eventos con `app.exec_()`, lo que permite que la aplicación responda a las interacciones del usuario.

En resumen, este código proporciona una interfaz sencilla e intuitiva para comprobar si un número es par o impar, utilizando PyQt5 para gestionar los elementos gráficos y las interacciones del usuario.



## E03\_Area de un cuadrado



```
from PyQt5 import QtWidgets, uic
import sys

class AreaCuadradoApp(QtWidgets.QMainWindow):
    def __init__(self):
        super(AreaCuadradoApp, self).__init__()
        # Cargar la interfaz desde el archivo .ui
        self.loadUi('area_cuadrado.ui', self)

        # Conectar los widgets
        self.inputLado = self.findChild(QtWidgets.QLineEdit, "inputLado")
        self.btnCalcular = self.findChild(QtWidgets.QPushButton, "btnCalcular")
        self.lblResultado = self.findChild(QtWidgets.QLabel, "lblResultado")

        # Conectar el botón a la función calcular_area
        self.btnCalcular.clicked.connect(self.calcular_area)

    def calcular_area(self):
        try:
            lado = float(self.inputLado.text())
            if lado < 0:
                self.lblResultado.setText("El lado debe ser positivo.")
            else:
                area = lado ** 2
                self.lblResultado.setText(f"Área: {area:.2f} cm²")
        except ValueError:
            self.lblResultado.setText("Por favor, introduce un número válido.")

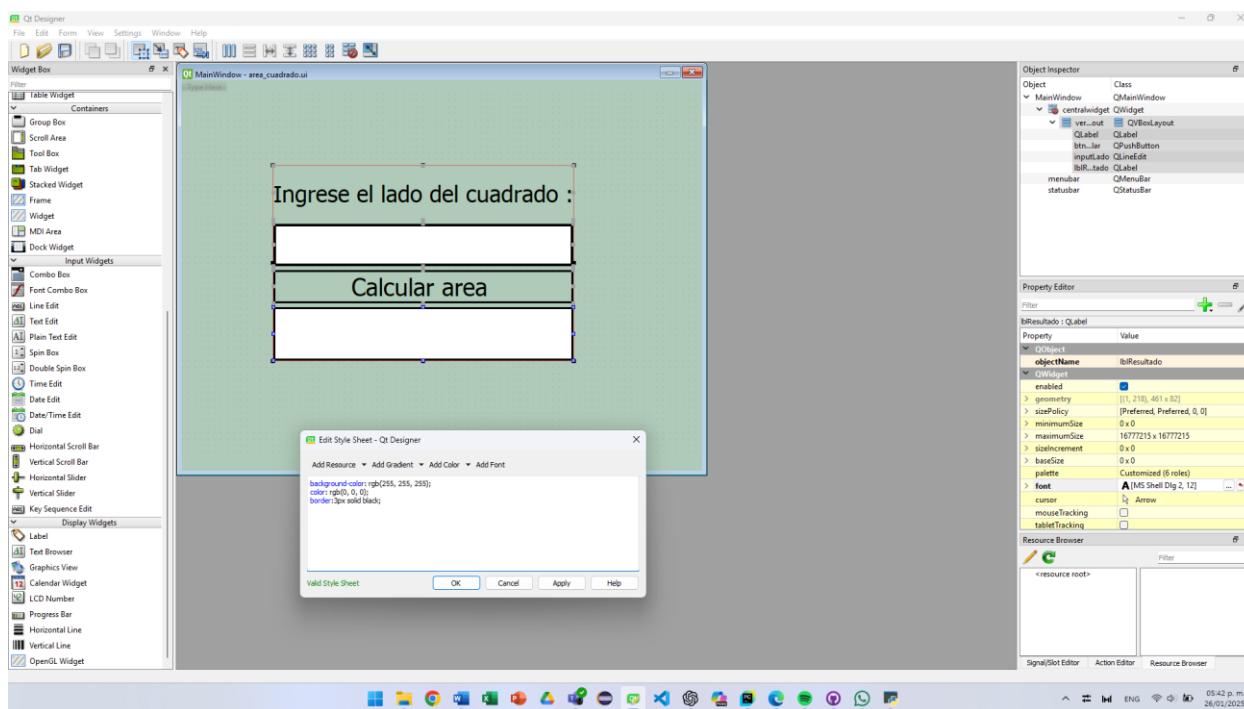
if __name__ == "__main__":
    import sys

    app = QtWidgets.QApplication(sys.argv)
    window = AreaCuadradoApp()
    window.show()
    sys.exit(app.exec_())
```

Ingrese el lado del cuadrado :

5  
Calcular area

Área: 25.00 cm<sup>2</sup>



Este código es una aplicación desarrollada en Python con PyQt5 que permite calcular el área de un cuadrado a partir del valor ingresado por el usuario. La interfaz gráfica está diseñada en un archivo .ui, lo que permite separar el diseño visual de la lógica del programa.



La estructura del código se basa en la clase `AreaCuadradoApp`, que hereda de `QMainWindow`, lo que significa que la ventana principal de la aplicación se gestiona a través de esta clase. En su método `__init__`, se carga la interfaz de usuario utilizando `uic.loadUi()`, asegurando que los elementos gráficos sean accesibles desde la lógica del programa.

Dentro del constructor, se utilizan `findChild()` para localizar los elementos clave de la interfaz:

- `QLineEdit (inputLado)` permite al usuario ingresar la longitud del lado del cuadrado.
- `QPushButton (btnCalcular)` actúa como un botón que, al ser presionado, activa el cálculo del área.
- `QLabel (lblResultado)` se usa para mostrar el resultado o mensajes de error.

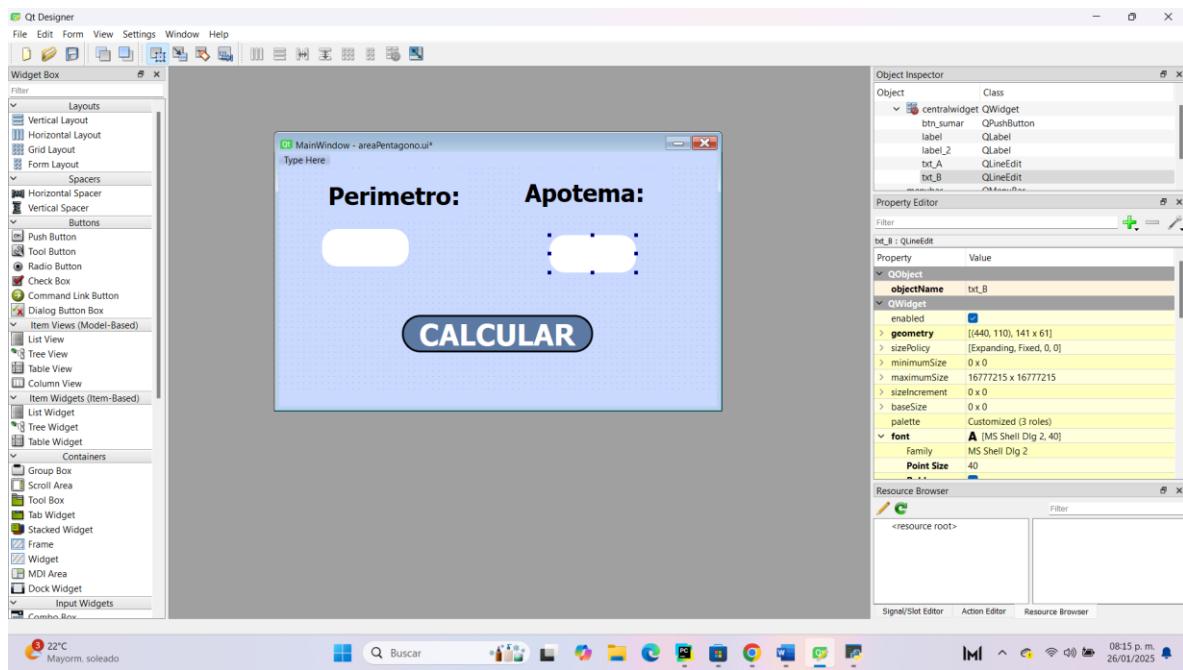
Cuando el usuario ingresa un valor y presiona el botón, se ejecuta la función `calcular_area()`. En esta función, primero se obtiene el valor ingresado en el campo de texto y se intenta convertir en un número de tipo float. Luego, se realiza una verificación para asegurarse de que el número sea positivo, ya que una longitud negativa no tiene sentido en este contexto. Si el valor es válido, se calcula el área utilizando la fórmula  $A=L^2$  y se muestra el resultado en la etiqueta de la interfaz. Si el usuario introduce un valor inválido (como letras o símbolos no numéricos), se captura el error y se muestra un mensaje de advertencia en la etiqueta.

Finalmente, la ejecución del programa comienza en la sección `if __name__ == "__main__":` donde se crea una instancia de `QApplication`, que es necesaria para manejar la interfaz gráfica de PyQt5. Luego, se instancia `AreaCuadradoApp`, se muestra la ventana de la aplicación y se inicia el bucle de eventos con `app.exec_()`, lo que permite que la aplicación sea interactiva y responda a las acciones del usuario.

En resumen, este código es una aplicación sencilla pero funcional que permite calcular el área de un cuadrado con una interfaz gráfica amigable. Se estructura de manera modular, separando la interfaz del cálculo matemático, lo que facilita su mantenimiento y posible expansión en el futuro.



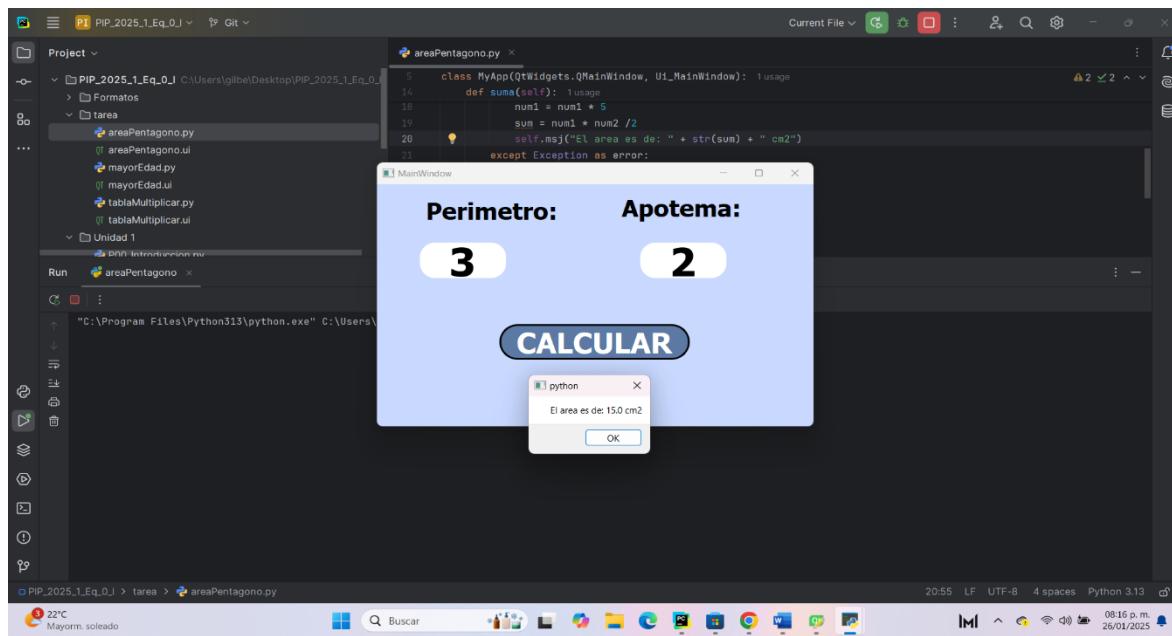
## E04\_Areas de un pentágono



PIP\_2025\_1\_Eq\_0\_J

areaPentagono.py

```
1 > import ...
2 qtCreatorfile = "AreaPentagono.ui" # Nombre del archivo aquí.
3 UI_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)
4 class MyApp(QtWidgets.QMainWindow, UI_MainWindow):
5     def __init__(self):
6         QtWidgets.QMainWindow.__init__(self)
7         UI_MainWindow.__init__(self)
8         self.setupUi(self)
9         # Área de los Signals
10        self.btn_sumar.clicked.connect(self.suma)
11
12        # Área de los Slots
13    def suma(self): 1 usage
14        try:
15            num1 = float(self.txt_A.text())
16            num2 = float(self.txt_B.text())
17            num1 = num1 * 5
18            sum = num1 * num2 / 2
19            self.msj("El área es de: " + str(sum) + " cm²")
20        except Exception as error:
21            print(error)
22
23
24    def msj(self, txt): 1 usage
25        m = QtWidgets.QMessageBox()
26        m.setText(txt)
27        m.exec_()
28
29    if __name__ == "__main__":
30        app = QtWidgets.QApplication(sys.argv)
31        window = MyApp()
32        window.show()
33        sys.exit(app.exec_())
34
```



Este ejercicio consiste en crear una aplicación gráfica utilizando Python y la biblioteca PyQt5 para sumar dos números ingresados por el usuario. A continuación, se detallan los componentes y su propósito:

#### 1. Importaciones:

- sys: Permite acceder a funciones y parámetros del sistema.
- PyQt5: Biblioteca para crear interfaces gráficas. uic se usa para cargar archivos .ui y QtWidgets contiene los widgets necesarios.

#### 2. Carga del archivo de interfaz:

- qtCreatorFile: Nombre del archivo de interfaz gráfica (P04\_SumaDosNumeros.ui).
- uic.loadUiType(qtCreatorFile): Carga el diseño de la interfaz desde el archivo .ui.

#### 3. Definición de la clase principal:

- MyApp: Clase principal que hereda de QMainWindow y Ui\_MainWindow.
- \_\_init\_\_: Constructor que inicializa la ventana principal y configura la interfaz.
- self.btn\_sumar.clicked.connect(self.sumar): Conecta el botón de sumar con la función sumar.

#### 4. Función para sumar:

- sumar: Función que se ejecuta al hacer clic en el botón de sumar.
- a y b: Obtienen los valores ingresados en los campos de texto y los convierten a float.



- r: Calcula la suma de a y b.
- self.msg: Muestra un mensaje con el resultado de la suma.
- except: Captura y muestra cualquier error que ocurra durante la ejecución.

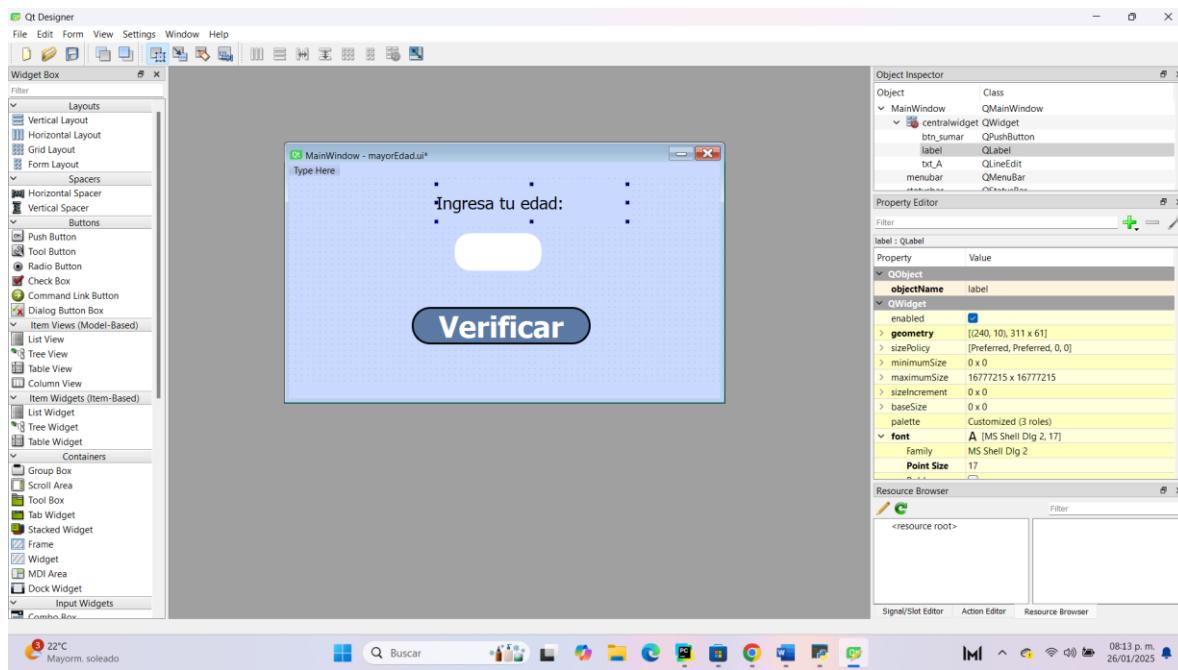
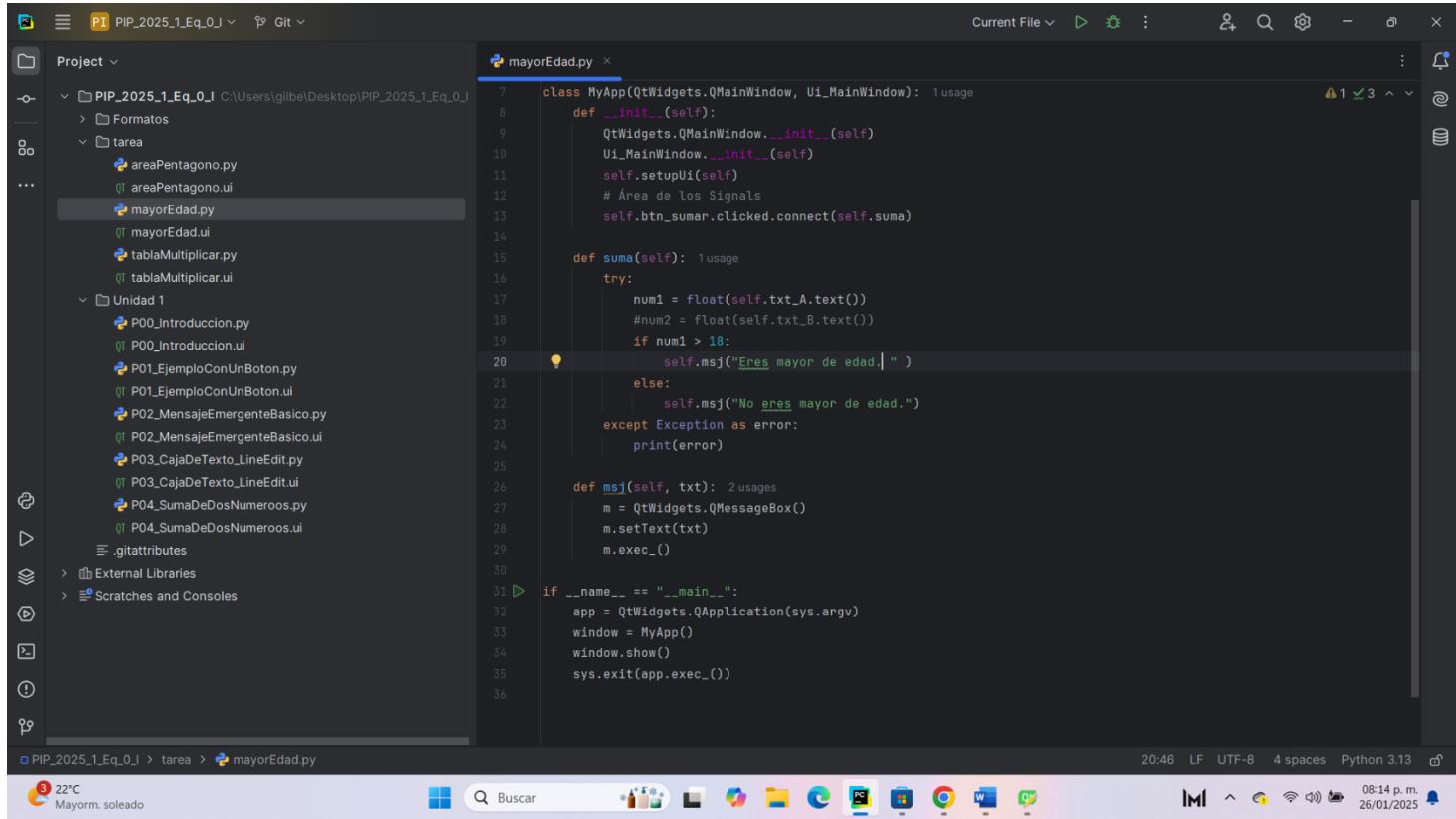
**5. Función para mostrar mensajes:**

- msg: Muestra un cuadro de mensaje con el texto proporcionado.

**6. Ejecución de la aplicación:**

- if \_\_name\_\_ == "\_\_main\_\_": Asegura que el código solo se ejecute si el archivo se ejecuta directamente.
- app = QtWidgets.QApplication(sys.argv): Crea una instancia de la aplicación.
- window = MyApp(): Crea una instancia de la ventana principal.
- window.show(): Muestra la ventana.
- sys.exit(app.exec\_()): Inicia el bucle de eventos de la aplicación.

## E05\_Comprobar si es mayor de edad una persona

```

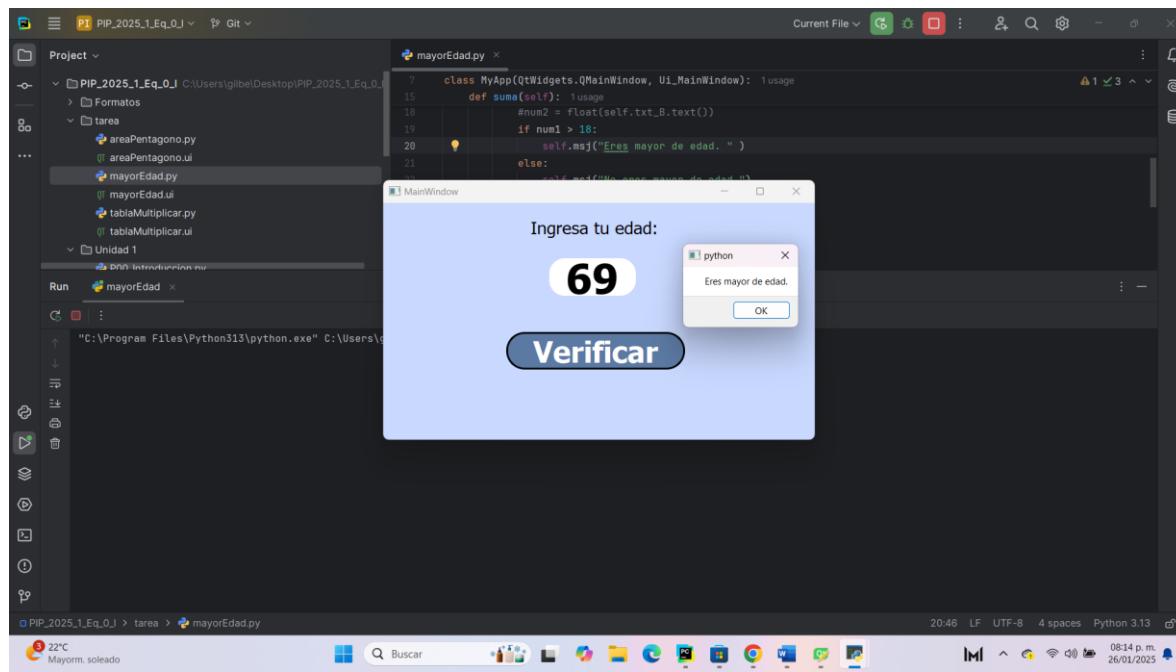
class MyApp(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        # Área de los Signals
        self.btn_sumar.clicked.connect(self.suma)

    def suma(self):
        try:
            num1 = float(self.txt_A.text())
            num2 = float(self.txt_B.text())
            if num1 > 18:
                self.msj("Eres mayor de edad. ")
            else:
                self.msj("No eres mayor de edad.")
        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_())

```



Este ejercicio, P05\_SumaDosNumeros\_V2.py, es una versión mejorada del ejercicio anterior y también utiliza Python y PyQt5 para crear una aplicación gráfica que suma dos números ingresados por el usuario. A continuación, se detallan los componentes y su propósito:

Este ejercicio consiste en crear una aplicación gráfica utilizando Python y la biblioteca PyQt5 para sumar dos números ingresados por el usuario. A continuación, se detallan los componentes y su propósito:

#### 1. Importaciones:

- sys: Permite acceder a funciones y parámetros del sistema.
- PyQt5: Biblioteca para crear interfaces gráficas. uic se usa para cargar archivos .ui y QtWidgets contiene los widgets necesarios.

#### 2. Carga del archivo de interfaz:

- qtCreatorFile: Nombre del archivo de interfaz gráfica (P05\_SumaDosNumeros\_V2.ui).
- uic.loadUiType(qtCreatorFile): Carga el diseño de la interfaz desde el archivo .ui.

#### 3. Definición de la clase principal:

- MyApp: Clase principal que hereda de QMainWindow y Ui\_MainWindow.
- \_\_init\_\_: Constructor que inicializa la ventana principal y configura la interfaz.
- self.btn\_sumar.clicked.connect(self.sumar): Conecta el botón de sumar con la función sumar.

#### 4. Función para sumar:

- sumar: Función que se ejecuta al hacer clic en el botón de sumar.
- a y b: Obtienen los valores ingresados en los campos de texto y los convierten a float.
- r: Calcula la suma de a y b.



- self.txt\_resultado.setText(str(r)): Muestra el resultado de la suma en un campo de texto en lugar de un cuadro de mensaje.
- except: Captura y muestra cualquier error que ocurra durante la ejecución.

#### 5. Función para mostrar mensajes:

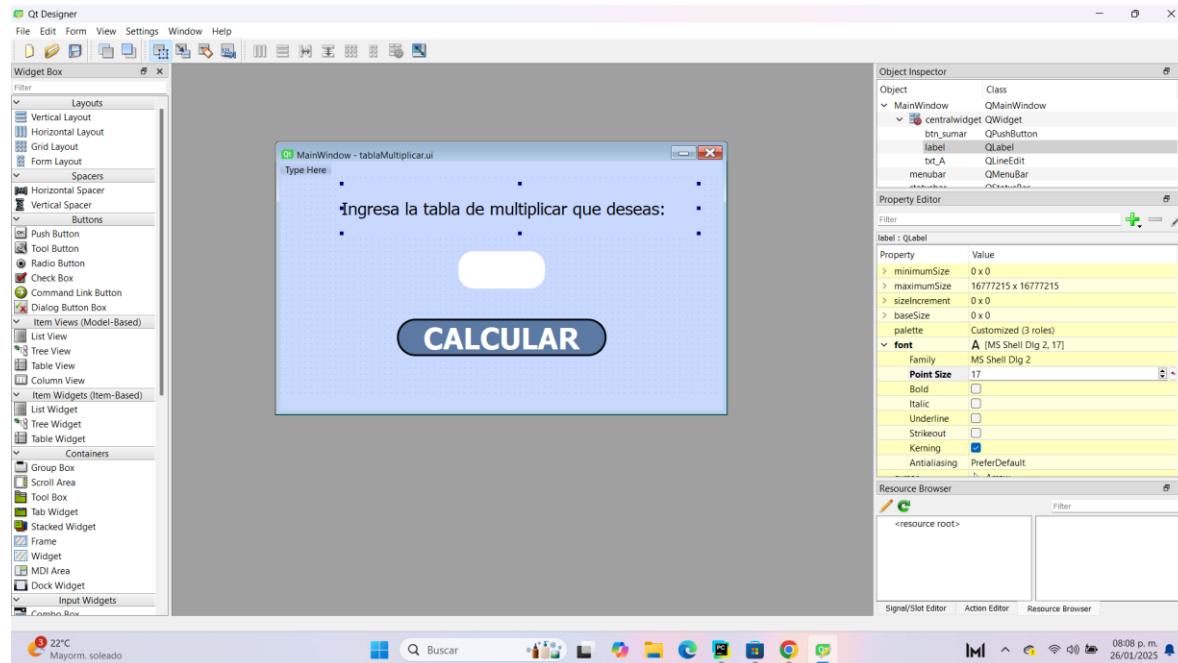
- msj: Muestra un cuadro de mensaje con el texto proporcionado. Aunque esta función no se usa en esta versión, está definida por si se necesita en el futuro.

#### 6. Ejecución de la aplicación:

- if \_\_name\_\_ == "\_\_main\_\_": Asegura que el código solo se ejecute si el archivo se ejecuta directamente.
- app = QtWidgets.QApplication(sys.argv): Crea una instancia de la aplicación.
- window = MyApp(): Crea una instancia de la ventana principal.
- window.show(): Muestra la ventana.
- sys.exit(app.exec\_()): Inicia el bucle de eventos de la aplicación.

Este ejercicio demuestra cómo crear una aplicación gráfica simple en Python usando PyQt5 para sumar dos números ingresados por el usuario, con la mejora de mostrar el resultado directamente en la interfaz en lugar de un cuadro de mensaje.

#### E06\_Imprimir en consola/MessageBox una tabla de multiplicar



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

- Project:** PIP\_2025\_1\_Eq\_0\_J
- Current File:** tablaMultiplicar.py
- Code Content (tablaMultiplicar.py):**

```
1 import sys
2 from PyQt5 import uic, QtWidgets
3 qtCreatorFile = "tablaMultiplicar.ui" # Nombre del archivo aquí.
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        # Área de los Signals
11        self.btn_sumar.clicked.connect(self.suma)
12
13        # Área de los Slots
14    def suma(self): 1 usage
15        try:
16            num1 = int(self.txt_A.text())
17            resultado = f"Tabla de multiplicar del {num1}\n"
18
19            for i in range(1, 11):
20                resultado += f"\n{num1} x {i} = {num1 * i}"
21
22            self.msj(resultado) | 2 usages
23        except ValueError:
24            self.msj("Por favor, ingrese un número válido.")
25        except Exception as error:
26            print(error)
27
28    def msj(self, txt): 2 usages
29        m = QtWidgets.QMessageBox()
30        m.setText(txt)
31        m.exec_()
32
```

- Toolbars and Status Bar:** Current File, Git, Project, External Libraries, Scratches and Consoles.
- Bottom Status Bar:** 22:33, LF, UTF-8, 4 spaces, Python 3.10, 22°C, Mayor, soleado, 08:10 p. m., 26/01/2025.

Este ejercicio, P06\_PromedioNumeros-Save.py, crea una aplicación gráfica utilizando Python y PyQt5 para calcular y guardar el promedio de una serie de calificaciones ingresadas por el usuario. A continuación, se detallan los componentes y su propósito:

## 1. Importaciones:

- sys: Permite acceder a funciones y parámetros del sistema.



- PyQt5: Biblioteca para crear interfaces gráficas. uic se usa para cargar archivos .ui y QtWidgets contiene los widgets necesarios.

## 2. Carga del archivo de interfaz:

- qtCreatorFile: Nombre del archivo de interfaz gráfica (P06\_PromedioNumeros-Save.ui).
- uic.loadUiType(qtCreatorFile): Carga el diseño de la interfaz desde el archivo .ui.

## 3. Definición de la clase principal:

- MyApp: Clase principal que hereda de QMainWindow y Ui\_MainWindow.
- \_\_init\_\_: Constructor que inicializa la ventana principal y configura la interfaz.
- self.btn\_agregar.clicked.connect(self.agregar): Conecta el botón de agregar con la función agregar.
- self.btn\_guardar.clicked.connect(self.guardar): Conecta el botón de guardar con la función guardar.
- self.calificaciones: Lista para almacenar las calificaciones ingresadas.

## 4. Función para agregar calificaciones:

- agregar: Función que se ejecuta al hacer clic en el botón de agregar.
- calificacion: Obtiene el valor ingresado en el campo de texto y lo convierte a int.
- self.calificaciones.append(calificacion): Agrega la calificación a la lista.
- prom: Calcula el promedio de las calificaciones.
- self.txt\_promedio.setText(str(prom)): Muestra el promedio en un campo de texto.

## 5. Función para guardar calificaciones:

- guardar: Función que se ejecuta al hacer clic en el botón de guardar.
- archivo = open("../Archivos/calificaciones.csv", "w"): Abre un archivo para escribir las calificaciones.
- for c in self.calificaciones: Escribe cada calificación en el archivo.
- archivo.flush() y archivo.close(): Aseguran que los datos se guarden correctamente.
- self.msj("Archivo Guardado con Exito!"): Muestra un mensaje indicando que el archivo se guardó con éxito.

## 6. Función para mostrar mensajes:

- msj: Muestra un cuadro de mensaje con el texto proporcionado.

## 7. Ejecución de la aplicación:

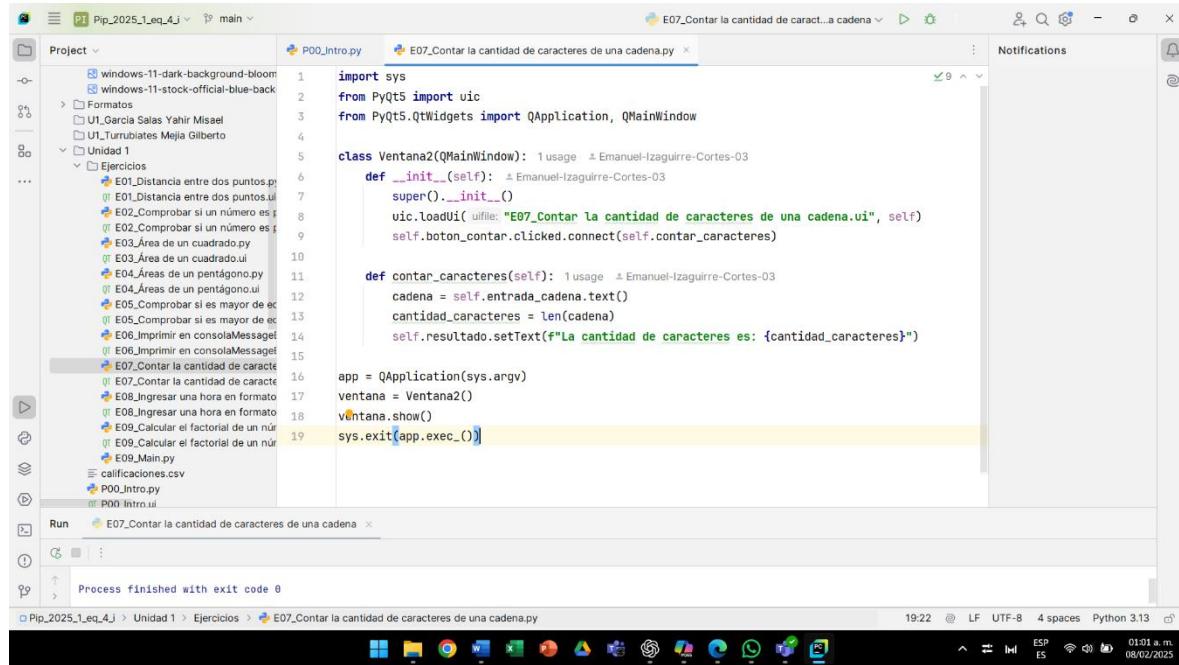


- if \_\_name\_\_ == "\_\_main\_\_": Asegura que el código solo se ejecute si el archivo se ejecuta directamente.
- app = QtWidgets.QApplication(sys.argv): Crea una instancia de la aplicación.
- window = MyApp(): Crea una instancia de la ventana principal.
- window.show(): Muestra la ventana.
- sys.exit(app.exec\_()): Inicia el bucle de eventos de la aplicación.

---

**Conclusión:** Este ejercicio demuestra cómo crear una aplicación gráfica en Python usando PyQt5 para calcular y guardar el promedio de una serie de calificaciones ingresadas por el usuario. La aplicación permite agregar calificaciones, calcular el promedio en tiempo real y guardar las calificaciones en un archivo CSV. Esto es útil para gestionar y almacenar datos de manera eficiente.

#### E07\_cantidad de caracteres de una cadena



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

- Project View:** Shows a tree structure of files under "Unidad 1".
- Code Editor:** Displays the Python file `E07_Contar la cantidad de caracteres de una cadena.py`. The code is as follows:

```
import sys
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow

class Ventana2(QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi("E07_Contar la cantidad de caracteres de una cadena.ui", self)
        self.boton_contar.clicked.connect(self.contar_caracteres)

    def contar_caracteres(self):
        cadena = self.entrada_cadena.text()
        cantidad_caracteres = len(cadena)
        self.resultado.setText(f"La cantidad de caracteres es: {cantidad_caracteres}")

app = QApplication(sys.argv)
ventana = Ventana2()
ventana.show()
sys.exit(app.exec_())
```

- Run Tab:** Shows the output: "Process finished with exit code 0".
- System Tray:** Shows the date and time: "08/02/2025 01:01 a.m."



```
import sys
from PyQt5 import uic
from PyQt5.QtWidgets import QApplication, QMainWindow
```

Este ejercicio, P07\_PromedioNumeros-Load.py, crea una aplicación gráfica utilizando Python y PyQt5 para calcular, cargar y guardar el promedio de una serie de calificaciones ingresadas por el usuario. A continuación, se detallan los componentes y su propósito:

---

#### 1. Importaciones:

- sys: Permite acceder a funciones y parámetros del sistema.
- PyQt5: Biblioteca para crear interfaces gráficas. uic se usa para cargar archivos .ui y QtWidgets contiene los widgets necesarios.

#### 2. Carga del archivo de interfaz:

- qtCreatorFile: Nombre del archivo de interfaz gráfica (P07\_PromedioNumeros-Load.ui).
- uic.loadUiType(qtCreatorFile): Carga el diseño de la interfaz desde el archivo .ui.

#### 3. Definición de la clase principal:

- MyApp: Clase principal que hereda de QMainWindow y Ui\_MainWindow.
- \_\_init\_\_: Constructor que inicializa la ventana principal y configura la interfaz.
- self.btn\_cargar.clicked.connect(self.cargar): Conecta el botón de cargar con la función cargar.
- self.btn\_agregar.clicked.connect(self.agregar): Conecta el botón de agregar con la función agregar.
- self.btn\_guardar.clicked.connect(self.guardar): Conecta el botón de guardar con la función guardar.
- self.calificaciones: Lista para almacenar las calificaciones ingresadas.

#### 4. Función para cargar calificaciones:

- cargar: Función que se ejecuta al hacer clic en el botón de cargar.
- archivo = open("../Archivos/calificaciones.csv"): Abre el archivo de calificaciones.



- contenido = archivo.readlines(): Lee todas las líneas del archivo.
- datos = [int(x) for x in contenido]: Convierte cada línea en un entero y las almacena en una lista.
- self.calificaciones = datos: Asigna los datos cargados a la lista de calificaciones.
- self.promedio(): Calcula y muestra el promedio de las calificaciones.

**5. Función para agregar calificaciones:**

- agregar: Función que se ejecuta al hacer clic en el botón de agregar.
- calificacion: Obtiene el valor ingresado en el campo de texto y lo convierte a int.
- self.calificaciones.append(calificacion): Agrega la calificación a la lista.
- self.promedio(): Calcula y muestra el promedio de las calificaciones.

**6. Función para calcular el promedio:**

- promedio: Calcula el promedio de las calificaciones.
- prom = sum(self.calificaciones) / len(self.calificaciones): Calcula el promedio.
- self.txt\_promedio.setText(str(prom)): Muestra el promedio en un campo de texto.

**7. Función para guardar calificaciones:**

- guardar: Función que se ejecuta al hacer clic en el botón de guardar.
- archivo = open("../Archivos/calificaciones.csv", "w"): Abre un archivo para escribir las calificaciones.
- for c in self.calificaciones: Escribe cada calificación en el archivo.
- archivo.flush() y archivo.close(): Aseguran que los datos se guarden correctamente.
- self.msj("Archivo Guardado con Exito!"): Muestra un mensaje indicando que el archivo se guardó con éxito.

**8. Función para mostrar mensajes:**

- msj: Muestra un cuadro de mensaje con el texto proporcionado.

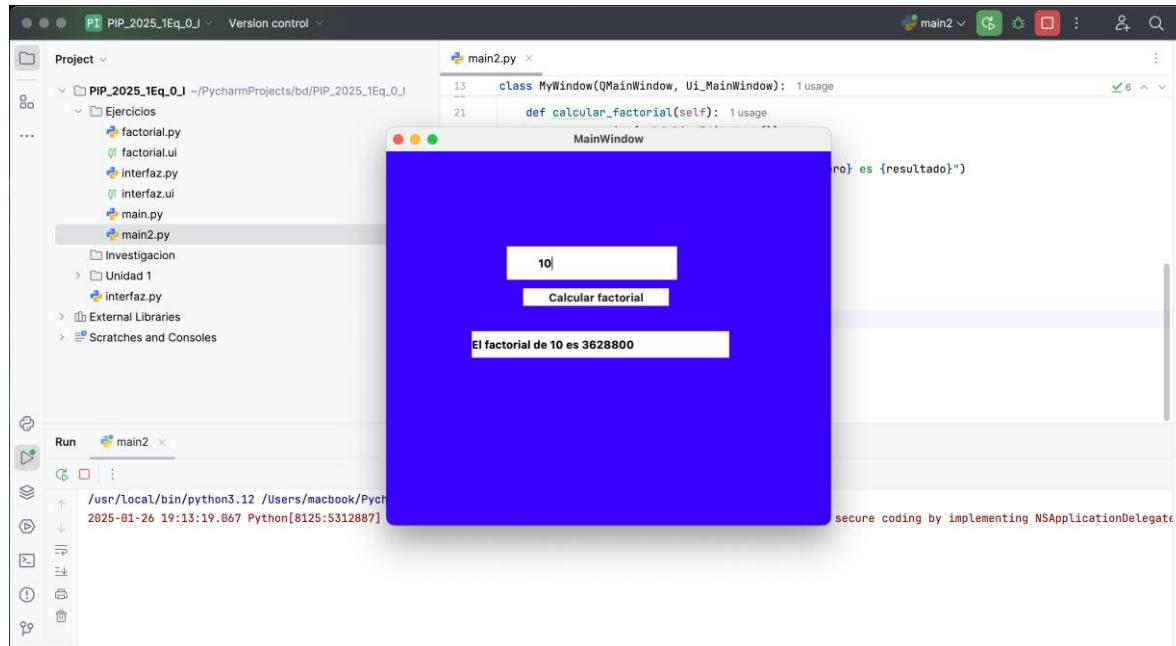
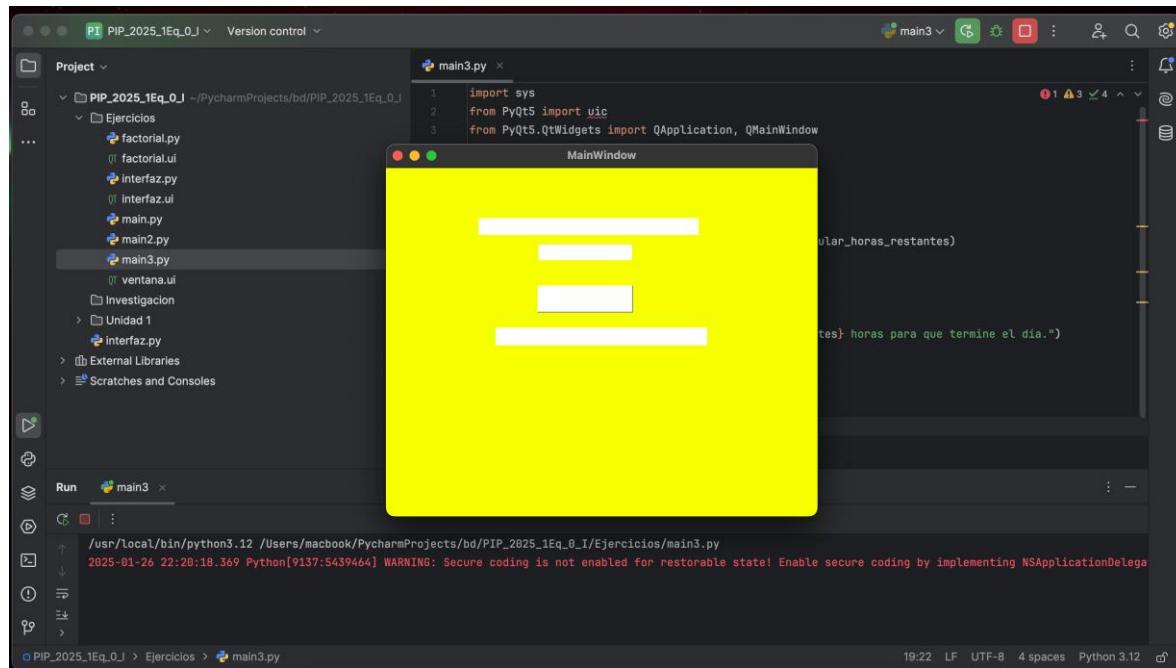
**9. Ejecución de la aplicación:**

- if \_\_name\_\_ == "\_\_main\_\_": Asegura que el código solo se ejecute si el archivo se ejecuta directamente.
- app = QtWidgets.QApplication(sys.argv): Crea una instancia de la aplicación.
- window = MyApp(): Crea una instancia de la ventana principal.
- window.show(): Muestra la ventana.
- sys.exit(app.exec\_()): Inicia el bucle de eventos de la aplicación.

---

**Conclusión:** Este ejercicio demuestra cómo crear una aplicación gráfica en Python usando PyQt5 para calcular, cargar y guardar el promedio de una serie de calificaciones ingresadas por el usuario. La aplicación permite cargar calificaciones desde un archivo, agregar nuevas calificaciones, calcular el promedio en tiempo real y guardar las calificaciones en un archivo CSV. Esto es útil para gestionar y almacenar datos de manera eficiente, permitiendo la reutilización y actualización de los datos.

E08\_Cuantas horas le restan al día para terminarse



Este ejercicio, P08\_PromedioNumeros-Load\_V2.py, crea una aplicación gráfica utilizando Python y PyQt5 para calcular, cargar y guardar el promedio de una serie de calificaciones ingresadas por el usuario. A continuación, se detallan los componentes y su propósito:



**1. Importaciones:**

- sys: Permite acceder a funciones y parámetros del sistema.
- PyQt5: Biblioteca para crear interfaces gráficas. uic se usa para cargar archivos .ui y QtWidgets contiene los widgets necesarios.

**2. Carga del archivo de interfaz:**

- qtCreatorFile: Nombre del archivo de interfaz gráfica (P08\_PromedioNumeros-Load\_V2.ui).
- uic.loadUiType(qtCreatorFile): Carga el diseño de la interfaz desde el archivo .ui.

**3. Definición de la clase principal:**

- MyApp: Clase principal que hereda de QMainWindow y Ui\_MainWindow.
- \_\_init\_\_: Constructor que inicializa la ventana principal y configura la interfaz.
- self.btn\_cargar.clicked.connect(self.cargar): Conecta el botón de cargar con la función cargar.
- self.btn\_agregar.clicked.connect(self.agregar): Conecta el botón de agregar con la función agregar.
- self.btn\_guardar.clicked.connect(self.guardar): Conecta el botón de guardar con la función guardar.
- self.calificaciones: Lista para almacenar las calificaciones ingresadas.

**4. Función para cargar calificaciones:**

- cargar: Función que se ejecuta al hacer clic en el botón de cargar.
- archivo = open("../Archivos/calificaciones.csv"): Abre el archivo de calificaciones.
- contenido = archivo.readlines(): Lee todas las líneas del archivo.
- datos = [int(x) for x in contenido]: Convierte cada línea en un entero y las almacena en una lista.
- self.calificaciones = datos: Asigna los datos cargados a la lista de calificaciones.
- self.promedio(): Calcula y muestra el promedio de las calificaciones.
- self.txt\_lista\_calificaciones.setText(str(self.calificaciones)): Muestra la lista de calificaciones en un campo de texto.

**5. Función para agregar calificaciones:**

- agregar: Función que se ejecuta al hacer clic en el botón de agregar.
- calificacion: Obtiene el valor ingresado en el campo de texto y lo convierte a int.
- self.calificaciones.append(calificacion): Agrega la calificación a la lista.
- self.promedio(): Calcula y muestra el promedio de las calificaciones.
- self.txt\_lista\_calificaciones.setText(str(self.calificaciones)): Muestra la lista de calificaciones en un campo de texto.

**6. Función para calcular el promedio:**

- promedio: Calcula el promedio de las calificaciones.
- prom = sum(self.calificaciones) / len(self.calificaciones): Calcula el promedio.
- self.txt\_promedio.setText(str(prom)): Muestra el promedio en un campo de texto.

**7. Función para guardar calificaciones:**

- guardar: Función que se ejecuta al hacer clic en el botón de guardar.
- archivo = open("../Archivos/calificaciones.csv", "w"): Abre un archivo para escribir las calificaciones.



- for c in self.calificaciones: Escribe cada calificación en el archivo.
- archivo.flush() y archivo.close(): Aseguran que los datos se guarden correctamente.
- self.msj("Archivo Guardado con Exito!"): Muestra un mensaje indicando que el archivo se guardó con éxito.

#### 8. Función para mostrar mensajes:

- msj: Muestra un cuadro de mensaje con el texto proporcionado.

#### 9. Ejecución de la aplicación:

- if \_\_name\_\_ == "\_\_main\_\_": Asegura que el código solo se ejecute si el archivo se ejecuta directamente.
- app = QtWidgets.QApplication(sys.argv): Crea una instancia de la aplicación.
- window = MyApp(): Crea una instancia de la ventana principal.
- window.show(): Muestra la ventana.
- sys.exit(app.exec\_()): Inicia el bucle de eventos de la aplicación.

**Conclusión:** Este ejercicio demuestra cómo crear una aplicación gráfica en Python usando PyQt5 para calcular, cargar y guardar el promedio de una serie de calificaciones ingresadas por el usuario. La aplicación permite cargar calificaciones desde un archivo, agregar nuevas calificaciones, calcular el promedio en tiempo real y guardar las calificaciones en un archivo CSV. Además, muestra la lista de calificaciones en la interfaz, lo que facilita la gestión y visualización de los datos

#### E09\_Calcular la factorial de un número

