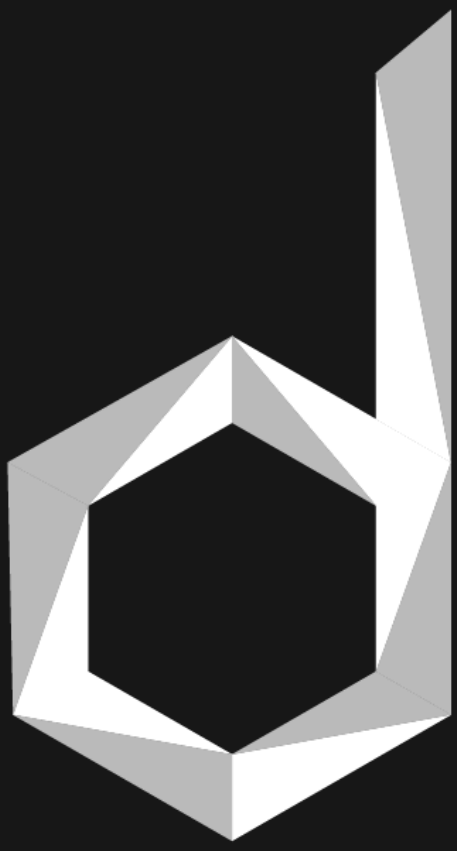
INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

INSTRUMENTACIÓN VIRTUAL

PYTHON 3.9.7, C# & LABVIEW

Graphical User Interface (GUI): PyQt5

Contenido

Teoría – **GUI (Graphical User Interface)**: PyQt5 2

Ejercicios – **GUI**: 4

1.- GUIs Básicas: Window con botón y Window con un Layout que contiene una imagen. 4

Código Python 4

Resultado del Código Python 14

2.- GUI con una Gráfica Estática: Interfaz gráfica que muestra una gráfica creada con Matplotlib. 15

Código Python 15

Resultado del Código Python 20

3.- GUI de Instrumentación Virtual con Arduino: Graficación en tiempo real de tensión analógica. 21

Código IDE Arduino: 22

Código Python 23

Resultado del Código Python 49

Teoría – GUI (Graphical User Interface): PyQt5

Las interfaces gráficas o GUI por sus siglas en inglés son ventanas con elementos gráficos con los que puede interactuar el usuario como botones, áreas de texto, controles de texto, desplegables, listas, controles numéricos, imágenes, gráficas, etc. Esto sirve para realizar cualquier acción que se quiera ejecutar de forma gráfica con un código hecho enteramente con el lenguaje de programación Python, esto no se puede realizar con la forma simple de Python por lo que se debe de importar una librería que permita diseñar las distintas partes que conforman una GUI.

Existen varias librerías que sirven para la creación de interfaces gráficas, la más simple y básica que existe se llama **wxPython**, la cual está basada en la programación orientada a objetos (POO), pero si es que se quiere diseñar una interfaz gráfica con un aspecto más estético y profesional, es más recomendable usar la librería **PyQt5**.

La librería **PyQt5** es una muy utilizada y versátil, su programación es más sencilla y reducida en comparación con la librería **wxPython**, pero su principal desventaja y diferencia es que está disponible bajo dos licencias:

- Una licencia comercial.
- Una licencia GPL de código abierto.

Esto significa que, si se desea desarrollar aplicaciones comerciales con **PyQt5**, se deberá adquirir una licencia comercial. Por otro lado, **wxPython** se distribuye bajo una licencia de código abierto y permite su uso tanto en aplicaciones comerciales como en proyectos de código abierto, esa es la mayor diferencia entre las dos, aunque en su estructura de código también se pueden observar diferencias, descritas a continuación:

La arquitectura de diseño de las GUI creadas con la librería **wxPython** es la siguiente:

- **Método main:** Es un método en el lenguaje de programación Python a través del cual se ejecutan los métodos de todas las clases incluidas en el programa. **Se declara hasta el final del programa.**
- **Frame:** Es una clase perteneciente a la librería **wxPython** que sirve para crear la ventana de la GUI. **Se debe crear después de las clases que crean los contenedores de la interfaz gráfica.**
    - **SplitterWindow:** La mayoría de los widgets se colocan dentro de los Panel para que se puedan acomodar correctamente, pero existe este widget especial que sirve para cuando existen 2 contenedores principales en vez de uno solo y lo que hace es dividir el espacio de la ventana en dos partes ajustables. Con ajustable nos referimos a que existirá una línea de separación entre las dos partes del **Frame** que permitirá hacerlas más grandes o chicas cuando se arrastre dicha línea.
- **Panel:** Es una clase perteneciente a la librería **wxPython** que sirve para crear los contenedores que se encuentran dentro del **Frame** de la GUI y que a su vez contienen los elementos gráficos con los que va a interactuar el usuario llamados **Widgets**. **Se debe crear de forma separada y antes de la clase que crea la ventana de la interfaz gráfica.**
    - **Widgets:** Son los botones, áreas de texto, controles de texto, checkbox, radio buttons, desplegables, listas, controles numéricos, imágenes, gráficas, etc. con los que interactúa el usuario en la GUI.

La arquitectura de diseño de las GUI creadas con la librería **PyQt5** es la siguiente:

- **Método main:** Es un método en el lenguaje de programación Python a través del cual se ejecutan los métodos de todas las clases incluidas en el programa. **Se declara hasta el final del programa.**
- **Window:** Es una clase perteneciente a la librería **PyQt5** que sirve para crear la ventana de la GUI, la gran diferencia de diseño en comparación con la librería **wxPython** es que todos los contenedores de la interfaz gráfica se pueden declarar dentro de la ventana principal, aunque si se quiere se pueden crear clases adicionales que ejecuten ciertas acciones, pero **todos los contenedores serán manejados y declarados dentro de este tipo de clase.**
    - **Layout:** Es un contenedor perteneciente a la librería **PyQt5** que permite almacenar y organizar en una forma específica varios **Widgets** que conforman la interfaz gráfica, es el equivalente al **Panel** de la clase **wxPython**.
    - **Widgets:** Son los botones, áreas de texto, controles de texto, checkbox, radio buttons, desplegables, listas, controles numéricos, imágenes, gráficas, etc. con los que interactúa el usuario en la GUI.

Adicionalmente es importante mencionar que la gran ventaja que proporciona el usar la librería **PyQt5** es que, para mejorar el aspecto estético de sus elementos, se pueden utilizar líneas de código del lenguaje **CSS** y etiquetas **HTML**.

Ejercicios – GUI:

Se realizarán 3 programas donde se diseñarán y explorarán las distintas partes y características de las interfaces gráficas.

1.- GUIs Básicas: Window con botón y Window con un Layout que contiene una imagen.

En el primer programa se crearán 2 GUIs:

1. GUI de un Window con un Layout que contiene un botón de tipo switch, donde el botón cuenta con un ícono extraído de una imagen, este percibe todas las veces que es presionado, al hacerlo cambia su aspecto y lo imprime en consola.
2. GUI de un Window con un Layout que incluye una imagen, para cargar una imagen se deben usar dos widgets distintos, uno que carga la matriz digital que representa la imagen y otro que permite mostrar la imagen en la interfaz gráfica, además se debe escalar la imagen (cambiar su tamaño) para que quepa en el GUI.

Código Python

```
# -*- coding: utf-8 -*-


#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#LIBRERÍAS:
#PyQt5 - QtWidgets: La clase QtWidgets proporciona todos los elementos que conforman las interfaces gráficas
#(GUI) hechas con la librería PyQt5, entre dichas herramientas se incluyen:
# - Widgets: Elementos gráficos básicos de los que se conforma una GUI como lo son botones (QPushButton), cajas
#   de texto (QLineEdit), texto estático (QLabel), listas desplegables (QComboBox), barras de progreso
#   (QProgressBar), casillas de verificación (QCheckBox), imágenes (QLabel), etc.
#   - Diálogos: Los diálogos son elementos gráficos que  facilitan la interacción con el usuario, incluyen
#     widgets como cuadros de diálogo de mensajes (QMessageBox), cuadros de diálogo para seleccionar archivos o
#     directorios (QFileDialog), cuadros de diálogo de entrada de texto (QInputDialog), etc.
# - Layouts: Es un contenedor de PyQt5 que permite almacenar y organizar varios widgets, es el equivalente al
#   Panel de la librería wxPython.
#           - Widget: Es un contenedor que puede almacenar widgets directamente, proporcionando funcionalidades
#             para mostrar, ocultar, establecer posición, tamaño, manejar eventos, etc. de los diferentes
#             botones, checkboxes, áreas de texto, comboboxes, radiobuttons, listboxes, etc.
# - Ventana: Es la ventana principal de la GUI, que a su vez contiene todos los contenedores con los widgets que
#   conforman la GUI, se declara a través de la clase QMainWindow de la librería pyQt5 y es el equivalente al
#   Frame de wxPython.
```

```
from PyQt5 import QtWidgets
#PyQt5 - QtWidgets: Clase que incluye métodos para trabajar con temporizadores, tamaño de elementos, fechas,
#archivos, directorios, señales, hilos, subprocesos, etc.
from PyQt5 import QtCore
#PyQt5 - QtGui: Clase que incluye clases y métodos para trabajar con gráficos, fuentes, colores, imágenes,
#íconos y otros elementos visuales utilizados en una interfaz gráfica (GUI) de PyQt5.
from PyQt5 import QtGui
import sys #sys: Librería que permite interactuar directamente con el sistema operativo y consola del ordenador.

#GUI (Graphical User Interface): Es una ventana con elementos como botones, áreas de texto, desplegables,
#imágenes, etc. que sirven para realizar alguna acción de forma gráfica para el usuario. A continuación, veremos
#como se crean este tipo de elementos en Python utilizando la librería wxPython o PyQt5.
#La librería PyQt5 es una muy utilizada y versátil, su programación es más sencilla y reducida en comparación
#con la librería wxPython, pero su principal desventaja y diferencia es que PyQt5 está disponible bajo dos
#licencias: una licencia comercial y una licencia GPL de código abierto, esto significa que si se desea
#desarrollar aplicaciones comerciales con PyQt5, se deberá adquirir una licencia comercial. Por otro lado,
#wxPython se distribuye bajo una licencia de código abierto y permite su uso tanto en aplicaciones comerciales
#como en proyectos de código abierto.

#MainWindow: La clase hereda de la clase QMainWindow, que a su vez hereda de la clase QtWidgets y ambas
#pertenecen a la librería PyQt5. El elemento representa la ventana del GUI y dentro de ella crea una instancia
#de alguna clase para agregar cualquier elemento, ya sea un Widget directamente o un Canvas (contenedor).
class MainWindow(QtWidgets.QMainWindow):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor.
    def __init__(self):
        #super(Llamada al constructor heredado).__init__(Parámetros que se le asignan): Lo que hace el método
        #super() es llamar al constructor de la clase padre de la clase actual (si es que no se le indica ningún
        #parámetro) o a cualquier clase que se le indique en su parámetro, después la instrucción .__init__()
        #asigna valores default a los parámetros del constructor de la clase padre (si es que no se indica
        #ningún parámetro), aunque de igual manera se pueden asignar parámetros adicionales, cualquier parámetro
        #incluido en el método init, será considerado como adicional. En conclusión, lo que está realizando la
        #línea de código es primero llamar al constructor de la superclase para realizar las tareas de
        #inicialización requeridas antes de indicar parámetros adicionales a la instancia de la clase actual.
        super().__init__()
        #PyQt5.QtWidgets.QMainWindow.setWindowTitle(): Método para colocar un título al Window creado con PyQt5.
        self.setWindowTitle("My App")

        #CREACIÓN DE LOS WIDGETS: Botón
        #Instancia de la librería PyQt5 por medio del constructor de la clase QPushButton que hereda de la clase
        #QtWidgets y sirve para crear un widget de tipo botón, en este se deben indicar como parámetros:
        # - parent: Especifica el objeto padre al que se asociará el botón. Si se proporciona, el botón se
        #   colocará dentro del widget padre.
        # - text: Con este parámetro se indica el texto que aparecerá sobre el botón.
        # - icon: Establece el ícono que se mostrará junto al texto del botón, debe utilizarse un objeto QIcon
```

```
        #   perteneciente a la librería QtGui para que se pueda agregar un ícono.
        # - checkable: Especifica si el botón funciona como un switch (que mantiene su estado) o como un push
        #   button (que no mantiene su estado a menos que se mantenga presionado).
        #       - checkable = True:     Botón tipo switch.
        #       - checkable = False:    Botón tipo push button.
        # - autoDefault: Indica si el botón es el predeterminado del diálogo. Si se establece en True, el botón
        #   responderá automáticamente a la tecla "Enter", sino no lo hará.
        # - flat: Con False se indica que el botón se muestre sin un marco, con True aparece el marco.
        # - menu: Especifica el menú desplegable asociado al botón.
        # - iconSize: Establece el tamaño del ícono del botón, para ello recibe un objeto QSize:
        #       - QtCore.QSize(ancho, alto): Objeto que indica el tamaño del ícono.

        #CREACIÓN DE ÍCONO PARA INCLUIR EN EL BOTÓN:
        #Variable que guarda el directorio y el nombre del archivo creado, se deben reemplazar los guiones\ por /
        #Para leer una imagen o cualquier otro archivo se usa la dirección relativa o absoluta de un directorio:
        # - Dirección relativa: Es una dirección que busca un archivo desde donde se encuentra la carpeta del
        #   archivo python actualmente, esta se debe colocar entre comillas simples o dobles.
        # - Dirección absoluta: Es una dirección que coloca toda la ruta desde el disco duro C o cualquier otro
        #   que se esté usando hasta la ubicación del archivo, la cual se debe colocar entre comillas simples o
        #   dobles.
        #   ..      : Significa que nos debemos salir de la carpeta donde nos encontramos actualmente.
        #   /       : Sirve para introducirnos a alguna carpeta cuyo nombre se coloca después del slash.
        #   .ext    : Se debe colocar siempre el nombre del archivo + su extensión.
        iconPath = "Archivos_Ejercicios_Python/Img/IconLogoDi_cer0.png"
        #PyQt5.QtGui.QIcon(): Constructor de la clase QIcon que hereda de la clase QtGui y perteneciente a la
        #librería PyQt5, usado para crear un objeto que ícono que pueda ser añadido a cualquier widget como lo
        #puede ser un botón, un texto estático, etc. El tamaño de dicha imagen será reducido automáticamente.
        icono = QtGui.QIcon(iconPath)
        #Se declaran como self.nombreObjeto los widgets a los que se les vaya a extraer o introducir datos en el
        #transcurso del funcionamiento de la interfaz gráfica, en este caso se aplica al botón porque este va a
        #cambiar el texto que tiene escrito cuando sea presionado.
        self.button = QtWidgets.QPushButton(text = "\t\t\t\tPresionameeee!", icon = icono, iconSize = QtCore.QSize(90, 90),
checkable = True)
        #PyQt5.QtWidgets.QPushButton.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de
        #métodos, no todos) a los widgets de una interfaz gráfica de usuario (GUI).
        # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
        #   esta no es admitida por PyQt5:
        #   background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 rgb(255,0,0), stop:1 rgb(0,0,255));
        self.button.setStyleSheet("background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,255,255), stop:1
rgb(91,150,242));")

        #widget.setFixedSize(): Método que sirve para indicar el tamaño de un elemento en una interfaz gráfica
        #hecha con la librería PyQt5, para ello recibe un objeto QSize:
        # - QtCore.QSize(ancho, alto): Objeto que indica el tamaño del widget.
        self.button.setFixedSize(QtCore.QSize(250, 150))
```

```
#PyQt5.QtWidgets.QMainWindow.setCentralWidget() = self.setCentralWidget(): Método aplicado al objeto de
#la clase QMainWindow, del que hereda esta clase propia para establecer el widget central de la ventana
#principal, ya que en PyQt5 las ventanas generalmente se dividen en diferentes áreas:
# - Una barra de menú en la parte superior, para ello se debe indicar que objeto es el menu bar.
#       - widget.setMenuBar(widget, QMenuBar)
# - Una barra de herramientas opcional en la parte superior o inferior.
#       - widget.setStatusBar(widget, QStatusBar)
# - Un área central donde se coloca el contenido principal de la ventana.
#       - widget.setCentralWidget(widget)
#El método setCentralWidget() se utiliza para especificar qué widget se debe colocar en el área central
#de la ventana creada con esta clase propia.
self.setCentralWidget(self.button)

#Instancia_Widget.evento_señal.connect(función_que_reacciona_al_evento): Este método se utiliza para
#enlazar un evento a un controlador de eventos, que es una función que se ejecuta cuando ocurra el
#evento, para ello se usa el nombre del widget, seguido del evento de tipo señal que detona el método,
#la palabra reservdada .connect() y entre paréntesis se coloca el nombre de la función que ejecutará
#alguna acción cuando ese evento ocurra. Normalmente las funciones que describen las acciones a
#realizar por los widgets de la GUI se encuentran dentro de esta misma clase, pero fuera de su
#constructor.
# - Tipos de Eventos en Python:
#       - clicked: Señal emitida cuando se hace clic en un elemento, como un botón.
#       - doubleClicked: Señal emitida cuando se hace doble clic en un elemento.
#       - pressed: Señal emitida cuando se presiona un elemento, como un botón.
#       - released: Señal emitida cuando se suelta un elemento, como un botón.
#       - textChanged: Señal emitida cuando el texto de un elemento, como un campo de texto, cambia.
#       - currentIndexChanged: Señal emitida cuando se cambia el índice seleccionado en un elemento,
#         como en un menú desplegable.
#       - activated: Señal emitida cuando se selecciona un elemento, como un elemento de un menú
#         desplegable o una opción de una lista.
#       - keyPressed: Señal emitida cuando se presiona una tecla en el teclado.
#       - keyReleased: Señal emitida cuando se suelta una tecla en el teclado.
#       - mousePressEvent: Señal emitida cuando se presiona un botón del mouse.
#       - mouseReleaseEvent: Señal emitida cuando se suelta un botón del mouse.
#       - mouseMoveEvent: Señal emitida cuando se mueve el mouse.
#       - valueChanged: Señal emitida cuando se selecciona un nuevo elemento en un combo box (lista
#         desplegable).
#       - timeout: Señal emitida cuando transcurre cada intervalo de tiempo especificado en un
#         temporizador.
#Es importante mencionar que en  PyQt5, cuando se conecta una función a un evento, la función conectada
#puede recibir argumentos adicionales proporcionados por la señal emitida. Estos argumentos son
#transmitidos automáticamente por el sistema de señales y slots de PyQt5.
# - Tipos de argumentos retornados al suceder un evento:
#       - *args y **kwargs: Muchas señales en PyQt5 permiten enviar argumentos adicionales a través de
```

```
        #          *args (tupla de argumentos posicionales) y **kwargs (diccionario de argumentos de palabras
        #          clave). Estos parámetros pueden variar según la señal específica y su contexto de uso.
        #       - checked: Algunas señales, como "clicked" en un botón, pueden enviar el estado de alternancia
        #         del widget. Este parámetro indica si el widget está marcado y suele ser de tipo booleano.
        #       - text: En widgets de entrada de texto, como QLineEdit o QTextEdit, las señales pueden enviar el
        #         texto ingresado o modificado como parámetro.
        #       - index: En widgets que tienen índices o selecciones, como QComboBox o QListView, las señales
        #         pueden enviar el índice seleccionado como parámetro.
        #       - position: En widgets que trabajan con eventos de posición, como QMouseEvent, las señales
        #         pueden enviar la posición del cursor o del evento como parámetro.
        #Al presionar el botón se ejecutará el método botonPresionado, declarado dentro de la misma clase.
        self.button.clicked.connect(self.botonPresionado)
        #Al presionar el botón se ejecutará el método the_button_was_toggled, declarado dentro de la misma clase
        self.button.clicked.connect(self.the_button_was_toggled)

    #función botonPresionado(): Método creado dentro de la clase propia llamada MainWindow que recibe como
    #parámetro el evento que lo activa, para posteriormente ejecutar cierta acción.
    #En este caso el evento es activado cuando se presiona el botón y lo que hace es checar que dice el texto
    #que se encuentra sobre el botón, para alternar lo que este dice cuando sea presionado.
    def botonPresionado(self):
        #PyQt5.QtWidgets.QPushButton.text(): Método que devuelve el texto que esté situado sobre un botón.
        if (self.button.text() == "\t\t\t\tPresionameeee!"):
            #PyQt5.QtWidgets.QPushButton.setText(): Método que coloca el texto indicado en el paréntesis sobre
            #el botón al que se esté aplicando el método.
            self.button.setText("\t\t\t\tPresionado!")
        else:
            self.button.setText("\t\t\t\tPresionameeee!")
        #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
        print("Presionado!")

    #función botonPresionado(): Método creado dentro de la clase propia llamada MainWindow que recibe como
    #parámetro el evento que lo activa para posteriormente ejecutar cierta acción, además recibe el parámetro
    #checked que es obtenido a través de la señal clicked, este permite saber si un widget ha sido seleccionado
    #o no.
    #En este caso el evento es activado cuando se cambia el estado del botón y lo que hace es imprimir un
    #mensaje en consola donde se indica con True o False si el botón que funciona como Switch está presionado
    #o no.
    def the_button_was_toggled(self, checked):
        #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
        print("¿El switch se encuentra presionado?:", checked)


#__name__ == __main__: Método main, esta función es super importante ya que sirve para instanciar las clases del
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.
```

```
if(__name__ == "__main__"):
    #Instancia de la librería PyQt5 por medio del constructor de la clase QApplication, que hereda de la clase
    #QtWidgets para crear un objeto que funcione como la base de una GUI.
    # - sys.argv: Se refiere a un vector llamado "argument vector" que puede ser accedido desde la librería sys,
    #   este incluye en su contenido el nombre del archivo que se quiere ejecutar y los argumentos necesarios
    #   que se le deben pasar para que efectúe su ejecución. Se le debe pasar como parámetro al constructor de
    #   la clase QApplication para que se pueda crear la base de la GUI.
    #       - Por ejemplo, si se ejecutara el siguiente comando en consola:
    #           python mi_script.py arg1 arg2 arg3
    #         El contenido de sys.argv sería:
    #           ['mi_script.py', 'arg1', 'arg2', 'arg3']
    app = QtWidgets.QApplication(sys.argv)
    #Instancia de nuestra clase propia llamada MainWindow que fue creada en este mismo programa (window se
    #refiere a la ventana del GUI en PyQt5) e incluye una instancia de la clase GraficaPyQt5 para agregar un
    #widget que crea una gráfica dentro, el constructor vacío lo que hace es indicar que se cree y muestre la
    #ventana.
    window = MainWindow()
    #PyQt5.QtWidgets.QMainWindow.move() = window.move(): Método utilizado para indicar la posición inicial de
    #la ventana dentro de la pantalla del ordenaror, este método recibe como parámetro una tupla que indica la
    #posición:
    # - (x, y): Con este atributo se indica la posición inicial del Frame en pixeles, siendo la posición 0,0 la
    #   esquina superior izquierda, donde las "y" positivas indican que se mueva el botón hacia abajo y las "x"
    #   positivas hacia la derecha.
    window.move(300, 300)
    #PyQt5.QtWidgets.QMainWindow.resize(): Método que permite cambiar el tamaño del widget, contenedor o ventana
    #de una GUI, indicando su ancho y alto en.
    window.resize(500, 500)
    #PyQt5.QtWidgets.QMainWindow.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos,
    #no todos) a los widgets, contenedores o ventanas de una interfaz gráfica de usuario (GUI).
    window.setStyleSheet("background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 rgb(219,160,32), stop:1 rgb(170,0,0));")
    #PyQt5.QtWidgets.QMainWindow.show() = window.show(): Método aplicado al objeto de la clase QMainWindow,
    #del que hereda esta clase propia para mostrar la ventana del GUI.
    window.show()
    #PyQt5.QtWidgets.QApplication.exec_(): Método para que se ejecute en un loop infinito el GUI, logrando así
    #que no se ejecute una vez y luego cierre por sí solo, sino que solo se cierre solamente al dar clic en el
    #tache del window.
    app.exec_()



#VentanaImagen: La clase hereda de la clase QMainWindow, que a su vez hereda de la clase QtWidgets y ambas
#pertenecen a la librería PyQt5. El elemento representa la ventana del GUI y crea una instancia de la clase
#GraficaPyQt5 para agregar dentro de la ventana una gráfica.
```

```
class VentanaImagen(QtWidgets.QMainWindow):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor. A través de este constructor se recibe el ancho y
    #alto de la imagen, que se ajusta al tamaño de la interfaz gráfica.
    def __init__(self, widthImagen, heightImagen):
        #super(Llamada al constructor heredado).__init__(Parámetros que se le asignan): Lo que hace el método
        #super() es llamar al constructor de la clase padre de la clase actual (si es que no se le indica ningún
        #parámetro) o a cualquier clase que se le indique en su parámetro, después la instrucción .__init__()
        #asigna valores default a los parámetros del constructor de la clase padre (si es que no se indica
        #ningún parámetro), aunque de igual manera se pueden asignar parámetros adicionales, cualquier parámetro
        #incluido en el método init, será considerado como adicional. En conclusión, lo que está realizando la
        #línea de código es primero llamar al constructor de la superclase para realizar las tareas de
        #inicialización requeridas antes de indicar parámetros adicionales a la instancia de la clase actual.
        super().__init__()
        #PyQt5.QtWidgets.QMainWindow.setWindowTitle(): Método para colocar un título al Window creado con PyQt5.
        self.setWindowTitle("GUI Imagen y texto estáticos")

        #CREACIÓN DE LOS WIDGETS: Texto Estático, clase QLabel
        #Instancia de la librería PyQt5 por medio del constructor de la clase QLabel que hereda de la clase
        #QtWidgets y sirve para crear un widget que muestre un texto estático o una imagen en una interfaz
        #gráfica, se le deben indicar los siguientes parámetros cuando se usa para crear texto estático:
        # - parent: Especifica el widget padre del QLabel. Si se proporciona, el texto estático se colocará
        #   dentro del widget padre.
        # - text: Permite especificar el texto que se mostrará en el widget. Puede ser una cadena de texto en
        #   formato plano o enriquecido con etiquetas HTML.
        #Para dar estilo al QLabel cuando se utiliza para mostrar texto estático es mejor utilizar etiquetas
        #HTML que contengan un style que les dé estilo por medio de instrucciones CSS, además es importante
        #mencionar que para el style se deben usar comillas simples ('') para que no tenga conlficto con las
        #comillas dobles del parámetro text = "". PyQt5 acepta algunas instrucciones CSS pero no todas.
        text_label = QtWidgets.QLabel(text = "<p style='background-color: black; font-size: 30px; font-family: Consolas,
monospace; color: white;'>di_cer0 Logo</p>")

        #CREACIÓN DE LOS WIDGETS: Imágen, clase QLabel
        #Las clases PyQt5.QtGui.QPixmap y PyQt5.QtWidgets.QLabel están relacionadas y se utilizan en conjunto
        #para trabajar con imágenes en la interfaz gráfica.
        #   PyQt5.QtGui.QPixmap: Representa una imagen en memoria creada con datos en bruto, osea un formato
        #   matricial 3D conformado 3 capas o dimensiones RGB que contienen valores de 0 a 255. Esta clase
        #   proporciona métodos para cargar, manipular y transformar imágenes.
        #
        #   PyQt5.QtWidgets.QLabel: Es una representación de imagen que puede ser utilizada directamente en los
        #   controles y widgets de la interfaz gráfica de PyQt5.
        #Variable que guarda el directorio y el nombre del archivo creado, se reemplazan los guiones\ por / para
        #leer una imagen o cualquier otro archivo se usa la dirección relativa o absoluta de un directorio:
        # - Dirección relativa: Es una dirección que busca un archivo desde donde se encuentra la carpeta del
        #   archivo python actualmente, esta se debe colocar entre comillas simples o dobles.
```

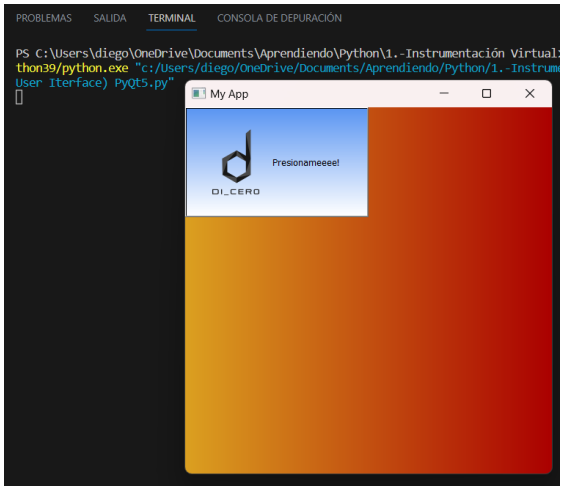
```
# - Dirección absoluta: Es una dirección que coloca toda la ruta desde el disco duro C o cualquier otro
#   que se esté usando hasta la ubicación del archivo, la cual se debe colocar entre comillas simples o
#   dobles.
#   ..      : Significa que nos debemos salir de la carpeta donde nos encontramos actualmente.
#   /       : Sirve para introducirnos a alguna carpeta cuyo nombre se coloca después del slash.
#   .ext    : Se debe colocar siempre el nombre del archivo + su extensión.
path = "Archivos_Ejercicios_Python/Img/IconLogoDi_cer0.png"
#Cargar una Imagen: Se crea una instancia de la librería PyQt5 por medio del constructor de la clase
#QPixmap para cargar una imagen en memoria con datos en bruto.
# - filepath: En este atributo se indica el nombre de archivo junto con la ruta completa de donde se
#   encuentra la imagen que se va a cargar, la cual debe estar indicada en forma de string, con este
#   atributo no se sigue la nomenclatura de poner el nombre del atributo seguido de su valor
#   (filepath = valor), solamente se pone el path entre comillas.
# - format: El formato de imagen deseado. Puede ser una cadena como "PNG", "JPEG", "BMP", etc. Si no se
#   especifica, se intentará detectar automáticamente el formato.
rawImage = QtGui.QPixmap(path, format = "PNG")
#PyQt5.QtGui.QPixmap.scaled(): Método utilizado para redimensionar una imagen conformada por datos
#brutos, representada por un objeto QPixmap, se realiza a través de la siguiente sintaxis:
#scaledImage = originalImage.scaled(width, height, aspectRatioMode)
# - originalImage: Es el objeto QPixmap original que se desea redimensionar.
# - width: Es el nuevo ancho de la imagen redimensionada.
# - height: Es la nueva altura de la imagen redimensionada.
# - aspectRatioMode: Es una constante de la clase QtCore.Qt.AspectRatioMode que especifica cómo se debe
#   ajustar la imagen para mantener la proporción.
#           - QtCore.Qt.AspectRatioMode.IgnoreAspectRatio: La imagen se ajusta exactamente al tamaño
#             especificado, sin mantener la proporción original. Esto puede dar lugar a una distorsión
#             de la imagen.
#           - QtCore.Qt.AspectRatioMode.KeepAspectRatio: La imagen se redimensiona manteniendo la
#             proporción original y ajustándose dentro del tamaño especificado para que quepa
#             completamente sin recortarla. Esto puede generar márgenes en blanco si la relación de
#             aspecto difiere del tamaño especificado.
#           - QtCore.Qt.AspectRatioMode.KeepAspectRatioByExpanding: La imagen se redimensiona
#             manteniendo la proporción original y ocupando todo el tamaño especificado. Esto puede
#             provocar que parte de la imagen quede fuera de los límites del tamaño especificado.
#En todos los parámetos mencionados para este método, no se indica explícitamente su nombre, solo se
#asigna un valor y el método identifica cuál es por su orden.
#Todas las operaciones donde se afecta a una imagen se deben realizar a la imagen representada en
#datos brutos, para luego asignarse al widget de la GUI.
scaledImage = rawImage.scaled(widthImagen, heightImagen, QtCore.Qt.AspectRatioMode.KeepAspectRatio)
#Mostrar la imagen en un widget: Para mostrar la imagen de tipo Bitmap en un widget gráfico de se usa
#una instancia de la clase PyQt5.QtWidgets.QLabel, que es parte de la biblioteca PyQt5 y se utiliza para
#crear un widget que muestre un texto estático o una imagen en una interfaz gráfica, se le deben indicar
#los siguientes parámetros cuando se usa para crear una imagen estática:
# - parent: Este parámetro es opcional y especifica el widget padre del QLabel. Indica a qué widget
#   pertenece el QLabel.
```

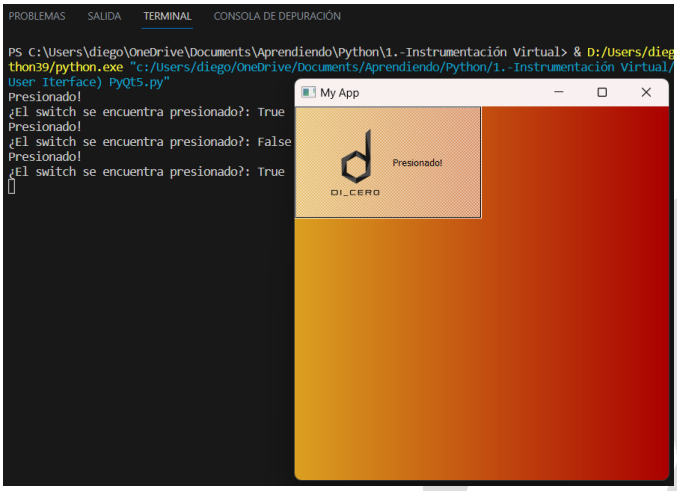
```
# - pixmap (QPixmap): La imagen que se mostrará en la etiqueta.
#Para dar estilo al QLabel cuando se utiliza para mostrar texto estático es mejor utilizar etiquetas
#HTML que contengan un style que les dé estilo por medio de instrucciones CSS, además es importante
#mencionar que para el style se deben usar comillas simples ('') para que no tenga conlficto con las
#comillas dobles del parámetro text = "". PyQt5 acepta algunas instrucciones CSS pero no todas.
image_label = QtWidgets.QLabel(pixmap = scaledImage)    #Imagen redimensionada.

#ALINEACIÓN DEL CONTENIDO EN LOS LABELS (IMÁGENES O TEXTO ESTÁTICOS):
#PyQt5.QtWidgets.QLabel.setAlignment(): Método utilizado para establecer la alineación del contenido
#dentro de un objeto QLabel. Permite controlar cómo se posiciona y alinea el texto o la imagen dentro
#del espacio disponible en la etiqueta. Los valores que recibe el método son:
# - QtCore.Qt.AlignLeft: Alinea el contenido a la izquierda.
# - QtCore.Qt.AlignRight: Alinea el contenido a la derecha.
# - QtCore.Qt.AlignHCenter: Alinea el contenido en el centro horizontal.
# - QtCore.Qt.AlignJustify: Justifica el contenido.
# - QtCore.Qt.AlignJustify: Justifica el contenido.
# - QtCore.Qt.AlignBottom: Alinea el contenido en la parte inferior.
# - QtCore.Qt.AlignVCenter: Alinea el contenido en el centro vertical.
# - QtCore.Qt.AlignCenter: Alinea el contenido en el centro tanto horizontal como vertical.
#Estos valores se pueden combinar utilizando el operador OR (|) para lograr una alineación específica.
text_label.setAlignment(QtCore.Qt.AlignCenter)          #QLabel texto colocado en medio.
image_label.setAlignment(QtCore.Qt.AlignCenter)         #QLabel imagen colocado en medio.

#CONTENEDORES DE ELEMENTOS: La biblioteca PyQt5 ofrece varios tipos de contenedores que se pueden
#utilizar para organizar los widgets en una interfaz gráfica. Los más comunes son:
# - QVBoxLayout: Organiza los widgets en una disposición vertical, uno debajo del otro.
# - QHBoxLayout: Organiza los widgets en una disposición horizontal, uno al lado del otro.
# - QGridLayout: Organiza los widgets en una cuadrícula bidimensional de filas y columnas.
# - QFormLayout: Diseñado específicamente para crear formularios, donde los widgets se colocan en pares
#   de etiqueta y campo de entrada.
# - QStackedLayout: Permite apilar varios widgets uno encima del otro y mostrar uno a la vez.
# - QTabWidget: Permite crear pestañas donde se pueden colocar diferentes conjuntos de widgets en cada
#   pestaña.
# - QScrollArea: Proporciona una vista desplazable para un contenido que puede ser mayor que el área
#   visible.
# - QGroupBox: Crea un grupo que puede contener y organizar otros widgets.
# - QSplitter: Permite dividir el área de visualización en secciones redimensionables que contienen
#   widgets diferentes.
# - QWidget: Proporciona una ventana o área rectangular en la que se pueden colocar otros widgets para
#   crear una interfaz gráfica, un QWidget puede contener otros widgets o contenedores dentro.
#Instancia de la clase QWidget, que hereda de la clase QtWidgets y pertenece a la librería PyQt5, dicho
#objeto funciona como un contenedor que puede almacenar widgets directamente, proporcionando
#funcionalidades para mostrar, ocultar, establecer posición, tamaño, manejar eventos, etc. de los
#diferentes botones, checkboxes, áreas de texto, comboboxes, radiobuttons, listboxes, ventanas de
#diálogo (ventana que muestra el explorador de archivos), etc.
```

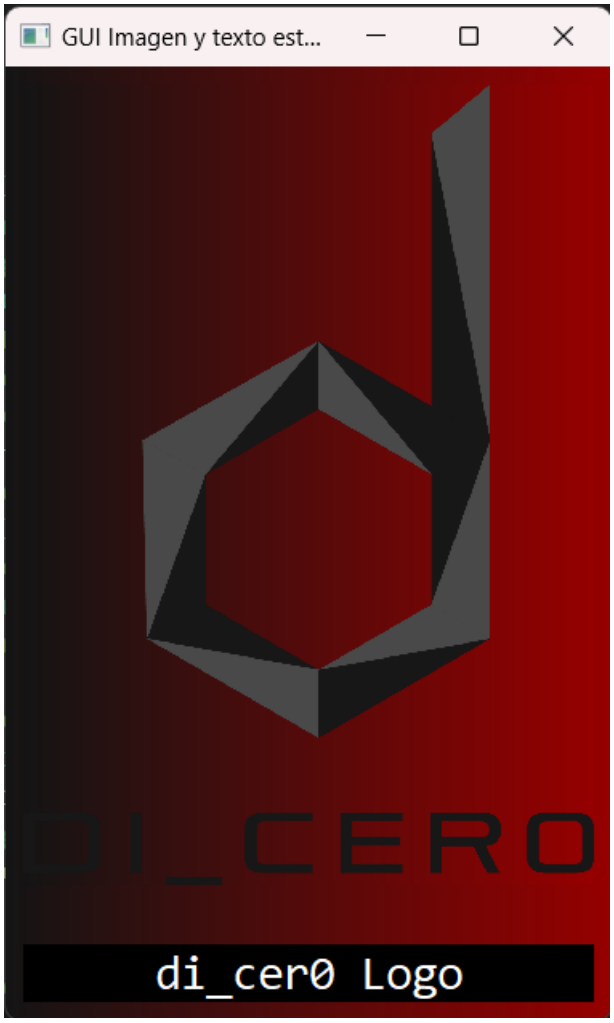
```
        central_widget = QtWidgets.QWidget()                    #Contenedor de una imagen y un texto estáticos.
        #PyQt5.QtWidgets.QMainWindow.setCentralWidget() = self.setCentralWidget(): Método aplicado al objeto de
        #la clase QMainWindow, del que hereda esta clase propia para establecer el widget central de la ventana
        #principal, ya que en PyQt5 las ventanas generalmente se dividen en diferentes áreas:
        # - Una barra de menú en la parte superior, para ello se debe indicar que objeto es el menu bar.
        #       - widget.setMenuBar(widget, QMenuBar)
        # - Una barra de herramientas opcional en la parte superior o inferior.
        #       - widget.setStatusBar(widget, QStatusBar)
        # - Un área central donde se coloca el contenido principal de la ventana.
        #       - widget.setCentralWidget(widget)
        #El método setCentralWidget() se utiliza para especificar qué widget se debe colocar en el área central
        #de la ventana creada con esta clase propia.
        self.setCentralWidget(central_widget)                   #Contenedor colocado en el área central.

        #Objeto de la clase QVBoxLayout, el cual se utiliza para organizar los widgets en una disposición
        #vertical, proporcionando una forma conveniente de colocar los widgets uno debajo del otro en una
        #ventana o en otro contenedor.
        # - parent: Si el constructor de esta clase recibe como parámetro un objeto QWidget, ese será el
        #   contenedor principal del objeto QVBoxLayout que organiza sus elementos verticalmente.
        # - Si no recibe ningún parámetro, este es un contenedor vacío sin widget principal que aceptará
        #   varios elementos o contenedores y los irá colocando verticalmente uno después del otro.
        layout = QtWidgets.QVBoxLayout(central_widget) #Alineación vertical de contenedores.
        #PyQt5.QtWidgets.QVBoxLayout.addWidget(): Método usado para añadir un widget de manera secuencial en la
        #columna de un diseño vertical, como lo puede ser un botón, lista desplegable, texto, imagen, etc. Se
        #indica el órden en el que se colocarán los elementos dependiendo de cual fue añadido primero y cual
        #después.
        layout.addWidget(image_label)                           #Al contenedor primero se coloca la imagen.
        #PyQt5.QtWidgets.QVBoxLayout.addWidget(): Método que permite agregar un espacio vacío indicado en
        #pixeles entre los widgets de un contenedor QVBoxLayout, este se debe colocar exactamente entre los
        #elementos que se desea separar, antes de siquiera agregarlo al contenedor con el método .addWidget()
        layout.addSpacing(30)                                   #Luego un espacio de separación de 30px.
        layout.addWidget(text_label)                            #Y finalmente se coloca el texto.


#__name__ == __main__: Método main, esta función es super importante ya que sirve para instanciar las clases del
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.
if(__name__ == "__main__"):
    #Instancia de la librería PyQt5 por medio del constructor de la clase QApplication, que hereda de la clase
    #QtWidgets para crear un objeto que funcione como la base de una GUI.
    # - sys.argv: Se refiere a un vector llamado "argument vector" que puede ser accedido desde la librería sys,
    #   este incluye en su contenido el nombre del archivo que se quiere ejecutar y los argumentos necesarios
    #   que se le deben pasar para que efectúe su ejecución. Se le debe pasar como parámetro al constructor de
    #   la clase QApplication para que se pueda crear la base de la GUI.
    #       - Por ejemplo, si se ejecutara el siguiente comando en consola:
```

```
    #           python mi_script.py arg1 arg2 arg3
    #         El contenido de sys.argv sería:
    #           ['mi_script.py', 'arg1', 'arg2', 'arg3']
    appImg = QtWidgets.QApplication(sys.argv)
    #Instancia de nuestra clase propia llamada MainWindow que fue creada en este mismo programa (window se
    #refiere a la ventana del GUI en PyQt5) e incluye una instancia de la clase GraficaPyQt5 para agregar un
    #widget que crea una gráfica dentro, el constructor vacío lo que hace es indicar que se cree y muestre la
    #ventana. Además en este caso se le pasa como parámetro el ancho y alto de la ventana e imagen al
    #constructor de la clase.
    ancho = 500                                             #Ancho de la ventana e imagen de 500px.
    alto = 500                                              #Alto de la ventana e imagen de 500px.
    #Debido a que la imagen se redimensionó manteniendo su proporción original y ajustándose dentro del tamaño
    #especificado para que quepa completamente sin recortarla, solamente hará caso al ancho de la ventana, no a
    #su altura, ya que sino esta se vería deformada.
    windowImg = VentanaImagen(ancho, alto)
    #PyQt5.QtWidgets.QMainWindow.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos,
    #no todos) a los widgets, contenedores o ventanas de una interfaz gráfica de usuario (GUI).
    windowImg.setStyleSheet("background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 rgb(20,20,20), stop:1
rgb(150,0,0));")
    #PyQt5.QtWidgets.QMainWindow.show() = window.show(): Método aplicado al objeto de la clase QMainWindow,
    #del que hereda esta clase propia para mostrar la ventana del GUI.
    windowImg.show()
    #PyQt5.QtWidgets.QApplication.exec_(): Método para que se ejecute en un loop infinito el GUI, logrando así
    #que no se ejecute una vez y luego cierre por sí solo, sino que solo se cierre solamente al dar clic en el
    #tache del window.
    appImg.exec_()
```

Resultado del Código Python







2.- GUI con una Gráfica Estática: Interfaz gráfica que muestra una gráfica creada con Matplotlib.

En este programa se creará una sola GUI que muestre una gráfica estática creada con la librería Matplotlib.

Código Python

```
# -*- coding: utf-8 -*-


#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.
```

```
#LIBRERÍAS:
#PyQt5 - QtWidgets: La clase QtWidgets proporciona todos los elementos que conforman las interfaces gráficas
#(GUI) hechas con la librería PyQt5, entre dichas herramientas se incluyen:
# - Widgets: Elementos gráficos básicos de los que se conforma una GUI como lo son botones (QPushButton), cajas
#   de texto (QLineEdit), texto estático (QLabel), listas desplegables (QComboBox), barras de progreso
#   (QProgressBar), casillas de verificación (QCheckBox), imágenes (QLabel), etc.
#   - Diálogos: Los diálogos son elementos gráficos que  facilitan la interacción con el usuario, incluyen
#     widgets como cuadros de diálogo de mensajes (QMessageBox), cuadros de diálogo para seleccionar archivos o
#     directorios (QFileDialog), cuadros de diálogo de entrada de texto (QInputDialog), etc.
# - Layouts: Es un contenedor de PyQt5 que permite almacenar y organizar varios widgets, es el equivalente al
#   Panel de la librería wxPython.
#           - Widget: Es un contenedor que puede almacenar widgets directamente, proporcionando funcionalidades
#             para mostrar, ocultar, establecer posición, tamaño, manejar eventos, etc. de los diferentes
#             botones, checkboxes, áreas de texto, comboboxes, radiobuttons, listboxes, etc.
# - Ventana: Es la ventana principal de la GUI, que a su vez contiene todos los contenedores con los widgets que
#   conforman la GUI, se declara a través de la clase QMainWindow de la librería pyQt5 y es el equivalente al
#   Frame de wxPython.
from PyQt5 import QtWidgets
#matplotlib - Figure: La clase Figure es la base para crear y organizar los elementos gráficos en Matplotlib,
#que es una librería de graficación matemática.
from matplotlib.figure import Figure
#matplotlib - FigureCanvasQTAgg: La clase FigureCanvasQTAgg proporcionada por la biblioteca Matplotlib en el
#módulo matplotlib.backends.backend_qt5agg se utiliza para mostrar y manejar gráficos generados por Matplotlib
#dentro de una ventana o contenedor de pyQt5. En este caso se utiliza para mostrar una gráfica que podría
#actualizar sus datos en tiempo real o no, en este caso se mantendrá estática.
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
import sys #sys: Librería que permite interactuar directamente con el sistema operativo y consola del ordenador.

#GUI (Graphical User Interface): Es una ventana con elementos como botones, áreas de texto, desplegables,
#imágenes, etc. que sirven para realizar alguna acción de forma gráfica para el usuario. A continuación, veremos
#como se crean este tipo de elementos en Python utilizando la librería wxPython o PyQt5.
#La librería PyQt5 es una muy utilizada y versátil, su programación es mas sencilla y reducida en comparación
#con la librería wxPython, pero su principal desventaja y diferencia es que PyQt5 está disponible bajo dos
#licencias: una licencia comercial y una licencia GPL de código abierto, esto significa que si se desea
#desarrollar aplicaciones comerciales con PyQt5, se deberá adquirir una licencia comercial. Por otro lado,
#wxPython se distribuye bajo una licencia de código abierto y permite su uso tanto en aplicaciones comerciales
#como en proyectos de código abierto.

#GraficaPyQt5: Esta clase propia hereda de la clase FigureCanvasQTAgg, que pertenece a la librería PyQt5 y
#permite mostrar gráficos generados por Matplotlib en un Widget de PyQt5, permitiendo manejar eventos de
#interacción del usuario, como hacer zoom, seleccionar puntos en el gráfico, etc.
class GraficaPyQt5(FigureCanvasQTAgg):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor. Los parámetros que puede recibir el constructor de
```

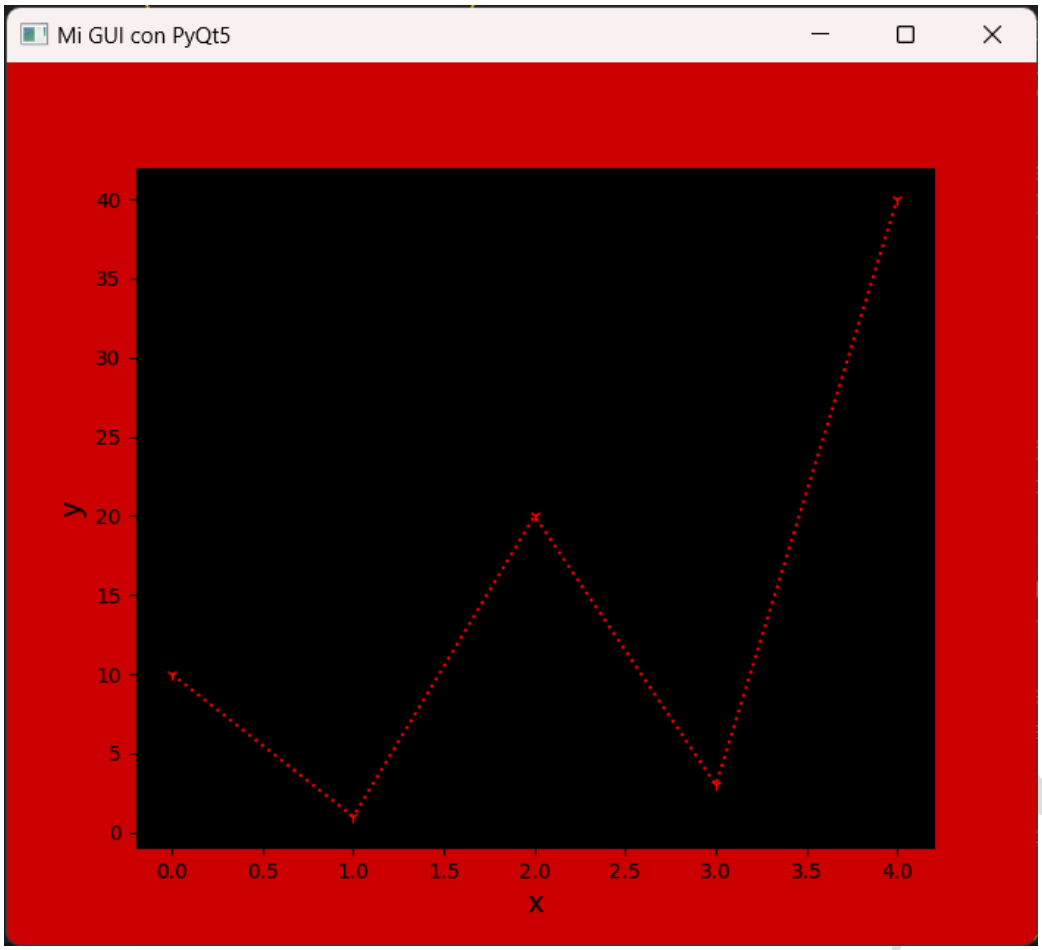
```
#la clase son los siguientes:
# - figure: La instancia de la clase Figure que se va a asociar con el lienzo. Por defecto, se establece en
#   None. Este parámetro es el primero y no usa la sintaxis: figure = valor, solamente se pone.
# - parent: El widget padre al que se asocia el lienzo. Por defecto, se establece en None.
# - width: El ancho del lienzo en píxeles o pulgadas. En este caso solo se indica un valor inicial pero el
#   ancho real será asignado cuando se cree una instancia de la clase.
# - height: La altura del lienzo en píxeles o pulgadas. En este caso solo se indica un valor inicial pero el
#   ancho real será asignado cuando se cree una instancia de la clase.
# - dpi: La resolución del lienzo en puntos por pulgada. Por defecto, se establece en 80 y puede afectar al
#   tamaño de la gráfica.
# - bgcolor: Indica el color de fondo del lienzo. Puede ser especificado a través de las siguientes formas:
#       - Nombre color: Por medio de un string se pueden declarar valores que reconoce Matplotlib como por
#         ejemplo, white', 'red', 'blue', etc.
#       - Tupla RGB: Por medio de una tupla se especifican los tonos de color RGB (Rojo, Verde Azul) con
#         valores que van de 0 a 1, como por ejemplo, (1, 1, 1) para blanco.
def __init__(self, parent = None, width = None, height = None, dpi = 120, bgcolor = (0.8, 0, 0)):
    #Figure(): Constructor de la clase Figure perteneciente a la librería Matplotlib, usado para crear un
    #lienzo en el que se puedan dibujar gráficos, actúando como un contenedor para los subgráficos.
    #A este objeto se le pueden asignar los parámetros indicados en el constructor de la clase
    #FigureCanvasQTAgg.
    fig = Figure(figsize = (width, height), dpi = dpi)
    #matplotlib.Figure().set_facecolor(): Método que sirve para establecer el color de fondo de una gráfica.
    #A este objeto se le puede asignar el parámetros bgcolor indicado en el constructor de la clase
    #FigureCanvasQTAgg.
    fig.set_facecolor(bgcolor)
    #matplotlib.Figure().add_subplot(): Método aplicado a un objeto de la clase Figure, perteneciente a la
    #librería Matplotlib, lo que hace es agregar un subgráfico al lienzo vacío, los números de su parámetro
    #lo que indican es:
    # - Primer Número:  Indica el número de filas de las subgráficas.
    # - Segundo Número: Indica el número de columnas de las subgráficas.
    # - Tercer Número:  Indica el índice de la subgráfica que se está creando en específico.
    #El parámetro 111 crea una sola gráfica de una columna con un solo espacio para mostrar una gráfica.
    #Se declaran como self.nombreObjeto los widgets a los que se les vaya a extraer o introducir datos en el
    #transcurso del funcionamiento de la interfaz gráfica.
    self.axes = fig.add_subplot(111)
    #super(Llamada al constructor heredado).__init__(Parámetros que se le asignan): Lo que hace el método
    #super() es llamar al constructor de la clase padre de la clase actual (si es que no se le indica ningún
    #parámetro) o a cualquier clase que se le indique en su parámetro, después la instrucción .__init__()
    #asigna valores default a los parámetros del constructor de la clase padre (si es que no se indica
    #ningún parámetro), aunque de igual manera se pueden asignar parámetros adicionales, cualquier parámetro
    #incluido en el método init, será considerado como adicional. En conclusión, lo que está realizando la
    #línea de código es primero llamar al constructor de la superclase para realizar las tareas de
    #inicialización requeridas antes de indicar parámetros adicionales a la instancia de la clase actual.
    super(GraficaPyQt5, self).__init__(fig) #Asigna el objeto fig al constructor de la clase GraficaPyQt5.
```

```
#MainWindow: La clase hereda de la clase QMainWindow, que a su vez hereda de la clase QtWidgets y ambas
#pertenecen a la librería PyQt5. El elemento representa la ventana del GUI y crea una instancia de la clase
#GraficaPyQt5 para agregar dentro de la ventana una gráfica.
class MainWindow(QtWidgets.QMainWindow):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor.
    def __init__(self):
        #super(Llamada al constructor heredado).__init__(Parámetros que se le asignan): Lo que hace el método
        #super() es llamar al constructor de la clase padre de la clase actual (si es que no se le indica ningún
        #parámetro) o a cualquier clase que se le indique en su parámetro, después la instrucción .__init__()
        #asigna valores default a los parámetros del constructor de la clase padre (si es que no se indica
        #ningún parámetro), aunque de igual manera se pueden asignar parámetros adicionales, cualquier parámetro
        #incluido en el método init, será considerado como adicional. En conclusión, lo que está realizando la
        #línea de código es primero llamar al constructor de la superclase para realizar las tareas de
        #inicialización requeridas antes de indicar parámetros adicionales a la instancia de la clase actual.
        super(MainWindow, self).__init__() #Asigna parámetros estándar al constructor de la clase MainWindow.
        #PyQt5.QtWidgets.QMainWindow.setWindowTitle(): Método para colocar un título al Window creado con PyQt5.
        self.setWindowTitle("Mi GUI con PyQt5")

        #Instancia de la clase GraficaPyQt5 para agregar la gráfica al Window, se le debe pasar como parámetro
        #al constructor de la clase GraficaPyQt5, los parámetros que se quiera editar para que no se ejecuten
        #los que vienen declarados por defecto, los que no se editen, se ejecutarán con el valor que fueron
        #declarados arriba en el constructor.
        sc = GraficaPyQt5(self, width = 7, height = 6, dpi = 100)

        #matplotlib.Figure.add_subplot().plot(): Método usado para gráficar, indicando como primer parámetro su
        #eje horizontal, luego su eje vertical y finalmente el estilo de la gráfica:
        # - Colores:             C1: color naranja, r: color rojo, b: color azul, g: verde, c: cyan, m: morado,
        #   y: amarillo, k: negro, w: blanco.
        # - Tipo de marcadores:  o: círculos, +: símbolos de más, .; puntos, v: Triángulo hacia abajo, h:
        #   Hexágono, etc.
        # - Tipo de Líneas:      -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto,
        #   'or': Nada.
        #Los marcadores se pueden obtener del siguiente link: https://matplotlib.org/stable/api/markers_api.html
        sc.axes.plot([0, 1, 2, 3, 4], [10, 1, 20, 3, 40], 'r1:') #'r1:' r: color rojo, 1: tri_down, :: línea punteada
        #matplotlib.Figure().add_subplot().set_xlabel(): Método para indicar el texto que aparece en el eje
        #horizontal de la gráfica, recibe los siguientes parámetros:
        # - xlabel: Especifica el texto que se mostrará en el eje x.
        # - fontname: Indica el estilo de la fuente:
        #       - Nombres de tipos de letra estándar: "Arial", "Times New Roman", "Helvetica", "Courier",
        #         "Monospaced", "Consolas", etc. Estos nombres deben ser compatibles con los tipos de letra
        #         instalados en el sistema operativo.
```

```
        #       - Nombres de tipos de letra genéricos: "serif", "sans-serif", "monospace", etc.
        #       - Rutas de archivo: Si se tiene un archivo de tipo de letra personalizado, se puede especificar
        #         la ruta del archivo como el valor de fontname.
        # - fontsize: Indica el tamaño de la fuente.
        # - labelpad: Especifica el espaciado entre la etiqueta del eje x y el eje en sí.
        sc.axes.set_xlabel(xlabel = "x", fontname = "Consolas", fontsize = 15)
        #matplotlib.Figure.add_subplot().set_ylabel(): Método para indicar el texto que aparece en el eje y.
        sc.axes.set_ylabel(ylabel = "y", fontname = "Consolas", fontsize = 15)
        #matplotlib.Figure.add_subplot().set_facecolor(): Método para indicar el color de fondo de la gráfica,
        #el cual puede ser indicado por los mismos colores previamente mencionados en el método plot() o se
        #pueden usar los siguientes con el código xkcd:
        # - Colores: xkcd:aqua, xkcd:aquamarine, xkcd:azure, xkcd:beige, etc. Los colores se obtienen de:
        #https://matplotlib.org/stable/tutorials/colors/colors.html
        sc.axes.set_facecolor('xkcd:black')

        #PyQt5.QtWidgets.QMainWindow.setCentralWidget() = self.setCentralWidget(): Método aplicado al objeto de
        #la clase QMainWindow, del que hereda esta clase propia para establecer el widget central de la ventana
        #principal, ya que en PyQt5 las ventanas generalmente se dividen en diferentes áreas:
        # - Una barra de menú en la parte superior, para ello se debe indicar que objeto es el menu bar.
        #       - widget.setMenuBar(widget, QMenuBar)
        # - Una barra de herramientas opcional en la parte superior o inferior.
        #       - widget.setStatusBar(widget, QStatusBar)
        # - Un área central donde se coloca el contenido principal de la ventana.
        #       - widget.setCentralWidget(widget)
        #El método setCentralWidget() se utiliza para especificar qué widget se debe colocar en el área central
        #de la ventana creada con esta clase propia.
        self.setCentralWidget(sc)


#__name__ == __main__: Método main, esta función es super importante ya que sirve para instanciar las clases del
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.
if(__name__ == "__main__"):
    #Instancia de la librería PyQt5 por medio del constructor de la clase QApplication, que hereda de la clase
    #QtWidgets para crear un objeto que funcione como la base de una GUI.
    # - sys.argv: Se refiere a un vector llamado "argument vector" que puede ser accedido desde la librería sys,
    #   este incluye en su contenido el nombre del archivo que se quiere ejecutar y los argumentos necesarios
    #   que se le deben pasar para que efectúe su ejecución. Se le debe pasar como parámetro al constructor de
    #   la clase QApplication para que se pueda crear la base de la GUI.
    #       - Por ejemplo, si se ejecutara el siguiente comando en consola:
    #           python mi_script.py arg1 arg2 arg3
    #         El contenido de sys.argv sería:
    #           ['mi_script.py', 'arg1', 'arg2', 'arg3']
    app = QtWidgets.QApplication(sys.argv)
    #Instancia de nuestra clase propia llamada MainWindow que fue creada en este mismo programa (window se
```

```
#refiere a la ventana del GUI en PyQt5) e incluye una instancia de la clase GraficaPyQt5 para agregar un
#widget que crea una gráfica dentro, el constructor vacío lo que hace es indicar que se cree y muestre la
#ventana.
window = MainWindow()
#PyQt5.QtWidgets.QMainWindow.move() = window.move(): Método utilizado para indicar la posición inicial de
#la ventana dentro de la pantalla del ordenaror, este método recibe como parámetro una tupla que indica la
#posición:
# - (x, y): Con este atributo se indica la posición inicial del Frame en pixeles, siendo la posición 0,0 la
#   esquina superior izquierda, donde las "y" positivas indican que se mueva el botón hacia abajo y las "x"
#   positivas hacia la derecha.
window.move(100, 100)
#PyQt5.QtWidgets.QMainWindow.show() = window.show(): Método aplicado al objeto de la clase QMainWindow,
#del que hereda esta clase propia para mostrar la ventana del GUI.
window.show()
#PyQt5.QtWidgets.QApplication.exec_(): Método para que se ejecute en un loop infinito el GUI, logrando así
#que no se ejecute una vez y luego cierre por sí solo, sino que solo se cierre solamente al dar clic en el
#tache del window.
app.exec_()
```

Resultado del Código Python



3.- GUI de Instrumentación Virtual con Arduino: Graficación en tiempo real de tensión analógica.

Para que funcione la GUI que capta y grafica en tiempo real los datos de tensión que recibe del pin analógico A0 de la placa de desarrollo Arduino y finalmente guarde esos datos recopilados en un archivo de Excel se deben seguir los siguientes pasos:

Pseudocódigo:

1. Ejecutar un código en el IDE de Arduino que indique que el pin A0 es de lectura analógica y el pin 13 es de salida digital, para que de esa forma se recaben los datos de tensión que vayan de 0 a 5V en el pin A0 y al mismo tiempo se haga parpadear un led en el pin 13, además de indicar cual es el puerto de conexión serial entre la placa de desarrollo y la computadora.
2. Reconocer el sistema operativo de la computadora para que de esta manera se puedan exponer en un widget llamado Combo Box perteneciente a la librería **PyQt5** que permite enlistar, mostrar y elegir de entre varios elementos, el puerto con el que nos queremos conectar al Arduino.
    a. Si no hay ningún puerto seleccionado y se quiere iniciar la recopilación de datos, deberá aparecer una ventana emergente que indique que no hay ningún puerto seleccionado.
3. **Por medio de un botón llamado START, inicializar la instrumentación virtual de la placa Arduino.**
4. Iniciar la comunicación serial con el puerto de conexión previamente utilizado en el IDE del Arduino.
    a. Si ocurre algún error cuando nos queramos conectar de forma serial con el puerto seleccionado, deberá aparecer una ventana emergente que indique que ha ocurrido un error al querer inicializar la comunicación serial entre el ordenador y la placa de desarrollo Arduino UNO.
5. Indicar el número de datos a recabar del Arduino a través de un widget llamado Spin Box perteneciente a la librería **PyQt5**.
6. Leer los datos del pin analógico A0 del Arduino y realizar su conversión de número digital a valor de tensión, esto se realiza tomando en cuenta el rango de valores de tensión (que va de 0 a 5V) y el rango de valores digitales que se conforma de 10 bits, cuando este número binario se convierte a decimal se considera que va de 0 a $2^{10} - 1 = 1023$, la conversión entonces se realiza a través de la siguiente operación:

$$Tensión = Tensión_{Binaria} * \frac{Tensión_{Max}}{Resolución_{ADC}} = Tensión_{Binaria} * \frac{5\ [V]}{2^{10} - 1} = Tensión_{Binaria} * \frac{5\ [V]}{1023}$$

7. Almacenar los datos de tensión y tiempo en una lista, tupla o diccionario.
8. Graficar el vector de datos recabados de tiempo vs. tensión.
9. Actualizar dinámicamente la gráfica para que se actualicen sus valores y se muestren en tiempo real.
    a. Esto se realiza utilizando la clase **FigureCanvasQTAgg** perteneciente a la librería **PyQt5**.
10. Imprimir en consola todos los valores de tensión recabados de los n valores indicados en el Spin Box.
    a. Debido a un concepto llamado señal, que es parecido a los eventos de **wxPython**, cuando se da clic en el Spin Box que indica el número de muestras a recopilar, este se actualiza durante la ejecución del programa, permitiendo cambiando su valor.

11. **Por medio de un botón llamado STOP, detener en cualquier momento la instrumentación virtual de la placa Arduino.**
12. **Por medio de un botón llamado SAVE, abrir el explorador de archivos para poder guardar los datos recabados en un archivo de Excel.**
    a. El máximo de valores que se pueden recopilar son 32,000.

Código IDE Arduino:

Es importante mencionar que los archivos de Arduino a fuerza deben encontrarse dentro de una carpeta que tenga el mismo nombre que el nombre del archivo con extensión .ino que almacena el programa escrito en lenguaje Arduino, este nombre tanto de la carpeta no puede contener espacios.

```
//PROGRAMA PARA RECOPILAR DATOS Y MANDARLOS A PYTHON PARA QUE LOS GRAFIQUE EN TIEMPO REAL
int led = 13;           //Puerto de salida donde hay un led integrado en el Arduino
int voltage = A0;       //Declaración del puerto de entrada analógico que se quiere leer
int n = 0;              //variable que lleva la cuenta de los ciclos de parpadeo del LED.

//CONFIGURACIÓN DE LOS PINES Y COMUNICACIÓN SERIAL
void setup() {
  /*En esta parte del código Arduino se indican los puertos de salida, de entrada y la velocidad
  de la comunicación serial*/
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
      - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
      - segundo parámetro: Usa la insctrucción OUTPUT para indicar que el pin es una salida o
        INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable*/
  pinMode(led, OUTPUT); //El pin 13 es una salida digital.
  /*Serial.begin(baudRate): Este método inicializa la comunicación serial entre la placa Arduino
  y la computadora, además de que configura su velocidad de transmisión dada en unidad de baudios
  (bit trasmitido por segundo) que recibe como su único parámetro:
      - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es
        compatible con la mayoría de los dispositivos y programas.
      - Sin embargo, si se necesita una transferencia de datos más rápida y el hardware/software
        lo admiten, se puede optar por velocidades más altas como 115200 o 57600 baudios.
  Es importante asegurarse de que la velocidad de transmisión especificada coincida con la
  velocidad de comunicación del otro dispositivo al que se conecta el Arduino. Si la velocidad de
  transmisión no coincide, los datos pueden no transmitirse o recibirse correctamente*/
  Serial.begin(9600);   //El pin 13 es una salida digital.
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO
void loop() {
  /*El código principal se coloca dentro de la instrucción loop() para que se ejecute
  interminablemente en el microcontrolador ATMEGA328P de Arduino*/
  /*La operación % significa módulo y lo que hace esta es dividir un número y ver el resultado de
    su residuo, en el caso de la operación 3%2 == 0, lo que está haciendo es dividir 3/2 y ver si
    su residuo es cero, que en este caso no lo sería, ya que residuo = 1.
      - n%2 == 0: La operación verifica si el valor de n es divisible por 2 sin dejar residuo.
        En otras palabras, verifica si n es un número par*/
  if(n%2 == 0){         //Si n es par se prende el led
    /*digitalWrite(Pin, State): Lo que hace este método es mandar una salida digital a un pin en
    específico que se indica como su primer parámetro, en su segundo parámetro se puede mandar la
    constante HIGH para mandar 5V al pin o LOW para mandar 0V, osea no mandar nada*/
    digitalWrite(led, HIGH);  //Con esta línea de código se prende el led
  }else{                //Si n es impar NO se prende el led
    digitalWrite(led, LOW);   //Con esta línea de código se apaga el led
  }
  n = n+1;              //Después de ver si n es par, se le suma 1 antes de su siguiente ejecución.
  //Posteriormente se reinicia la variable después de que haya llegado a n = 100.
  if(n == 100){
    n = 0;
  }

  /*analogRead(): El método se utiliza para leer valores analógicos de un pin específico, permitiendo
  leer la tensión analógica presente en un pin y convertirla en un valor digital.
      - Pines Analógicos A0, A1,..., A5: No es necesario configurarlos explícitamente, ya que el método
        se encarga de establecerlos como entradas analógicas automáticamente.
      - Pines Digitales 0, 1,..., 13: Estos pines antes de utilizarlos se deben establecer por medio del
        método pinMode() como entradas.
  El ADC del Arduino es de 10 bits, esto significa que cuando reciba su valor máximo de 5V, en la consola
  imprimirá (2^10)-1 = 1023, ya que es el valor máximo que puede convertir de analógico a digital porque
  recibe tensiones de 0 a 5V y por lo tanto cuenta con valores digitales de 0 a 1023 en formato decimal*/
  int data = analogRead(voltage);//Lectura analógica del puerto A0 para imprimirlo en la consola de Arduino
```

```
  /*Serial.println(): Método que imprime en las Herramientas Monitor Serie y Serial Plotter el valor dado
  en su parámetro.*/
  Serial.println(data);

  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos*/
  delay(1000);          //Esto retrasa 500 milisegundos el código antes de volver a ejecutarse.
}
```

Código Python

```
# -*- coding: utf-8 -*-


#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.


#LIBRERÍAS:
#PyQt5 - QtWidgets: La clase QtWidgets proporciona todos los elementos que conforman las interfaces gráficas
#(GUI) hechas con la librería PyQt5, entre dichas herramientas se incluyen:
# - Widgets: Elementos gráficos básicos de los que se conforma una GUI como lo son botones (QPushButton), cajas
#   de texto (QLineEdit), texto estático (QLabel), listas desplegables (QComboBox), barras de progreso
#   (QProgressBar), casillas de verificación (QCheckBox), imágenes (QLabel), etc.
#   - Diálogos: Los diálogos son elementos gráficos que  facilitan la interacción con el usuario, incluyen
#     widgets como cuadros de diálogo de mensajes (QMessageBox), cuadros de diálogo para seleccionar archivos o
#     directorios (QFileDialog), cuadros de diálogo de entrada de texto (QInputDialog), etc.
# - Layouts: Es un contenedor de PyQt5 que permite almacenar y organizar varios widgets, es el equivalente al
#   Panel de la librería PyQt5.
#           - Widget: Es un contenedor que puede almacenar widgets directamente, proporcionando funcionalidades
#             para mostrar, ocultar, establecer posición, tamaño, manejar eventos, etc. de los diferentes
#             botones, checkboxes, áreas de texto, comboboxes, radiobuttons, listboxes, etc.
# - Ventana: Es la ventana principal de la GUI, que a su vez contiene todos los contenedores con los widgets que
#   conforman la GUI, se declara a través de la clase QMainWindow de la librería pyQt5 y es el equivalente al
#   Frame de PyQt5.
from PyQt5 import QtWidgets
#PyQt5 - QtWidgets: Clase que incluye métodos para trabajar con temporizadores, tamaño de elementos, fechas,
#archivos, directorios, señales, hilos, subprocesos, etc.
from PyQt5 import QtCore
#matplotlib - Figure: La clase Figure es la base para crear y organizar los elementos gráficos en Matplotlib,
#que es una librería de graficación matemática.
#PyQt5 - QtGui: Clase que incluye clases y métodos para trabajar con gráficos, fuentes, colores, imágenes,
#íconos y otros elementos visuales utilizados en una interfaz gráfica (GUI) de PyQt5.
from PyQt5 import QtGui
```

```
#matplotlib - Figure: La clase Figure es la base para crear y organizar los elementos gráficos en Matplotlib,
#que es una librería de graficación matemática.
from matplotlib.figure import Figure
#matplotlib - FigureCanvasQTAgg: La clase FigureCanvasQTAgg proporcionada por la biblioteca Matplotlib en el
#módulo matplotlib.backends.backend_qt5agg se utiliza para mostrar y manejar gráficos generados por Matplotlib
#dentro de una ventana o contenedor de pyQt5. En este caso se utiliza para mostrar una gráfica que podría
#actualizar sus datos en tiempo real o no, en este caso se mantendrá estática.
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
import numpy as np #numpy: Librería que realiza operaciones matemáticas complejas (matriciales).
import time #time: Librería del manejo de tiempos, como retardos, contadores, etc.
import serial #serial: Librería que establece una comunicación serial con  microcontroladores, módems, etc.
import sys #sys: Librería que permite interactuar directamente con el sistema operativo y consola del ordenador.
import glob #glob: Librería que sirve para buscar archivos o directorios.

#GUI (Graphical User Interface): Es una ventana con elementos como botones, áreas de texto, desplegables,
#imágenes, etc. que sirven para realizar alguna acción de forma gráfica para el usuario. A continuación, veremos
#como se crean este tipo de elementos en Python utilizando la librería wxPython o PyQt5.
#La librería PyQt5 es una muy utilizada y versátil, su programación es mas sencilla y reducida en comparación
#con la librería wxPython, pero su principal desventaja y diferencia es que PyQt5 está disponible bajo dos
#licencias: una licencia comercial y una licencia GPL de código abierto, esto significa que si se desea
#desarrollar aplicaciones comerciales con PyQt5, se deberá adquirir una licencia comercial. Por otro lado,
#wxPython se distribuye bajo una licencia de código abierto y permite su uso tanto en aplicaciones comerciales
#como en proyectos de código abierto.

#GraficaPyQt5: Esta clase propia hereda de la clase FigureCanvasQTAgg, que pertenece a la librería PyQt5 y
#permite mostrar gráficos generados por Matplotlib en un Widget de PyQt5, permitiendo manejar eventos de
#interacción del usuario, como hacer zoom, seleccionar puntos en el gráfico, etc.
class GraficaPyQt5(FigureCanvasQTAgg):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor. Los parámetros que puede recibir el constructor de
    #la clase son los siguientes:
    # - figure: La instancia de la clase Figure que se va a asociar con el lienzo. Por defecto, se establece en
    #   None. Este parámetro es el primero y no usa la sintaxis: figure = valor, solamente se pone.
    # - parent: El widget padre al que se asocia el lienzo. Por defecto, se establece en None.
    # - width: El ancho del lienzo en píxeles o pulgadas.
    # - height: La altura del lienzo en píxeles o pulgadas.
    # - dpi: La resolución del lienzo en puntos por pulgada. Por defecto, se establece en 80 y puede afectar al
    #   tamaño de la gráfica.
    # - bgcolor: Indica el color de fondo del lienzo. Puede ser especificado a través de las siguientes formas:
    #       - Nombre color: Por medio de un string se pueden declarar valores que reconoce Matplotlib como por
    #         ejemplo, white', 'red', 'blue', etc.
    #       - Tupla RGB: Por medio de una tupla se especifican los tonos de color RGB (Rojo, Verde Azul) con
    #         valores que van de 0 a 1, como por ejemplo, (1, 1, 1) para blanco.
    def __init__(self, parent = None, width = None, height = None, dpi = None, bgcolor = (0.044, 0.06, 0.104)):
        #Figure(): Constructor de la clase Figure perteneciente a la librería Matplotlib, usado para crear un
```

```
        #lienzo en el que se puedan dibujar gráficos, actúando como un contenedor para los subgráficos.
        #A este objeto se le pueden asignar los parámetros indicados en el constructor de la clase
        #FigureCanvasQTAgg.
        fig = Figure(figsize = (width, height), dpi = dpi)
        #matplotlib.Figure().set_facecolor(): Método que sirve para establecer el color de fondo de una gráfica.
        #A este objeto se le puede asignar el parámetros bgcolor indicado en el constructor de la clase
        #FigureCanvasQTAgg.
        fig.set_facecolor(bgcolor)
        #matplotlib.Figure().add_subplot(): Método aplicado a un objeto de la clase Figure, perteneciente a la
        #librería Matplotlib, lo que hace es agregar un subgráfico al lienzo vacío, los números de su parámetro
        #lo que indican es:
        # - Primer Número:  Indica el número de filas de las subgráficas.
        # - Segundo Número: Indica el número de columnas de las subgráficas.
        # - Tercer Número:  Indica el índice de la subgráfica que se está creando en específico.
        #El parámetro 111 crea una sola gráfica de una columna con un solo espacio para mostrar una gráfica.
        #Se declaran como self.nombreObjeto los widgets a los que se les vaya a extraer o introducir datos en el
        #transcurso del funcionamiento de la interfaz gráfica.
        self.axes = fig.add_subplot(111)
        #matplotlib.Figure().add_subplot().set_xlabel(): Método para indicar el texto que aparece en el eje
        #horizontal de la gráfica, recibe los siguientes parámetros:
        # - xlabel: Especifica el texto que se mostrará en el eje x.
        # - fontname: Indica el estilo de la fuente:
        #       - Nombres de tipos de letra estándar: "Arial", "Times New Roman", "Helvetica", "Courier",
        #         "Monospaced", "Consolas", etc. Estos nombres deben ser compatibles con los tipos de letra
        #         instalados en el sistema operativo.
        #       - Nombres de tipos de letra genéricos: "serif", "sans-serif", "monospace", etc.
        #       - Rutas de archivo: Si se tiene un archivo de tipo de letra personalizado, se puede especificar
        #         la ruta del archivo como el valor de fontname.
        # - fontsize: Indica el tamaño de la fuente.
        # - labelpad: Especifica el espaciado entre la etiqueta del eje x y el eje en sí
        # - color: Indica el color del texto colocado en el eje x, para ello es válido usar colores:
        #       - Básicos CSS: como "red", "blue", "green", etc.
        #       - Colores hexadecimales: "#FF0000", "#00FF00", etc.
        #       - Colores RGB: "rgb(255,255,255)" para el color blanco y "rgb(0,0,0)" para el negro.
        self.axes.set_xlabel(xlabel = "Time [s]", fontname = "Consolas", fontsize = 8, color = "white")
        #matplotlib.Figure.add_subplot().set_ylabel(): Método para indicar el texto que aparece en el eje y.
        self.axes.set_ylabel(ylabel = "Voltage [V]", fontname = "Consolas", fontsize = 8, color = "white")
        #matplotlib.Figure().add_subplot().tick_params(): Método para acceder a las propiedades gráficas de los
        #números que aparecen en los ejes de la gráfica creada con la librería matplotlib.
        # - axis: Indica el eje que se quiere afectar, ya sea 'x', 'y', 'both' o 'all'.
        # - colors: Indica el color de los números colocados en el eje x o y, para ello es válido usar colores:
        #       - Básicos CSS: como "red", "blue", "green", etc.
        #       - Colores hexadecimales: "#FF0000", "#00FF00", etc.
        #       - Colores RGB: "rgb(255,255,255)" para el color blanco y "rgb(0,0,0)" para el negro.
        # - labelsize: Permite especificar el tamaño de letra en pixeles.
```

```
        # - pad: espacio entre las marcas y las etiquetas en puntos.
        self.axes.tick_params(axis = "x", colors = "white", labelsize = 8)
        self.axes.tick_params(axis = "y", colors = "white", labelsize = 8)
        #matplotlib.Figure.add_subplot().set_facecolor(): Método para indicar el color de fondo de la gráfica,
        #el cual puede ser indicado por los mismos colores previamente mencionados en el método plot() o se
        #pueden usar los siguientes con el código xkcd:
        # - Colores: xkcd:aqua, xkcd:aquamarine, xkcd:azure, xkcd:beige, etc. Los colores se obtienen de:
        #https://matplotlib.org/stable/tutorials/colors/colors.html
        self.axes.set_facecolor('#0f0223')

        #super(Llamada al constructor heredado).__init__(Parámetros que se le asignan): Lo que hace el método
        #super() es llamar al constructor de la clase padre de la clase actual (si es que no se le indica ningún
        #parámetro) o a cualquier clase que se le indique en su parámetro, después la instrucción .__init__()
        #asigna valores default a los parámetros del constructor de la clase padre (si es que no se indica
        #ningún parámetro), aunque de igual manera se pueden asignar parámetros adicionales, cualquier parámetro
        #incluido en el método init, será considerado como adicional. En conclusión, lo que está realizando la
        #línea de código es primero llamar al constructor de la superclase para realizar las tareas de
        #inicialización requeridas antes de indicar parámetros adicionales a la instancia de la clase actual.
        super(GraficaPyQt5, self).__init__(fig) #Asigna el objeto fig al constructor de la clase GraficaPyQt5.



#MainWindow: La clase hereda de la clase QMainWindow, que a su vez hereda de la clase QtWidgets y ambas
#pertenecen a la librería PyQt5. El elemento representa la ventana del GUI y crea una instancia de la clase
#GraficaPyQt5 para agregar dentro de la ventana una gráfica.
class MainWindow(QtWidgets.QMainWindow):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor.
    def __init__(self):
        #super(Llamada al constructor heredado).__init__(Parámetros que se le asignan): Lo que hace el método
        #super() es llamar al constructor de la clase padre de la clase actual (si es que no se le indica ningún
        #parámetro) o a cualquier clase que se le indique en su parámetro, después la instrucción .__init__()
        #asigna valores default a los parámetros del constructor de la clase padre (si es que no se indica
        #ningún parámetro), aunque de igual manera se pueden asignar parámetros adicionales, cualquier parámetro
        #incluido en el método init, será considerado como adicional. En conclusión, lo que está realizando la
        #línea de código es primero llamar al constructor de la superclase para realizar las tareas de
        #inicialización requeridas antes de indicar parámetros adicionales a la instancia de la clase actual.
        super(MainWindow, self).__init__() #Asigna parámetros estándar al constructor de la clase MainWindow.
        #PyQt5.QtWidgets.QMainWindow.setWindowTitle(): Método para colocar un título al Window creado con PyQt5.
        self.setWindowTitle("Instrumentación Virtual - PyQt5 con Arduino")

        #Instancia de la clase GraficaPyQt5 para agregar la gráfica al Window, se le debe pasar como parámetro
        #al constructor de la clase GraficaPyQt5, los parámetros que se quiera editar para que no se ejecuten
        #los que vienen declarados por defecto, los que no se editen, se ejecutarán con el valor que fueron
        #declarados arriba en el constructor.
```

```
        self.canvas = GraficaPyQt5(self, width = 5, height = 4, dpi = 120)

        #CREACIÓN DE LOS WIDGETS: Botón
        #Instancia de la librería PyQt5 por medio del constructor de la clase QPushButton que hereda de la clase
        #QtWidgets y sirve para crear un widget de tipo botón, en este se deben indicar como parámetros:
        # - parent: Especifica el objeto padre al que se asociará el botón. Si se proporciona, el botón se
        #   colocará dentro del widget padre.
        # - text: Con este parámetro se indica el texto que aparecerá sobre el botón.
        # - icon: Establece el ícono que se mostrará junto al texto del botón, debe utilizarse un objeto QIcon
        #   perteneciente a la librería QtGui para que se pueda agregar un ícono.
        # - checkable: Especifica si el botón funciona como un switch (que mantiene su estado) o como un push
        #   button (que no mantiene su estado a menos que se mantenga presionado).
        #       - checkable = True:     Botón tipo switch.
        #       - checkable = False:    Botón tipo push button.
        # - autoDefault: Indica si el botón es el predeterminado del diálogo. Si se establece en True, el botón
        #   responderá automáticamente a la tecla "Enter", sino no lo hará.
        # - flat: Con False se indica que el botón se muestre sin un marco, con True aparece el marco.
        # - menu: Especifica el menú desplegable asociado al botón.
        # - iconSize: Establece el tamaño del ícono del botón, para ello recibe un objeto QSize:
        #       - QtCore.QSize(ancho, alto): Objeto que indica el tamaño del ícono.
        #CREACIÓN DE ÍCONO PARA INCLUIR EN EL BOTÓN:
        #Variable que guarda el directorio y el nombre del archivo creado, se reemplazan los guiones \ por /
        #para poder leer una imagen o cualquier otro archivo, se usa la dirección relativa o absoluta de un
        #directorio:
        # - Dirección relativa: Es una dirección que busca un archivo desde donde se encuentra la carpeta del
        #   archivo python actualmente, esta se debe colocar entre comillas simples o dobles.
        # - Dirección absoluta: Es una dirección que coloca toda la ruta desde el disco duro C o cualquier otro
        #   que se esté usando hasta la ubicación del archivo, la cual se debe colocar entre comillas simples o
        #   dobles.
        #   ..      : Significa que nos debemos salir de la carpeta donde nos encontramos actualmente.
        #   /       : Sirve para introducirnos a alguna carpeta cuyo nombre se coloca después del slash.
        #   .ext    : Se debe colocar siempre el nombre del archivo + su extensión.
        iconPath = "Archivos_Ejercicios_Python/Img/LogoBlancoDi_cer0.png"
        #PyQt5.QtGui.QIcon(): Constructor de la clase QIcon que hereda de la clase QtGui y perteneciente a la
        #librería PyQt5, usado para crear un objeto que ícono que pueda ser añadido a cualquier widget como lo
        #puede ser un botón, un texto estático, etc. El tamaño de dicha imagen será reducido automáticamente.
        logoDicer0 = QtGui.QIcon(iconPath)
        #Se declaran como self.nombreObjeto los widgets a los que se les vaya a extraer o introducir datos en el
        #transcurso del funcionamiento de la interfaz gráfica, en este caso se aplica al botón porque este va a
        #cambiar el texto que tiene escrito cuando sea presionado.
        self.btn_start = QtWidgets.QPushButton(text = "\t\t\t\tStart", icon = logoDicer0, iconSize = QtCore.QSize(30, 30))
        self.btn_stop = QtWidgets.QPushButton(text = "\t\t\t\tStop", icon = logoDicer0, iconSize = QtCore.QSize(30, 30))
        self.btn_save = QtWidgets.QPushButton(text = "\t\t\t\tSave", icon = logoDicer0, iconSize = QtCore.QSize(30, 30))
        #widget.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos, no todos) a los
        #widgets de una interfaz gráfica de usuario (GUI).
```

```
        # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
        #   esta no es admitida por PyQt5:
        #   background: qlineargradient(x1:punto_inicial, y1:punto_inicial, x2:punto_final, y2:punto_final, stop:0
rgb(R_inicial,G_inicial,B_inicial), stop:1 rgb(R_final,G_final,B_final));
        self.btn_start.setStyleSheet("background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,255,255), stop:1
rgb(91,150,242));")
        self.btn_stop.setStyleSheet("background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,255,255), stop:1
rgb(150,0,0));")
        self.btn_save.setStyleSheet("background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,255,255), stop:1
rgb(198,132,0));")
        #widget.Hide(): Método que sirve para esconder un widget en la GUI.
        self.btn_stop.hide()            #Esconde inicialmente el botón de STOP
        self.btn_save.hide()            #Esconde inicialmente el botón de SAVE

        #CREACIÓN DE LOS WIDGETS: Texto Estático, clase QLabel
        #Instancia de la librería PyQt5 por medio del constructor de la clase QLabel que hereda de la clase
        #QtWidgets y sirve para crear un widget que muestre un texto estático o una imagen en una interfaz
        #gráfica, se le deben indicar los siguientes parámetros cuando se usa para crear texto estático:
        # - parent: Especifica el widget padre del QLabel. Si se proporciona, el texto estático se colocará
        #   dentro del widget padre.
        # - text: Permite especificar el texto que se mostrará en el widget. Puede ser una cadena de texto en
        #   formato plano o enriquecido con etiquetas HTML.
        #Para dar estilo al QLabel cuando se utiliza para mostrar texto estático es mejor utilizar etiquetas
        #HTML que contengan un style que les dé estilo por medio de instrucciones CSS, además es importante
        #mencionar que para el style se deben usar comillas simples ('') para que no tenga conlficto con las
        #comillas dobles del parámetro text = "". PyQt5 acepta algunas instrucciones CSS pero no todas.
        lbl_com_port = QtWidgets.QLabel(text = "<p style='background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0
rgb(41,54,108), stop:1 rgb(34,15,64)); font-size: 15px; font-family: Courier New, monospace; color: white;'>COM Port: </p>")
        lbl_samples = QtWidgets.QLabel(text = "<p style='background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0
rgb(41,54,108), stop:1 rgb(34,15,64)); font-size: 15px; font-family: Courier New, monospace; color: white;'>Samples: </p>")

        #CREACIÓN DE LOS WIDGETS: Combo Box, Lista Desplegable
        #Instancia de la librería PyQt5 por medio del constructor de la clase QComboBox que hereda de la clase
        #QtWidgets y sirve para crear un widget que muestre una lista desplegable de elementos seleccionables
        #en una ventana o layout.
        self.cb_port = QtWidgets.QComboBox() #Combo box para mostrar todos los puertos detectados
        #widget.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos, no todos) a los
        #widgets de una interfaz gráfica de usuario (GUI).
        # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
        #   esta no es admitida por PyQt5:
        #   background: qlineargradient(x1:punto_inicial, y1:punto_inicial, x2:punto_final, y2:punto_final, stop:0
rgb(R_inicial,G_inicial,B_inicial), stop:1 rgb(R_final,G_final,B_final));
        self.cb_port.setStyleSheet("font-size: 15px; font-family: Courier New, monospace; color: black; background:
qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,255,255), stop:1 rgb(91,150,242));")
        #QtWidgets.QComboBox.addItems(): Método que se utiliza para agregar una lista de elementos al combo box.
```

```
        #Este puede recibir como parámetro directamente una lista, tupla o diccionario que representen los
        #elementos que se agregarán al combo box o puede recibir una función propia que cree dicha lista, para
        #luego añadir dichos elementos creados por la función al QComboBox.
        #Se ejecuta la función propia SerialPorts() declarada fuera del constructor pero dentro de la clase
        #MainWindow para obtener los puertos disponibles del ordenador actual por medio de la clase serial.
        self.cb_port.addItems(self.SerialPorts())

        #CREACIÓN DE LOS WIDGETS: Spin Box, control numérico
        #Instancia de la librería PyQt5 por medio del constructor de la clase QSpinBox que hereda de la clase
        #QtWidgets y sirve para crear un widget  que muestre un selector de números en una ventana o layout.
        spb_samples = QtWidgets.QSpinBox()
        #QtWidgets.QSpinBox.setMinimum(Valor_Mínimo): Método que indica el valor mínimo permitido en el control
        #numérico SpinBox.
        spb_samples.setMinimum(1)
        #QtWidgets.QSpinBox.setMaximum(Valor_Máximo): Método que indica el valor máximo permitido en el control
        #numérico SpinBox. El máximo de datos que puede recopilar un archivo de Excel sin fallar son 32,000.
        spb_samples.setMaximum(32000)
        #QtWidgets.QSpinBox.setSingleStep(Intervalo): Indica de cuánto en cuanto avanza el valor numérico del
        #SpinBox.
        spb_samples.setSingleStep(1)
        #widget.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos, no todos) a los
        #widgets de una interfaz gráfica de usuario (GUI).
        # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
        #   esta no es admitida por PyQt5:
        #   background: qlineargradient(x1:punto_inicial, y1:punto_inicial, x2:punto_final, y2:punto_final, stop:0
rgb(R_inicial,G_inicial,B_inicial), stop:1 rgb(R_final,G_final,B_final));
        spb_samples.setStyleSheet("font-size: 15px; font-family: Courier New, monospace; color: black; background:
qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,255,255), stop:1 rgb(91,150,242));")

        #CONTENEDORES DE ELEMENTOS: La biblioteca PyQt5 ofrece varios tipos de contenedores que se pueden
        #utilizar para organizar los widgets en una interfaz gráfica. Los más comunes son:
        # - QVBoxLayout: Organiza los widgets en una disposición vertical, uno debajo del otro.
        # - QHBoxLayout: Organiza los widgets en una disposición horizontal, uno al lado del otro.
        # - QGridLayout: Organiza los widgets en una cuadrícula bidimensional de filas y columnas.
        # - QFormLayout: Diseñado específicamente para crear formularios, donde los widgets se colocan en pares
        #   de etiqueta y campo de entrada.
        # - QStackedLayout: Permite apilar varios widgets uno encima del otro y mostrar uno a la vez.
        # - QTabWidget: Permite crear pestañas donde se pueden colocar diferentes conjuntos de widgets en cada
        #   pestaña.
        # - QScrollArea: Proporciona una vista desplazable para un contenido que puede ser mayor que el área
        #   visible.
        # - QGroupBox: Crea un grupo que puede contener y organizar otros widgets.
        # - QSplitter: Permite dividir el área de visualización en secciones redimensionables que contienen
        #   widgets diferentes.
        # - QWidget: Proporciona una ventana o área rectangular en la que se pueden colocar otros widgets para
```

```
#   crear una interfaz gráfica, un QWidget puede contener otros widgets o contenedores dentro.
#Objeto de la clase QVBoxLayout, el cual se utiliza para organizar los widgets en una disposición
#vertical, proporcionando una forma conveniente de colocar los widgets uno debajo del otro en una
#ventana o en otro contenedor.
# - parent: Si el constructor de esta clase recibe como parámetro un objeto QWidget, ese será el
#   contenedor principal del objeto QVBoxLayout que organiza sus elementos verticalmente.
# - Si no recibe ningún parámetro, este es un contenedor vacío sin widget principal que aceptará
#   varios elementos o contenedores y los irá colocando verticalmente uno después del otro.
main_layout = QtWidgets.QVBoxLayout()
#PyQt5.QtWidgets.QVBoxLayout.addWidget(): Método usado para añadir un widget de manera secuencial en la
#columna de un diseño vertical, como lo puede ser un botón, lista desplegable, texto, imagen, etc. Se
#indica el órden en el que se colocarán los elementos dependiendo de cual fue añadido primero y cual
#después.
main_layout.addWidget(self.canvas)
#Objeto de la clase QGridLayout, el cual se utiliza para organizar los widgets en una en una cuadrícula
#bidimensional de filas y columnas.
# - parent: Si el constructor de esta clase recibe como parámetro un objeto QWidget, ese será el
#   contenedor principal del objeto QGridLayout que organiza sus elementos en forma de rejilla.
# - Si no recibe ningún parámetro, este es un contenedor vacío sin widget principal que aceptará
#   varios elementos o contenedores y los irá colocando dependiendo de las coordenadas que se les
#   indique al utilizar el método .addWidget().
control_layout = QtWidgets.QGridLayout()
#PyQt5.QtWidgets.QGridLayout.addWidget(): Método usado para añadir un widget en una cuadrícula
#bidimensional compuesta por filas y columnas, donde la primera coordenada de filas y columnas se
#indica desde el número 0:
# - En su primer parámetro se indica el widget que se quiera aregar.
# - En su segundo parámetro se indica la fila donde se quiere colocar el elemento, contando desde 0.
# - En su tercer parámetro se indica la columna donde se quiere colocar el elemento, contando desde 0.
#Fila 1 = x = 0; Columna 1 = y = 0.
control_layout.addWidget(lbl_com_port, 0, 0)    #Agrega el texto estático que dice COM Ports: en (0,0)
control_layout.addWidget(self.cb_port, 1, 0)    #Agrega el Combo Box de puertos en (1,0)
control_layout.addWidget(lbl_samples, 0, 1)     #Agrega el texto estático que dice Samples: en (0,1)
control_layout.addWidget(spb_samples, 1, 1)     #Agrega el Spin Box, osea el control numérico en (1,1)
control_layout.addWidget(self.btn_start, 1, 2)  #Agrega el Botón de Start en (1,2)
control_layout.addWidget(self.btn_stop, 1, 3)   #Agrega el Botón de Stop en (1,3)
control_layout.addWidget(self.btn_save, 1, 4)   #Agrega el Botón de Save en (1,4)
#PyQt5.QtWidgets.QVBoxLayout.addLayout(): Método usado para añadir un layout (contenedor) de manera
#secuencial en la columna de un diseño vertical, se indica el órden en el que se colocarán los elementos
#dependiendo de cual fue añadido primero y cual después.
main_layout.addLayout(control_layout)
#Instancia de la clase QWidget, que hereda de la clase QtWidgets y pertenece a la librería PyQt5, dicho
#objeto funciona como un contenedor que puede almacenar widgets directamente, proporcionando
#funcionalidades para mostrar, ocultar, establecer posición, tamaño, manejar eventos, etc. de los
#diferentes botones, checkboxes, áreas de texto, comboboxes, radiobuttons, listboxes, ventanas de
#diálogo (ventana que muestra el explorador de archivos), etc.
```

```
        centralWidget = QtWidgets.QWidget()
        #PyQt5.QtWidgets.QVBoxLayout.setLayout(): Método usado para añadir un layout (que es un contenedor más
        #complejo) a un widget (que es un contenedor más sencillo), esto es útil hacerlo cuando se quiere
        #colocar el contenedor en una cierta posición (central, arriba, abajo, a la derecha o a la izquierda)
        #dentro de la ventana de la GUI.
        centralWidget.setLayout(main_layout)
        #ALINEACIÓN DE CONTENIDO EN UN WIDGET O LAYOUT:
        #PyQt5.QtWidgets.QMainWindow.setCentralWidget() = self.setCentralWidget(): Método aplicado al objeto de
        #la clase QMainWindow, del que hereda esta clase propia para establecer el widget central de la ventana
        #principal, ya que en PyQt5 las ventanas generalmente se dividen en diferentes áreas:
        # - Una barra de menú en la parte superior, para ello se debe indicar que objeto es el menu bar.
        #       - widget.setMenuBar(widget, QMenuBar)
        # - Una barra de herramientas opcional en la parte superior o inferior.
        #       - widget.setStatusBar(widget, QStatusBar)
        # - Un área central donde se coloca el contenido principal de la ventana.
        #       - widget.setCentralWidget(widget)
        #El método setCentralWidget() se utiliza para especificar qué widget se debe colocar en el área central
        #de la ventana creada con esta clase propia.
        self.setCentralWidget(centralWidget)            #Contenedor colocado en el área central de la GUI.

        #Instancia_Widget.evento_señal.connect(función_que_reacciona_al_evento): Este método se utiliza para
        #enlazar un evento a un controlador de eventos, que es una función que se ejecuta cuando ocurra el
        #evento, para ello se usa el nombre del widget, seguido del evento de tipo señal que detona el método,
        #la palabra reservdada .connect() y entre paréntesis se coloca el nombre de la función que ejecutará
        #alguna acción cuando ese evento ocurra. Normalmente las funciones que describen las acciones a
        #realizar por los widgets de la GUI se encuentran dentro de esta misma clase, pero fuera de su
        #constructor.
        # - Tipos de Eventos en Python:
        #       - clicked: Señal emitida cuando se hace clic en un elemento, como un botón.
        #       - doubleClicked: Señal emitida cuando se hace doble clic en un elemento.
        #       - pressed: Señal emitida cuando se presiona un elemento, como un botón.
        #       - released: Señal emitida cuando se suelta un elemento, como un botón.
        #       - textChanged: Señal emitida cuando el texto de un elemento, como un campo de texto, cambia.
        #       - currentIndexChanged: Señal emitida cuando se cambia el índice seleccionado en un elemento,
        #         como en un menú desplegable.
        #       - activated: Señal emitida cuando se selecciona un elemento, como un elemento de un menú
        #         desplegable o una opción de una lista.
        #       - keyPressed: Señal emitida cuando se presiona una tecla en el teclado.
        #       - keyReleased: Señal emitida cuando se suelta una tecla en el teclado.
        #       - mousePressEvent: Señal emitida cuando se presiona un botón del mouse.
        #       - mouseReleaseEvent: Señal emitida cuando se suelta un botón del mouse.
        #       - mouseMoveEvent: Señal emitida cuando se mueve el mouse.
        #       - valueChanged: Señal emitida cuando se selecciona un nuevo elemento en un combo box (lista
        #         desplegable).
        #       - timeout: Señal emitida cuando transcurre cada intervalo de tiempo especificado en un
```

```
        #         temporizador.
        #Es importante mencionar que en  PyQt5, cuando se conecta una función a un evento, la función conectada
        #puede recibir argumentos adicionales proporcionados por la señal emitida. Estos argumentos son
        #transmitidos automáticamente por el sistema de señales y slots de PyQt5.
        # - Tipos de argumentos retornados al suceder un evento:
        #       - *args y **kwargs: Muchas señales en PyQt5 permiten enviar argumentos adicionales a través de
        #          *args (tupla de argumentos posicionales) y **kwargs (diccionario de argumentos de palabras
        #          clave). Estos parámetros pueden variar según la señal específica y su contexto de uso.
        #       - checked: Algunas señales, como "clicked" en un botón, pueden enviar el estado de alternancia
        #         del widget. Este parámetro indica si el widget está marcado y suele ser de tipo booleano.
        #       - text: En widgets de entrada de texto, como QLineEdit o QTextEdit, las señales pueden enviar el
        #         texto ingresado o modificado como parámetro.
        #       - index: En widgets que tienen índices o selecciones, como QComboBox o QListView, las señales
        #         pueden enviar el índice seleccionado como parámetro.
        #       - position: En widgets que trabajan con eventos de posición, como QMouseEvent, las señales
        #         pueden enviar la posición del cursor o del evento como parámetro.
        #Al presionar un botón se ejecutará un método, declarado dentro de la misma clase.
        self.btn_start.clicked.connect(self.OnStartClick)       #Clic en Botón Start = OnStartClick()
        self.cb_port.activated.connect(self.add_port)           #Clic en una opción del Combo box = add_port()
        spb_samples.valueChanged.connect(self.samples_changed)  #Cambio en Spin box = samples_changed()
        self.btn_stop.clicked.connect(self.OnStopClick)         #Clic en Botón Stop  = OnStopClick()
        self.btn_save.clicked.connect(self.OnStartSaving)       #Clic en Botón Save  = OnStartSaving()

        #ATRIBUTOS DEL CONSTRUCTOR PERTENECIENTE A LA CLASE BottomPanel:
        self.com_port = ""              #Puerto seleccionado en el ComboBox

        self.period = 500               #Intervalo de muestreo (conteo) del Timer indicado en milisegundos.
        self.time_val = 0               #Variable que cuenta cada 1 segundos el tiempo de ejecución de la GUI.

        self.high_value_board = 5.0     #Valor de tensión Máxima = 5V
        self.board_resolution = 1023    #Resolución de 10 bits del ADC del Arduino: (2^10)-1 = 1023

        self.count = 0                  #Variable que cuenta los datos recopilados por la GUI.

        #Variable que obtiene los datos de tensión del pin A0 perteneciente al Arduino por medio de una
        #comunicación serial establecida a través del puerto elegido en el ComboBox.
        self.micro_board = None         #Datos de tensión obtenidos del pin A0 del Arduino.

        #QtWidgets.QSpinBox.minimum(): Método que obtiene el valor mínimo en un control numérico SpinBox.
        #La variable que guarda el número de muestras se inicializa con el valor mínimo del SpinBox.
        self.items = spb_samples.minimum()


    #función OnStartClick(): Método creado dentro de la clase propia llamada MainWindow que recibe como
    #parámetro el evento que lo activa, para posteriormente ejecutar cierta acción.
```

```
#En este caso el evento es activado por dar un clic sobre el botón de Start y lo que hace es primero checar
#si la comunicación serial está abierta, para poderla cerrar y volverla a abrir, luego checa si se ha
#seleccionado un puerto del ComboBox y si esto es cierto, iniciliza el temporizador y establece una
#comunicación serial con el puerto seleccionado, si no ha sido elegido ningún puerto del ListBox, muestra
#una ventana emergente que indique que no se ha seleccionado ningún puerto, además si es que ha ocurrido un
#error al intentar establecer la comunicación serial con el Arduino, mostrará un mensaje de error en una
#ventana emergente que indique tal cosa.
def OnStartClick(self):
    #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
    print("Start")
    print("Se recopilarán", self.items, "datos.")     #Se indica el número de muestras a recopilar.
    self.stp_acq = False            #Variable booleana que indica si se ha presionado el botón STOP.
    #MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
    # - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción
    #   durante su ejecución, el programa brinca al código del except
    # - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción
    #   cuando ocurra el error esperado.
    #Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que pueda
    #ocurrir un error durante su ejecución.
    try:
        #Instancia de la librería serial por medio del constructor de la clase Serial para establecer una
        #comunicación serial por medio de puertos seriales o USB con dispositivos externos como
        #microcontroladores, módems, teclados, impresoras, etc. Los parámetros que puede recibir el
        #constructor de la clase Serial son:
        # - port: Especifica el nombre en formato string del puerto serial al que se desea conectar.
        #           - Por ejemplo: "COM1" para sistemas operativos Windows o "/dev/ttyUSB1" para sistemas
        #             operativos Unix/Linux o iOS.
        # - baudrate: Define la velocidad de transmisión en baudios (bit trasmitido por segundo) para la
        #   comunicación serial.
        #           - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es
        #             compatible con la mayoría de los dispositivos y programas.
        #           - Sin embargo, si se necesita una transferencia de datos más rápida y el
        #             hardware/software lo admiten, se puede optar por velocidades más altas como 115200
        #             o 57600 baudios.
        # - bytesize: Especifica el tamaño de los bytes en la comunicación serial. Puede adoptar uno de los
        #   siguientes valores:
        #           - serial.FIVEBITS: Tamaño de 5 bits en los paquetes de la transmisión serial.
        #           - serial.SIXBITS: Tamaño de 6 bits en los paquetes de la transmisión serial.
        #           - serial.SEVENBITS: Tamaño de 7 bits en los paquetes de la transmisión serial.
        #           - serial.EIGHTBITS: Tamaño de 8 bits en los paquetes de la transmisión serial.
        # - parity: Indica el tipo de paridad utilizado en la comunicación serial. La paridad es un
        #   mecanismo utilizado en las comunicaciones seriales para verificar la integridad de los datos
        #   transmitidos, se basa en la adición de un bit adicional (bit de paridad) en el bit más
        #   significativo (hasta la izquierda) de cada paquete de datos transmitido. Al seleccionar la
        #   paridad, nos debemos asegurar de que tanto el dispositivo emisor como el receptor estén
```

```
#   configurados con la misma paridad para efecutar una comunicación adecuada:
#           - serial.PARITY_NONE: No se utiliza ningún bit de paridad. Esto implica que no se
#             verifica la integridad de los datos mediante la paridad.
#           - serial.PARITY_EVEN: Se utiliza la paridad par. Para ello se cuentan el número de bits
#             en el byte, incluido el bit de paridad:
#               - Si el número total de bits es impar, se establece el bit de paridad en 1 para que
#                 el número total de bits sea par.
#               - Si el número total de bits es par, se deja el bit de paridad en 0.
#                   - Por ejemplo, supongamos que se desea transmitir el byte 11010110. El bit de
#                     paridad en la transmisión de la comunicación se calcularía contando el número
#                     total de bits, que es 8, el número total de bits es par, por lo que el bit de
#                     paridad se establece en 0, Por lo tanto, el byte transmitido sería 011010110,
#                     donde el bit más significativo es el bit de paridad.
#                     Luego en el extremo receptor de la comunicación, se realizará un cálculo
#                     similar para verificar la integridad de los datos. Si el número total de bits,
#                     incluido el bit de paridad, no coincide con la paridad esperada (en este caso,
#                     par), se puede detectar un error en la transmisión de datos.
#           - serial.PARITY_ODD: Se utiliza paridad impar. El bit de paridad se establece de manera
#             que el número total de bits en el byte transmitido (incluido el bit de paridad) sea
#             impar.
#           - serial.PARITY_MARK: Se utiliza paridad de marca. El bit de paridad se establece en 1
#             (marcado) para todos los bytes transmitidos.
#           - serial.PARITY_SPACE: Se utiliza paridad de espacio. El bit de paridad se establece en
#             0 (espacio) para todos los bytes transmitidos.
# - stopbits: Define el número de bits de parada en la comunicación serial. El número de bits de
#   parada se utiliza para indicar el final de cada byte transmitido en la comunicación serial.
#   La elección del número de bits de parada depende de la configuración del dispositivo externo con
#   el que se está comunicando. El parámetro uno de los siguientes valores:
#           - serial.STOPBITS_ONE: Indica que se utiliza un bit de parada.
#           - serial.STOPBITS_ONE_POINT_FIVE: Indica que se utiliza un bit y medio de parada. Este
#             valor puede ser utilizado en algunas configuraciones especiales.
#           - serial.STOPBITS_TWO: Indica que se utilizan dos bits de parada.
# - timeout: Especifica el tiempo de espera en segundos para las operaciones de lectura. Si no se
#   recibe ningún dato dentro de este tiempo, la operación de lectura se interrumpe.
# - xonxoff: Con True o False indica si se utiliza el control de flujo XON/XOFF para la comunicación
#   serial.
# - rtscts: Con True o False indica si se utiliza el control de flujo RTS/CTS para la comunicación
#   serial.
# - dsrdtr: Con True o False indica si se utiliza el control de flujo DSR/DTR para la comunicación
#   serial.
# - write_timeout: Especifica el tiempo de espera en segundos para las operaciones de escritura.
#   Si no se puede escribir ningún dato dentro de este tiempo, la operación de escritura se
#   interrumpe.
# - inter_byte_timeout: Define el tiempo de espera en segundos entre la recepción de bytes
#   consecutivos durante las operaciones de lectura.
```

```
            #str(): Método que convierte un tipo de dato cualquiera en string.
            self.micro_board = serial.Serial(port = str(self.com_port), baudrate = 9600, timeout = 1)
            #time.sleep(): Método de la librería time que se utiliza para suspender la ejecución de un
            #programa durante un intervalo de tiempo específico dado en segundos.
            time.sleep(1)
        except:
            #PyQt5.QtWidgets.QMessageBox(): Método de la librería PyQt5 que se utiliza para mostrar una ventana
            #emergente en la interfaz gráfica. Esta ventana de diálogo muestra un mensaje específico al usuario
            #y puede contener botones para que el usuario realice una acción, como aceptar, cancelar, etc.
            dlg_board = QtWidgets.QMessageBox()
            #PyQt5.QtWidgets.QMessageBox.setWindowTitle(): Método para colocar un título en la ventana creada
            #con la librería PyQt5.
            dlg_board.setWindowTitle("Error en Instrumentación uy no!")
            #PyQt5.QtWidgets.QMessageBox.setText(): Método que se utiliza para establecer el texto principal de
            #un cuadro de diálogo QMessageBox en PyQt5. Este método recibe el siguiente parámetro:
            # - text: Indica el texto que se mostrará en el cuerpo principal del cuadro de diálogo. Puede ser
            #   una cadena de texto o admitir el formato HTML para formatear el texto. El nombre del parámetro
            #   no se menciona explícitamente.
            str_dlg_board = "<h3>No tienes ningún puerto seleccionado!</h3>"
            dlg_board.setText(str_dlg_board)
            #PyQt5.QtWidgets.QMessageBox.setStandardButtons(): Método usado para establecer los botones estándar
            #que se mostrarán en el cuadro de diálogo, esto se realiza a través de los siguientes parámetros:
            # - QtWidgets.QMessageBox.Ok: Botón "Aceptar".
            # - QtWidgets.QMessageBox.Open: Botón "Abrir".
            # - QtWidgets.QMessageBox.Save: Botón "Guardar".
            # - QtWidgets.QMessageBox.Cancel: Botón "Cancelar".
            # - QtWidgets.QMessageBox.Close: Botón "Cerrar".
            # - QtWidgets.QMessageBox.Yes: Botón "Sí".
            # - QtWidgets.QMessageBox.No: Botón "No".
            # - QtWidgets.QMessageBox.Abort: Botón "Abortar".
            # - QtWidgets.QMessageBox.Retry: Botón "Reintentar".
            # - QtWidgets.QMessageBox.Ignore: Botón "Ignorar".
            # - QtWidgets.QMessageBox.Reset: Botón "Restablecer".
            # - QtWidgets.QMessageBox.Help: Botón "Ayuda".
            # - QtWidgets.QMessageBox.Apply: Botón "Aplicar".
            # - QtWidgets.QMessageBox.YesToAll: Botón "Sí a todo".
            # - QtWidgets.QMessageBox.NoToAll: Botón "No a todo".
            # - QtWidgets.QMessageBox.SaveAll: Botón "Guardar todo".
            # - QtWidgets.QMessageBox.Default: Botón "Predeterminado".
            # - QtWidgets.QMessageBox.RestoreDefaults: Botón "Restaurar predeterminados".
            #Puedes combinar varios botones utilizando la compuerta lógica OR (|) para mostrar varios botones en
            #el cuadro de diálogo.
            dlg_board.setStandardButtons(QtWidgets.QMessageBox.Retry)
            #PyQt5.QtWidgets.QMessageBox.setIcon(): Método que se utiliza para establecer el ícono que se
            #muestra en un cuadro de diálogo QMessageBox. Recibe un parámetro que especifica el ícono a mostrar.
```

```
        #Los valores posibles para el parámetro son:
        # - QtWidgets.QMessageBox.NoIcon: No se muestra ningún ícono.
        # - QtWidgets.QMessageBox.Information: Muestra un ícono de información.
        # - QtWidgets.QMessageBox.Warning: Muestra un ícono de advertencia.
        # - QtWidgets.QMessageBox.Critical: Muestra un ícono de error/crítico.
        # - QtWidgets.QMessageBox.Question: Muestra un ícono de pregunta.
        dlg_board.setIcon(QtWidgets.QMessageBox.Information)
        #PyQt5.QtWidgets.QMessageBox.exec_(): Método para que se ejecute en un loop infinito el GUI,
        #logrando así que no se ejecute una vez y luego cierre por sí solo, sino que solo se cierre
        #solamente al dar clic en el tache de la ventana emergente.
        dlg_board.exec_()
        #Reinicio de la variable que guarda los datos de tensión obtenidos del pin A0 del Arduino.
        self.micro_board = None

    #Condicional if que checa si ya se ha seleccionado algún puerto y además se ha iniciado la conexión
    #serial, empezado ya a recopilar los datos de tensión del pin A0; si ésta es diferente de None, inicia
    #el conteo con un temporizador y empieza a almacenar el tiempo transcurrido y los datos de tensión
    #recabados en una lista que los relacione entre sí.
    if(self.com_port != "" and self.micro_board != None):
        #widget.Hide(): Método que sirve para esconder un widget en la GUI.
        self.btn_start.hide()       #Esconde el botón de START.
        self.btn_save.hide()        #Esconde el botón de SAVE.
        #widget.Show(): Método que sirve para mostrar un widget en la GUI.
        self.btn_stop.show()        #Muestra el botón de STOP.

        #Condicional if que comprueba que el valor de la variable que cuenta los datos recopilados por la
        #GUI es igual a cero, para que cuando esto sea cierto, se reinicie el valor de todas las variables
        #antes de comenzar de nuevo una recopilación de datos.
        if(self.count == 0):
            self.x = np.array([])   #Reinicio del numpy array del eje horizontal (x = tiempo [s]).
            self.y = np.array([])   #Reinicio del numpy array del eje vertical (y = tensión [V]).
            self.values = []        #Reinicio de la lista que guarda los valores de tiempo y tensión.

            self.time_val = 0       #Reinicio de la variable que cuenta cada 1 segundo.

            #Condicional if que comprueba si ya existía un dato restante en la variable que almacena los
            #datos de tensión recopilados del pin A0, si es así se aplica un método que limpie el buffer de
            #la conexión serial.
            if(self.micro_board != None):
                #serial.Serial().reset_input_buffer(): Método aplicado a un objeto de la clase Serial. Se
                #usa para eliminar los datos almacenados en el búfer de entrada del puerto serie. Esto puede
                #ser útil en situaciones donde se desea limpiar el búfer de entrada antes de iniciar una
                #nueva comunicación o cuando se necesite asegurar que no haya datos antiguos en el búfer
                #antes de comenzar a recibir nuevos datos.
                self.micro_board.reset_input_buffer()
```

```
#CREACIÓN DE WIDGETS: Temporizador
#Instancia de la librería PyQt5 por medio del constructor de la clase Timer, que hereda de la
#clase QtCore y se utiliza para crear un widget de tipo temporizador.
self.timer = QtCore.QTimer()
#INICIACIÓN DEL TEMPORIZADOR:
#QtCore.QTimer.setInterval(intervalo): Método que indica el intervalo de conteo de un
#temporizador en milisegundos.
self.timer.setInterval(self.period)
#Instancia_Widget.evento_señal.connect(función_que_reacciona_al_evento): Este método se utiliza
#para enlazar un evento a un controlador de eventos, que es una función que se ejecuta cuando
#ocurra el evento, para ello se usa el nombre del widget, seguido del evento de tipo señal que
#detona el método, la palabra reservdada .connect() y entre paréntesis se coloca el nombre de la
#función que ejecutará alguna acción cuando ese evento ocurra. Normalmente las funciones que
#describen las acciones a realizar por los widgets de la GUI se encuentran dentro de esta misma
#clase, pero fuera de su constructor.
# - Tipos de Eventos en Python:
#       - clicked: Señal emitida cuando se hace clic en un elemento, como un botón.
#       - doubleClicked: Señal emitida cuando se hace doble clic en un elemento.
#       - pressed: Señal emitida cuando se presiona un elemento, como un botón.
#       - released: Señal emitida cuando se suelta un elemento, como un botón.
#       - textChanged: Señal emitida cuando el texto de un elemento, como un campo de texto,
#         cambia.
#       - currentIndexChanged: Señal emitida cuando se cambia el índice seleccionado en un
#         elemento,
#         como en un menú desplegable.
#       - activated: Señal emitida cuando se selecciona un elemento, como un elemento de un menú
#         desplegable o una opción de una lista.
#       - keyPressed: Señal emitida cuando se presiona una tecla en el teclado.
#       - keyReleased: Señal emitida cuando se suelta una tecla en el teclado.
#       - mousePressEvent: Señal emitida cuando se presiona un botón del mouse.
#       - mouseReleaseEvent: Señal emitida cuando se suelta un botón del mouse.
#       - mouseMoveEvent: Señal emitida cuando se mueve el mouse.
#       - valueChanged: Señal emitida cuando se selecciona un nuevo elemento en un combo box
#         (lista desplegable).
#       - timeout: Señal emitida cuando transcurre cada intervalo de tiempo especificado en un
#         temporizador.
#Es importante mencionar que en  PyQt5, cuando se conecta una función a un evento, la función
#conectada puede recibir argumentos adicionales proporcionados por la señal emitida. Estos
#argumentos son transmitidos automáticamente por el sistema de señales y slots de PyQt5.
# - Tipos de argumentos retornados al suceder un evento:
#       - *args y **kwargs: Muchas señales en PyQt5 permiten enviar argumentos adicionales a
#         través de *args (tupla de argumentos posicionales) y **kwargs (diccionario de
#         argumentos de palabras clave). Estos parámetros pueden variar según la señal
#         específica y su contexto de uso.
```

```
            #       - checked: Algunas señales, como "clicked" en un botón, pueden enviar el estado de
            #         alternancia del widget. Este parámetro indica si el widget está marcado y suele ser de
            #         tipo booleano.
            #       - text: En widgets de entrada de texto, como QLineEdit o QTextEdit, las señales pueden
            #         enviar el texto ingresado o modificado como parámetro.
            #       - index: En widgets que tienen índices o selecciones, como QComboBox o QListView, las
            #         señales pueden enviar el índice seleccionado como parámetro.
            #       - position: En widgets que trabajan con eventos de posición, como QMouseEvent, las
            #         señales pueden enviar la posición del cursor o del evento como parámetro.
            #Cada que transcurra el intervalo de tiempo indicado en el temporizador, se ejecuta una función.
            self.timer.timeout.connect(self.update_plot)    #Intervalo transcurrido  = update_plot()
            #QtCore.QTimer.start(): Método que inicializa el conteo de un temporizador, para ello,
            #previamente se tuvo que haber usado el método .setInterval() para indicar su intervalo de
            #conteo en milisegundos.
            self.timer.start()
            print("\nTime [s] \t Voltage [V]")


#función update_plot(): Método creado dentro de la clase propia llamada MainWindow que recibe como
#parámetro el evento que lo activa, para posteriormente ejecutar cierta acción. En este caso el método se
#ejecuta Cada que transcurra el intervalo de tiempo indicado en el temporizador y lo que hace es actualizar
#el estado de la gráfica, para que los datos recopilados se muestren en tiempo real.
def update_plot(self):
    #MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
    # - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción
    #   durante su ejecución, el programa brinca al código del except
    # - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción
    #   cuando ocurra el error esperado.
    #Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que pueda
    #ocurrir un error durante su ejecución.
    try:
        #VECTOR TENSIÓN OBTENIDO DEL PIN A0:
        #serial.Serial.readline(): Método para leer una línea completa de datos desde el puerto serial.
        #Esta función lee caracteres desde el puerto serial hasta que encuentra un carácter de nueva línea
        #('\n') o una secuencia de retorno de carro-separador de línea ('\r\n'), que es es una combinación
        #de caracteres utilizada usualmente para indicar el final de una línea de texto en muchos sistemas
        #informáticos y de telecomunicaciones, donde el carácter de retorno de carro ('\r') mueve el cursor
        #hacia el inicio de la línea, como se hacía en las antiguas máquinas de escribir.
        #serial.Serial.decode(): Método que convierte los datos recibidos desde un dispositivo externo
        #conectado a través de una comunicación serial a una cadena de caracteres legible, se utiliza para
        #decodificar una secuencia de bytes en una cadena de caracteres utilizando un determinado esquema de
        #codificación:
        # - encoding (opcional): Especifica el esquema de codificación a utilizar para decodificar los bytes
        #   en una cadena de caracteres, si no se proporciona este parámetro, se utiliza la codificación
        #   predeterminada del sistema y los parámetros que puede recibir son los siguientes:
```

```
#           - 'utf-8': Esquema de codificación UTF-8 que es capaz de representar caracteres
#             especiales, letras acentuadas y otros caracteres no ASCII.
#           - 'utf-16 o utf-32': UTF-8 es ampliamente utilizado y recomendado para aplicaciones web
#             y transferencia de datos, mientras que UTF-16 y UTF-32 son comunes en sistemas que
#             requieren soporte completo para todos los caracteres Unicode.
#           - 'ascii': Especifica el esquema de codificación ASCII de 7 bits. Este esquema es
#             compatible con los caracteres ASCII estándar y no puede representar caracteres fuera
#             del rango ASCII.
#           - 'latin-1' o 'iso-8859-1': Especifica el esquema de codificación Latin-1. Este esquema
#             es capaz de representar los primeros 256 caracteres Unicode.
#           - 'cp437': El esquema de codificación CP437 fue ampliamente utilizado en los sistemas
#             informáticos antiguos, especialmente en los sistemas DOS y representa caracteres en un
#             rango de 8 bits, representando hasta 256 caracteres diferentes.
#str(): Método que convierte un tipo de dato cualquiera en string.
temp = str(self.micro_board.readline().decode('cp437'))
#El string crudo obtenido del método readline() y decodificado con el método decode() viene por
#default con una secuencia de retorno de carro-separador de línea ('\r\n'), por eso es que se debe
#utilizar el método replace para removerlo.
#replace(): Método que reemplaza un caracter que se encuentra en un string por otro declarado por
#nosotros, esto se ejecutará todas las veces que dicho caracter aparezca en el string.
temp = temp.replace("\n", "")

#CONVERSIÓN DE NUMEROS BINARIOS NUMÉRICOS DE TENSIÓN A VALORES DE TENSIÓN REALES:
#float(): Método que convierte un tipo de dato cualquiera en númerico decimal.
#Se realiza esta operación porque como el ADC del arduino lee de 0 a 5V y como tiene una resolución
#de 10 bits permitiendo que en el ADC los valores de tensión se interpreten como valores numéricos
#enteros que valen de 0 a (2^10)-1 = 1023, se hace una regla de 3 para que se imprima el valor de la
#tensión en consola en vez del valor decimal binario.
#Tensión = Tensión_decimal*(ValorMáximoTensión/ResoluciónADC) = Tensión_decimal*(5/1023)
value = (float(temp)*(self.high_value_board/self.board_resolution))

#VECTOR TIEMPO:
#Se usa una variable intermedia que va contando el tiempo transcurrido desde que se empezó a
#recopilar los valores de tensión del puerto analógico A0 del Arduino hasta que acaba. El intervalo
#de tiempo con el que cuenta el temporizador y el tiempo que se detiene el delay que se declarará
#después del except debe ser el mismo.
self.time_val = self.time_val + 1.0     #Variable que cuenta el tiempo de ejecución del programa.
self.count = self.count + 1             #Variable que cuenta los datos recopilados por la GUI.

#IMPRIMIR EN CONSOLA TIEMPO Y TENSIÓN:
msg_console = str(self.count) + ".- Time: " + str(self.time_val) + " [s]" + "\t"        #Tiempo [s].
msg_console += "Voltage: " + "{0:.5f}".format(value) + " [V]"                           #Volts [V].
print(msg_console)                                          #Imprimir en consola tiempo y tensión.

#append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
```

```
            #Lista que guarda los valores de tiempo [s] y tensión [V] recopilados.
            self.values.append(str(self.time_val) + ", " + "{0:.5f}".format(value))
            #matplotlib.Figure.add_subplot().cla(): Método cuyo nombre es una abreviatura de "clear axes" y
            #sirve para restablecer el estado de los ejes pertenecientes a una gráfica creada con la librería
            #matplotlib.
            self.canvas.axes.cla()
            self.x = np.append(self.x, self.time_val)           #Vector tiempo [segundos].
            self.y = np.append(self.y, value)                   #Vector tensión [V].

            #matplotlib.Figure().add_subplot().set_xlabel(): Método para indicar el texto que aparece en el eje
            #horizontal de la gráfica, recibe los siguientes parámetros:
            # - xlabel: Especifica el texto que se mostrará en el eje x.
            # - fontname: Indica el estilo de la fuente:
            #       - Nombres de tipos de letra estándar: "Arial", "Times New Roman", "Helvetica", "Courier",
            #         "Monospaced", "Consolas", etc. Estos nombres deben ser compatibles con los tipos de letra
            #         instalados en el sistema operativo.
            #       - Nombres de tipos de letra genéricos: "serif", "sans-serif", "monospace", etc.
            #       - Rutas de archivo: Si se tiene un archivo de tipo de letra personalizado, se puede
            #         especificar la ruta del archivo como el valor de fontname.
            # - fontsize: Indica el tamaño de la fuente.
            # - labelpad: Especifica el espaciado entre la etiqueta del eje x y el eje en sí
            # - color: Indica el color del texto colocado en el eje x, para ello es válido usar colores:
            #       - Básicos CSS: como "red", "blue", "green", etc.
            #       - Colores hexadecimales: "#FF0000", "#00FF00", etc.
            #       - Colores RGB: "rgb(255,255,255)" para el color blanco y "rgb(0,0,0)" para el negro.
            self.canvas.axes.set_xlabel(xlabel = "Time [s]", fontname = "Consolas", fontsize = 8, color = "white")
            #matplotlib.Figure.add_subplot().set_ylabel(): Método para indicar el texto que aparece en el eje y.
            self.canvas.axes.set_ylabel(ylabel = "Voltage [V]", fontname = "Consolas", fontsize = 8, color = "white")
            #matplotlib.Figure.add_subplot().plot(): Método usado para gráficar, indicando como primer parámetro
            #su eje horizontal, luego su eje vertical y finalmente el estilo de la gráfica:
            # - Colores:             C1: color naranja, r: color rojo, b: color azul, g: verde, c: cyan,
            #   m: morado, y: amarillo, k: negro, w: blanco.
            # - Tipo de marcadores:  o: círculos, +: símbolos de más, .; puntos, v: Triángulo hacia abajo,
            #   h: Hexágono, etc.
            # - Tipo de Líneas:      -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto,
            #   'or': Nada.
            #Los marcadores se obtienen del siguiente link: https://matplotlib.org/stable/api/markers_api.html
            self.canvas.axes.plot(self.x, self.y, 'c1:') #'c:' c: color cyan, 1: tri_down, :: línea punteada
            #matplotlib.FigureCanvas().draw(): Método que actualiza y muestra los datos recopilados en tiempo
            #real en la gráfica creada con el objeto que instancía la clase FigureCanvasQTAgg.
            self.canvas.draw()
        except:
            print("No se pudo actualizar la gráfica")

        #Condicional if que checa si se ha llegado al límite impuesto por el número de muestreos indicados en el
```

```
        #Spin Box, además de confirmar que el botón de Stop no ha sido activado, en caso de que cualquiera de
        #estas dos opciones sean ciertas, se detiene la recopilación de datos.
        if(self.count >= self.items or self.stp_acq == True):
            print("Se ha terminado de recopilar datos.")
            #widget.Hide(): Método que sirve para esconder un widget en la GUI, inicialmente los botones de
            #STOP y SAVE se encuentran escondidos por esta misma instrucción en el constructor, pero luego
            #al dar clic en el botón de START, se esconde el botón de START y se muestra el botón de STOP.
            self.btn_stop.hide()        #Esconde el botón de STOP.
            #widget.Show(): Método que sirve para mostrar un widget en la GUI.
            self.btn_start.show()       #Muestra el botón de START.
            self.btn_save.show()        #Muestra el botón de SAVE.

            #QtCore.QTimer.stop(): Método que detiene el conteo de un temporizador previamente empezado con el
            #método start().
            self.timer.stop()

            self.count = 0              #Reinicio de la variable que cuenta los datos recopilados por la GUI.
            self.stp_acq = False        #Variable booleana que indica si se ha presionado el botón STOP.

            #Condicional if que checa si hay algún puerto serial abierto, esto lo hace al ver el estado de la
            #variable booeana serialArduino, si esta es diferente de None, termina la comunicación serial, sino
            #sigue la ejecución del código como si nada.
            if(self.micro_board != None):
                #serial.Serial.close(): Método que cierra la comunicación serial. Es muy importante mencionar
                #que si no se ejecuta este método, el puerto serial se va a quedar bloqueado y no se podrá usar.
                self.micro_board.close()


    #función SerialPorts(): Método creado dentro de la clase propia llamada MainWindow que sirve para rellenar
    #los elementos del Combo Box que muestran todos los puertos disponibles en el ordenador a donde se podría
    #conectar la placa de desarrollo Arduino, de estos puertos se debe seleccionar el que haya sido elegido como
    #puerto de conexión dentro del IDE de Arduino.
    #def nombre_función -> tipo_de_dato: Es una sintaxis llamada anotación que se utiliza para indicar el tipo
    #de dato que devuelve una función. Es importante tener en cuenta que las anotaciones de tipo en Python son
    #opcionales y no afectan directamente el comportamiento o la ejecución de la función. Son principalmente
    #utilizadas para proporcionar información adicional a los desarrolladores.
    def SerialPorts(self) -> list:
        #sys.platform.startswith(): Método utilizado para para comprobar si el sistema operativo (OS) en el que
        #se está ejecutando este programa de Python coincide con una palabra específica, identificando si es:
        # - win:                Sistema operativo Windows.
        # - Linux o cygwin:     Sistema operativo Linux.
        # - darwin:             Sistema operativo iOS.
        #La variable sys.platform almacena un string que representa el sistema operativo en el que se está
        #ejecutando este programa de Python. El valor de sys.platform puede variar dependiendo del OS y la
        #configuración del entorno.
```

```
#El método startswith() comprueba si una cadena comienza con un string especificado, devolviendo True si
#el string original comienza con la cadena especificada y False en caso contrario.
if (sys.platform.startswith('win')):                                    #OS: Windows.
    #Bucle for en una sola línea: [instrucción      for   variable_local   in   range(inicio, final)]
    #Se ejecuta este bucle for en una sola línea para recopilar todos los puertos COM disponibles en el
    #ordenador actual, ya que en teoría Windows admite 256 puertos dependiendo del OS y del hardware.
    ports = ["COM%s" %(i+1) for i in range(256)]
    print("El sistema operativo que se está utilizando es: ", sys.platform)
elif(sys.platform.startswith('Linux') or sys.platform.startswith('cygwin')): #OS: Linux.
    #glob.glob(pathname): Método que devuelve una lista de rutas de archivos o directorios que coinciden
    #con el patrón especificado en pathname. El pathname puede contener palabras concretas o caracteres
    #comodín, denotados con asteriscos (*) o signos de interrogación (?), que representan uno o varios
    #caracteres en un nombre de archivo.
    ports = glob.glob("/dev/tty[A-Za-z]*")
    print("El sistema operativo que se está utilizando es: ", sys.platform)
elif(sys.platform.startswith('darwin')):                                #OS: iOS.
    #glob.glob(pathname): Método que devuelve una lista de rutas de archivos o directorios que coinciden
    #con el patrón especificado en pathname. El pathname puede contener palabras concretas o caracteres
    #comodín, denotados con asteriscos (*) o signos de interrogación (?), que representan uno o varios
    #caracteres en un nombre de archivo.
    ports = glob.glob("/dev/tty.*")
    print("El sistema operativo que se está utilizando es: ", sys.platform)
else:
    #raise: Instrucción que sirve para crear una excepción, esta a su vez debe ser parte de la clase
    #Exception para que sea un tipo de excepción correcta y dentro de su paréntesis se indica el mensaje
    #de error que arroja cuando se genere el error. Esta posible excepción debe ser cachada
    #posteriormente por una instrucción de manejo de excepciones (try except).
    raise EnvironmentError('Unsupported platform')

#Variable result donde se guardarán todos los puertos detectados por el programa dependiendo del sistema
#operativo
result = []

#Bucle for para intentar abrir todos los puertos enlistados y añadirlos a la lista result:
for port in ports:
    #MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
    # - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción
    #   durante su ejecución, el programa brinca al código del except
    # - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción
    #   cuando ocurra el error.
    #Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que
    #pueda ocurrir un error durante su ejecución.
    try:
        #Instancia de la librería serial por medio del constructor de la clase Serial para establecer
        #una comunicación serial por medio de puertos seriales o USB con dispositivos externos como
```

```
#microcontroladores, módems, teclados, impresoras, etc. Los parámetros que puede recibir el
#constructor de la clase Serial son:
# - port: Especifica el nombre en formato string del puerto serial al que se desea conectar.
#           - Por ejemplo: "COM1" para sistemas operativos Windows o "/dev/ttyUSB1" para
#             sistemas operativos Unix/Linux o iOS.
# - baudrate: Define la velocidad de transmisión en baudios (bit trasmitido por segundo) para la
#   comunicación serial.
#           - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es
#             compatible con la mayoría de los dispositivos y programas.
#           - Sin embargo, si se necesita una transferencia de datos más rápida y el
#             hardware/software lo admiten, se puede optar por velocidades más altas como 115200
#             o 57600 baudios.
# - bytesize: Especifica el tamaño de los bytes en la comunicación serial. Puede adoptar uno de
#   los siguientes valores:
#           - serial.FIVEBITS: Tamaño de 5 bits en los paquetes de la transmisión serial.
#           - serial.SIXBITS: Tamaño de 6 bits en los paquetes de la transmisión serial.
#           - serial.SEVENBITS: Tamaño de 7 bits en los paquetes de la transmisión serial.
#           - serial.EIGHTBITS: Tamaño de 8 bits en los paquetes de la transmisión serial.
# - parity: Indica el tipo de paridad utilizado en la comunicación serial. La paridad es un
#   mecanismo utilizado en las comunicaciones seriales para verificar la integridad de los datos
#   transmitidos, se basa en la adición de un bit adicional (bit de paridad) en el bit más
#   significativo (hasta la izquierda) de cada paquete de datos transmitido. Al seleccionar la
#   paridad, nos debemos asegurar de que tanto el dispositivo emisor como el receptor estén
#   configurados con la misma paridad para efecutar una comunicación adecuada:
#           - serial.PARITY_NONE: No se utiliza ningún bit de paridad. Esto implica que no se
#             verifica la integridad de los datos mediante la paridad.
#           - serial.PARITY_EVEN: Se utiliza la paridad par. Para ello se cuentan el número de
#             bits en el byte, incluido el bit de paridad:
#               - Si el número total de bits es impar, se establece el bit de paridad en 1 para
#                 que el número total de bits sea par.
#               - Si el número total de bits es par, se deja el bit de paridad en 0.
#                   - Por ejemplo, supongamos que se desea transmitir el byte 11010110. El bit
#                     de paridad en la transmisión de la comunicación se calcularía contando el
#                     número total de bits, que es 8, el número total de bits es par, por lo que
#                     el bit de paridad se establece en 0, Por lo tanto, el byte transmitido
#                     sería 011010110, donde el bit más significativo es el bit de paridad.
#                     Luego en el extremo receptor de la comunicación, se realizará un cálculo
#                     similar para verificar la integridad de los datos. Si el número total de
#                     bits, incluido el bit de paridad, no coincide con la paridad esperada (en
#                     este caso, par), se puede detectar un error en la transmisión de datos.
#           - serial.PARITY_ODD: Se utiliza paridad impar. El bit de paridad se establece de
#             manera que el número total de bits en el byte transmitido (incluido el bit de
#             paridad) sea impar.
#           - serial.PARITY_MARK: Se utiliza paridad de marca. El bit de paridad se establece en
#             1 (marcado) para todos los bytes transmitidos.
```

```
                #           - serial.PARITY_SPACE: Se utiliza paridad de espacio. El bit de paridad se establece
                #             en 0 (espacio) para todos los bytes transmitidos.
                # - stopbits: Define el número de bits de parada en la comunicación serial. El número de bits de
                #   parada se utiliza para indicar el final de cada byte transmitido en la comunicación serial.
                #   La elección del número de bits de parada depende de la configuración del dispositivo externo
                #   con el que se está comunicando. El parámetro uno de los siguientes valores:
                #           - serial.STOPBITS_ONE: Indica que se utiliza un bit de parada.
                #           - serial.STOPBITS_ONE_POINT_FIVE: Indica que se utiliza un bit y medio de parada.
                #             Este valor puede ser utilizado en algunas configuraciones especiales.
                #           - serial.STOPBITS_TWO: Indica que se utilizan dos bits de parada.
                # - timeout: Especifica el tiempo de espera en segundos para las operaciones de lectura. Si no
                #   se recibe ningún dato dentro de este tiempo, la operación de lectura se interrumpe.
                # - xonxoff: Con True o False indica si se utiliza el control de flujo XON/XOFF para la
                #   comunicación serial.
                # - rtscts: Con True o False indica si se utiliza el control de flujo RTS/CTS para la
                #   comunicación serial.
                # - dsrdtr: Con True o False indica si se utiliza el control de flujo DSR/DTR para la
                #   comunicación serial.
                # - write_timeout: Especifica el tiempo de espera en segundos para las operaciones de escritura.
                #   Si no se puede escribir ningún dato dentro de este tiempo, la operación de escritura se
                #   interrumpe.
                # - inter_byte_timeout: Define el tiempo de espera en segundos entre la recepción de bytes
                #   consecutivos durante las operaciones de lectura.
                #str(): Método que convierte un tipo de dato cualquiera en string.
                s = serial.Serial(port)     #Inicio de comunicación serial.
                #serial.Serial.close(): Método que cierra la comunicación serial. Es muy importante mencionar
                #que si no se ejecuta este método, el puerto serial se va a quedar bloqueado y no se podrá usar.
                s.close()                   #Terminación de la comunicación serial.
                #append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
                result.append(port)
            #Para identificar el tipo de excepción que ha ocurrido y utilizarlo en la instrucción except, se
            #puede utilizar la clase Exception, que es una clase incorporada en Python utilizada para describir
            #todos los tipos de excepciones, luego de colocar el nombre de la clase Exception se usa la palabra
            #reservada "as" seguida de un nombre de variable, esto nos permitirá acceder a la instancia de la
            #excepción y utilizarla dentro del except.
            except Exception as error:
                #type(clase).__name__: Esta instrucción no es un método, sino una expresión que se utiliza para
                #obtener el nombre de la clase de un objeto en Python, donde type(error) devuelve el tipo de
                #excepción en este caso ya que error es un objeto de una clase de excepción.
                # - __name__: Es un atributo especial en Python que se utiliza para obtener el nombre de la
                #   clase del objeto.
                print("Ocurrió el siguiente tipo de error al intentar conectarse a todos los puertos disponibles: ",
type(error).__name__)
                print("Este es el mensaje del error: ", error)
                #Aunque ocurra un error al tratar de encontrar todos los tipos de puertos, esto no significa
```

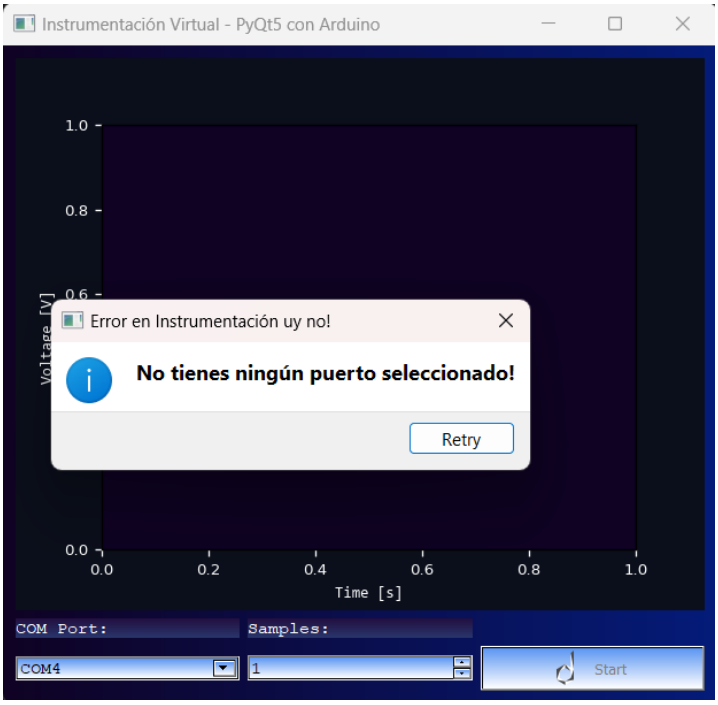
```
            #que el programa no vaya a funcionar, solo significa que no se ha podido conectar con todos los
            #puertos seriales que encontró en la computadora, muy seguramente porque puede que estos estén
            #siendo ya usados en otra cosa.

    print("Los puertos encontrados a los que se pudo conectar el programa fueron: \n", result)
    return result


#función add_port(): Método creado dentro de la clase propia llamada MainWindow que recibe como parámetro el
#evento que lo activa, para posteriormente ejecutar cierta acción. En este caso el método se ejecuta cada
#que se da clic en algún elemento contenido en el ComboBox y lo que hace es asignar su contenido a la
#variable cb_port que almacena el puerto que se quiere utilizar.
def add_port(self):
    #QtWidgets.QComboBox.currentText(): Método que devuelve el texto actualmente seleccionado en un objeto
    #QComboBox, este no recibe nada como parámetro y devuelve un string.
    self.com_port = self.cb_port.currentText()
    print("El puerto seleccionado fue: \n", self.com_port)


#función samples_changed(): Método creado dentro de la clase propia llamada MainWindow que recibe como
#parámetro el evento que lo activa y el número de muestras del control numérico Spin Box, para
#posteriormente ejecutar cierta acción. En este caso el método se ejecuta cada que cambia el valor del
#control numérico, por lo que se puede actualizar el número de muestras que se quiere recopilar en tiempo
#real y al hacerlo se asigna un nuevo valor a la variable items que almacena el número de muestras a
#recopilar.
def samples_changed(self, val_samples):
    self.items = val_samples
    #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
    print("Se recopilarán ", self.items, " datos.")


#función OnStopClick(): Método creado dentro de la clase propia llamada MainWindow que recibe como
#parámetro el evento que lo activa, para posteriormente ejecutar cierta acción.
#En este caso el evento es activado por dar un clic sobre el botón de Stop y lo que hace es primero esconder
#el botón de STOP, que previamente tuvo que ser activado y mostrado al dar clic en el botón de START y luego
#cambia el valor de la variable booleana stopAcquisition a True, al hacer esto se afectará la función
#update_plot(), deteniendo la ejecución del temporizador y logrando así que se detenga la recopilación de
#datos.
def OnStopClick(self):
    print("Stop")
    self.stp_acq = True             #Variable booleana que indica si se ha presionado el botón de STOP.


#función OnStartSaving(): Método creado dentro de la clase propia llamada MainWindow que recibe como
#parámetro el evento que lo activa, para posteriormente ejecutar cierta acción.
```

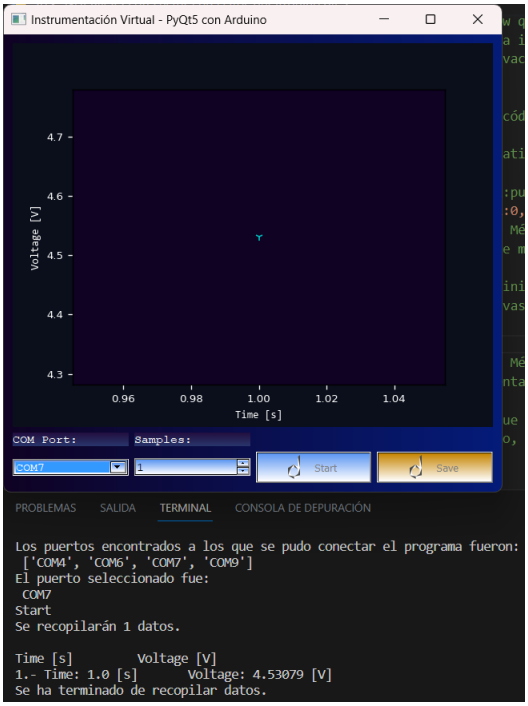
```
#En este caso el evento es activado por dar un clic sobre el botón de Save y lo que hace es abrir el
#explorador de archivos para nombrar el archivo Excel que guardará los datos recabados de tensión y tiempo,
#estos datos los tomará del vector self.values, creado en la función update_plot().
def OnStartSaving(self):
    print("Save Data")
    #Instancia de la librería PyQt5 por medio del constructor de la clase Options, que hereda de las clases
    #QFileDialog y QtWidgets para ejecutar un método que cree un cuadro de diálogo, el objeto Options se
    #puede modificar para especificar las opciones de comportamiento del diálogo de archivo, pero para ello
    #primero se debe crear un objeto y luego usar la compuerta lógica OR (|), ya que el constructor de la
    #clase Options() no recibe parámetros.
    options = QtWidgets.QFileDialog.Options()
    #Las opciones de configuración disponibles para el objeto Options() son:
    # - DontUseNativeDialog: Indica que no se debe utilizar el explorador de archivos nativo del sistema
    #   operativo y se debe utilizar el diálogo proporcionado por PyQt, esto significa que tendrá el mismo
    #   estilo que se indique a la ventana.
    # - ReadOnly: Abre el diálogo en modo de solo lectura, lo que impide al usuario guardar o modificar
    #   archivos existentes.
    # - HideNameFilterDetails: Oculta los detalles del filtro de nombre en el diálogo.
    # - DontResolveSymlinks: No resuelve los enlaces simbólicos al mostrar el diálogo de archivo.
    # - DontConfirmOverwrite: No muestra un mensaje de confirmación al guardar o sobrescribir un archivo
    #   existente.
    # - DontUseSheet: No utiliza una hoja de diálogo (específico de macOS).
    # - DontUseCustomDirectoryIcons: No utiliza iconos personalizados para los directorios en el diálogo.
    # - DontUseNativeFileSizeDisplay: No utiliza la representación nativa del tamaño de archivo en el
    #   diálogo.
    # - DontUseCustomDirectoryIcons: No utiliza iconos personalizados para los directorios en el diálogo.
    options |= QtWidgets.QFileDialog.DontUseNativeDialog
    #PyQt5.QtWidgets.QFileDialog.getSaveFileName(): El objeto QtWidgets.QFileDialog proporciona métodos para
    #mostrar una ventana de selección de archivos. Uno de estos métodos es getSaveFileName(), que se utiliza
    #para mostrar un diálogo de archivo para guardar un archivo y recibe los siguientes parámetros:
    # - parent: Es el objeto que se utiliza como referencia para mostrar el diálogo de archivo. Puede ser
    #   una ventana o un widget. Si se proporciona, la ventana se mostrará en la parte superior del objeto.
    # - caption: Es el título que se muestra en la parte superior de la ventana del explorador de archivos.
    # - directory: Es el directorio inicial que se muestra cuando se abre el diálogo de archivo. Se puede
    #   especificar una ubicación específica  o si no se proporciona solo se mostrará el directorio actual.
    # - filter: Son las opciones de filtro para los tipos de archivos que se mostrarán en la ventana. Se
    #   pueden especificar diferentes tipos de archivos separados por puntos y coma (;). Por ejemplo, se
    #   puede indicar un filtro para mostrar solo archivos CSV, PDF o todos los archivos.
    # - initialFilter: Es el filtro que se seleccionará inicialmente cuando se abra el explorador de
    #   archivos. Si se tiene varios filtros y solo uno se desea que sea seleccionado por defecto, se puede
    #   especificar en este parámetro.
    # - options: Son opciones adicionales para personalizar el comportamiento del explorador de archivos. Se
    #   puede usar para ocultar ciertos elementos, permitir la selección de múltiples archivos o mostrar
    #   miniaturas, modificar el aspecto estético de la ventaba, etc. Para ello se debe asignar el valor
    #   options al parámetro options y en el método main, usar métodos que afecten a todas las ventanas.
```

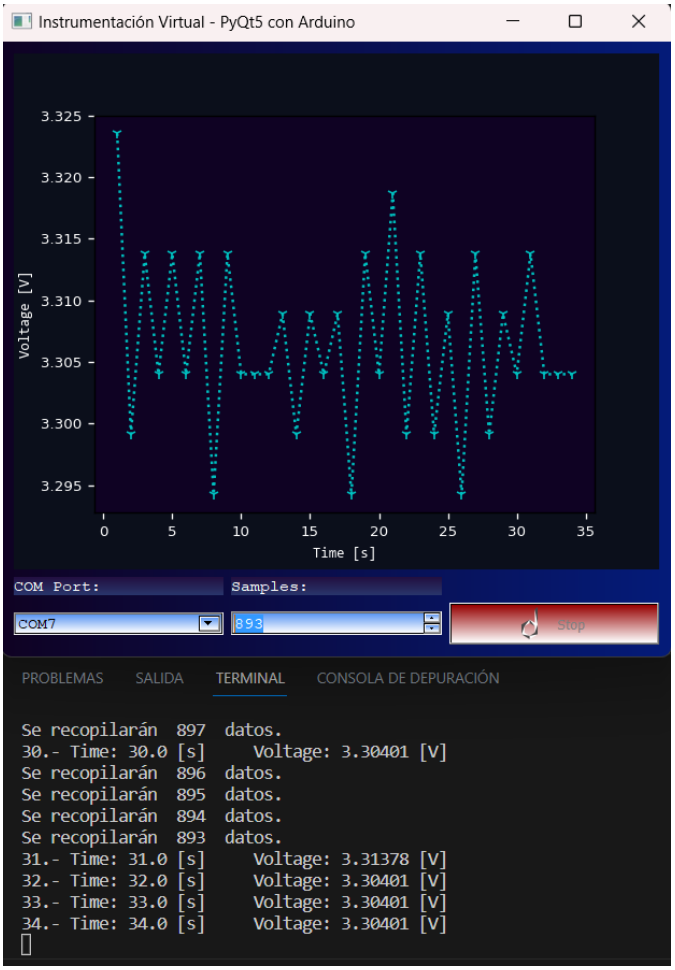
```
        #   Además se pueden combinar diferentes opciones utilizando operadores especiales.
        #El método .getSaveFileName() devuelve una tupla de dos valores:
        # - filename: Es un string que representa el nombre del archivo seleccionado por el usuario. Si el
        #   usuario no selecciona ningún archivo o cancela el diálogo, este valor será una cadena vacía.
        # - filter: Es un string que representa el filtro seleccionado por el usuario en el diálogo de archivo.
        #   El filtro corresponde al tipo de archivo seleccionado en el diálogo. Por ejemplo, si el usuario
        #   selecciona el filtro "csv Files (.csv)", este valor será la cadena "csv Files (.csv)". Si el usuario
        #   cancela el diálogo, este valor también será una cadena vacía.
        #En Python, si queremos almacenar ambos valores en dos variables distintas se usa la siguiente sintaxis:
        #       variable1, variable2 = QtWidgets.QFileDialog.getSaveFileName()
        #Pero si alguno de estos valores no nos interesa que se almacene en una variable, simplemente se coloca
        #un guión bajo para indicar eso, en este ejemplo el segundo valor de la tupla no se quiere usar:
        #       variable1, _ = QtWidgets.QFileDialog.getSaveFileName()
        nombreArchivo, _ = QtWidgets.QFileDialog.getSaveFileName(parent = self,
                                            caption = "Almacena los datos recopilados del pin A0 del Arduino",
                                            directory = "",
                                            filter = "csv Files (*.csv); ALL Files (*)",
                                            options = options)
        #En Python si solo se coloca un if con el nombre de una variable, la condición que se está evaluando
        #dentro de su paréntesis es que esa variable sea distinta de Null, si es así, se ejecuta la acción que
        #está dentro del condicional.
        #En este caso se está evaluando que ya se haya seleccionado un nombre de archivo para que se almacenen
        #los datos recabados del Arduino en él.
        if(nombreArchivo):
            #open(): Método que sirve para abrir un archivo cualquiera, para ello es necesario indicar dos
            #parámetros, el primero se refiere a la ruta relativa o absoluta del archivo previamente creado y la
            #segunda indica qué es lo que se va a realizar con él, el contenido del archivo se asigna a una
            #variable.
            # - w: Sirve para escribir en un archivo, pero borrará la información que previamente contenía el
            #   archivo.
            # - a: Sirve para escribir en un archivo sin que se borre la info anterior del archivo, se llama
            #   append.
            file = open(nombreArchivo, 'w')
            #myFile.write(): Método para colocar un string en un archivo previamente abrierto con el método
            #open(), en este caso se utiliza para colocar el valor de las dos columnas en el Excel, donde se
            #acomodan verticalmente los valores de tiempo y tensión recopilados.
            file.write("Library, PyQt5" + "\n")
            file.write("Time [s], Voltage [V]" + "\n")
            #Del vector values se obtienen los valores de tiempo y tensión recabados y agrupados.
            for i in range(len(self.values)):
                file.write(self.values[i]  + "\n")
            #file.close(): Método para cerrar un archivo previamente abrierto con el método open(), es peligroso
            #olvidar colocar este método, ya que la computadora lo considerará como si nunca hubiera sido
            #cerrado, por lo cual no podré volver a abrirlo al dar clic sobre él.
            file.close()
```

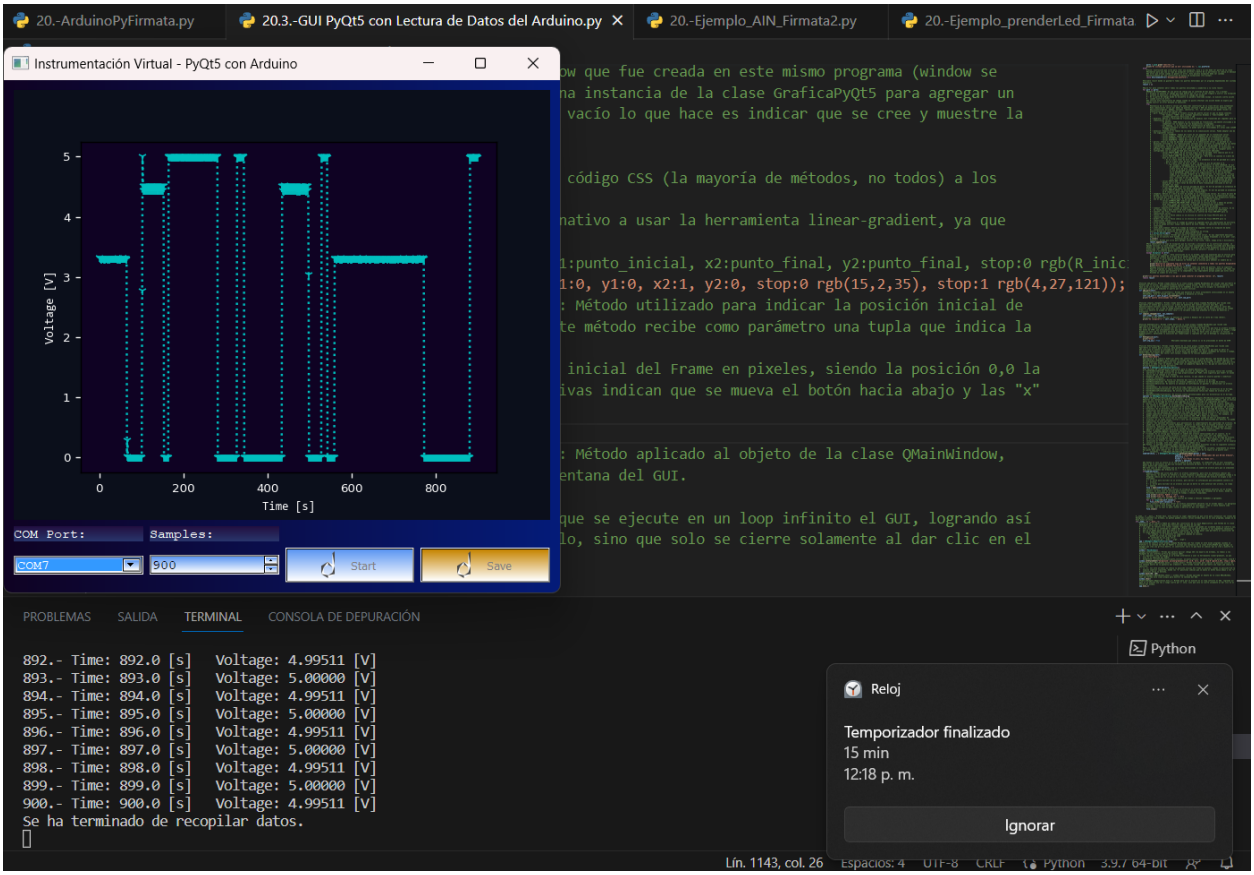
```
#__name__ == __main__: Método main, esta función es super importante ya que sirve para instanciar las clases del
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.
if(__name__ == "__main__"):
    #Instancia de la librería PyQt5 por medio del constructor de la clase QApplication, que hereda de la clase
    #QtWidgets para crear un objeto que funcione como la base de una GUI.
    # - sys.argv: Se refiere a un vector llamado "argument vector" que puede ser accedido desde la librería sys,
    #   este incluye en su contenido el nombre del archivo que se quiere ejecutar y los argumentos necesarios
    #   que se le deben pasar para que efectúe su ejecución. Se le debe pasar como parámetro al constructor de
    #   la clase QApplication para que se pueda crear la base de la GUI.
    #       - Por ejemplo, si se ejecutara el siguiente comando en consola:
    #           python mi_script.py arg1 arg2 arg3
    #         El contenido de sys.argv sería:
    #           ['mi_script.py', 'arg1', 'arg2', 'arg3']
    app = QtWidgets.QApplication(sys.argv)
    #Instancia de nuestra clase propia llamada MainWindow que fue creada en este mismo programa (window se
    #refiere a la ventana del GUI en PyQt5) e incluye una instancia de la clase GraficaPyQt5 para agregar un
    #widget que crea una gráfica dentro, el constructor vacío lo que hace es indicar que se cree y muestre la
    #ventana.
    window = MainWindow()
    #widget.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos, no todos) a los
    #widgets de una interfaz gráfica de usuario (GUI).
    # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
    #   esta no es admitida por PyQt5:
    #   background: qlineargradient(x1:punto_inicial, y1:punto_inicial, x2:punto_final, y2:punto_final, stop:0
rgb(R_inicial,G_inicial,B_inicial), stop:1 rgb(R_final,G_final,B_final));
    window.setStyleSheet("background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 rgb(15,2,35), stop:1 rgb(4,27,121));
color: gray;")
    #PyQt5.QtWidgets.QMainWindow.move() = window.move(): Método utilizado para indicar la posición inicial de
    #la ventana dentro de la pantalla del ordenaror, este método recibe como parámetro una tupla que indica la
    #posición:
    # - (x, y): Con este atributo se indica la posición inicial del Frame en pixeles, siendo la posición 0,0 la
    #   esquina superior izquierda, donde las "y" positivas indican que se mueva el botón hacia abajo y las "x"
    #   positivas hacia la derecha.
    window.move(100, 100)
    #PyQt5.QtWidgets.QMainWindow.show() = window.show(): Método aplicado al objeto de la clase QMainWindow,
    #del que hereda esta clase propia para mostrar la ventana del GUI.
    window.show()
    #PyQt5.QtWidgets.QApplication.exec_(): Método para que se ejecute en un loop infinito el GUI, logrando así
    #que no se ejecute una vez y luego cierre por sí solo, sino que solo se cierre solamente al dar clic en el
    #tache del window.
    app.exec_()
```

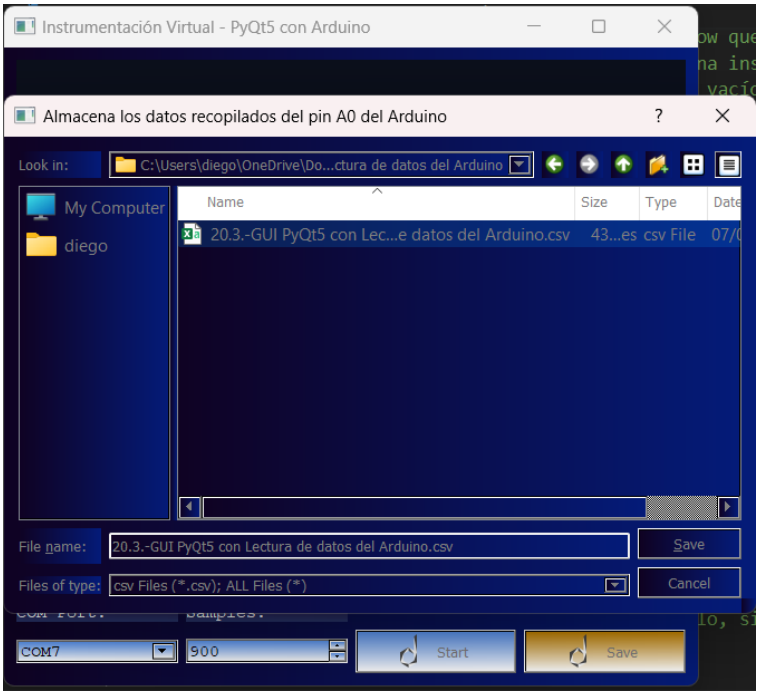
Resultado del Código Python

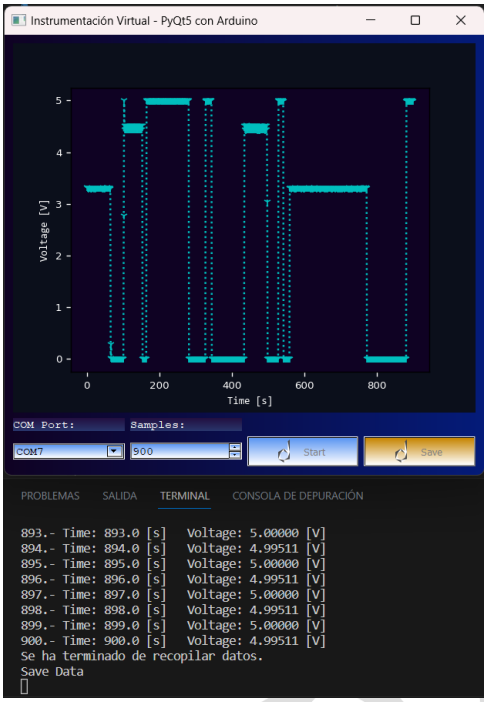


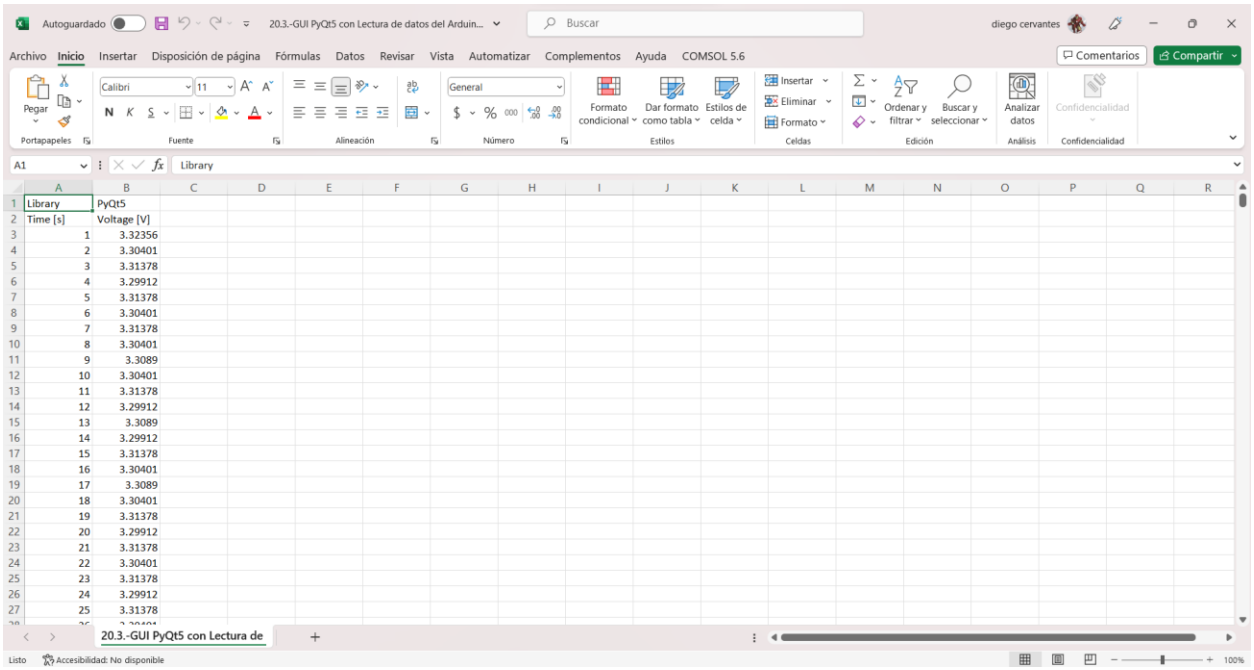




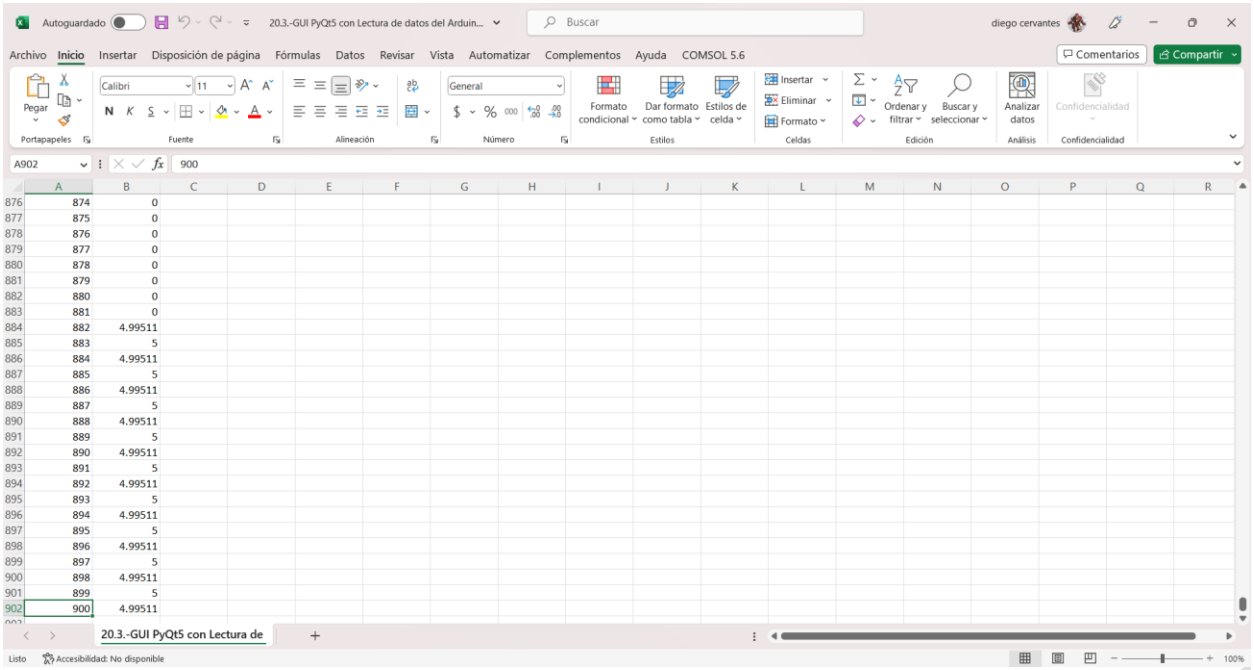

20.-ArduinoPyFirmata.py
20.3.-GUI PyQt5 con Lectura de Datos del Arduino.py
20.-Ejemplo_AIN_Firmata2.py
20.-Ejemplo_prenderLed_Firmata
Instrumentación Virtual - PyQt5 con Arduino
Voltage [V]
Time [s]
COM Port:
Samples:
COM7
900
Start
Save
PROBLEMAS
SALIDA
TERMINAL
CONSOLA DE DEPURACIÓN
892.- Time: 892.0 [s] Voltage: 4.99511 [V]
893.- Time: 893.0 [s] Voltage: 5.00000 [V]
894.- Time: 894.0 [s] Voltage: 4.99511 [V]
895.- Time: 895.0 [s] Voltage: 5.00000 [V]
896.- Time: 896.0 [s] Voltage: 4.99511 [V]
897.- Time: 897.0 [s] Voltage: 5.00000 [V]
898.- Time: 898.0 [s] Voltage: 4.99511 [V]
899.- Time: 899.0 [s] Voltage: 5.00000 [V]
900.- Time: 900.0 [s] Voltage: 4.99511 [V]
Se ha terminado de recopilar datos.
Reloj
Temporizador finalizado
15 min
12:18 p. m.
Ignorar
Python
Lín. 1143, col. 26


Instrumentación Virtual - PyQt5 con Arduino
Almacena los datos recopilados del pin A0 del Arduino
Look in:
C:\Users\diego\OneDrive\Do...ctura de datos del Arduino
My Computer
diego
Name
Size
Type
20.3.-GUI PyQt5 con Lec...e datos del Arduino.csv
43...es
csv File
File name:
20.3.-GUI PyQt5 con Lectura de datos del Arduino.csv
Save
Files of type:
csv Files (*.csv); ALL Files (*)
Cancel
COM7
900
Start
Save

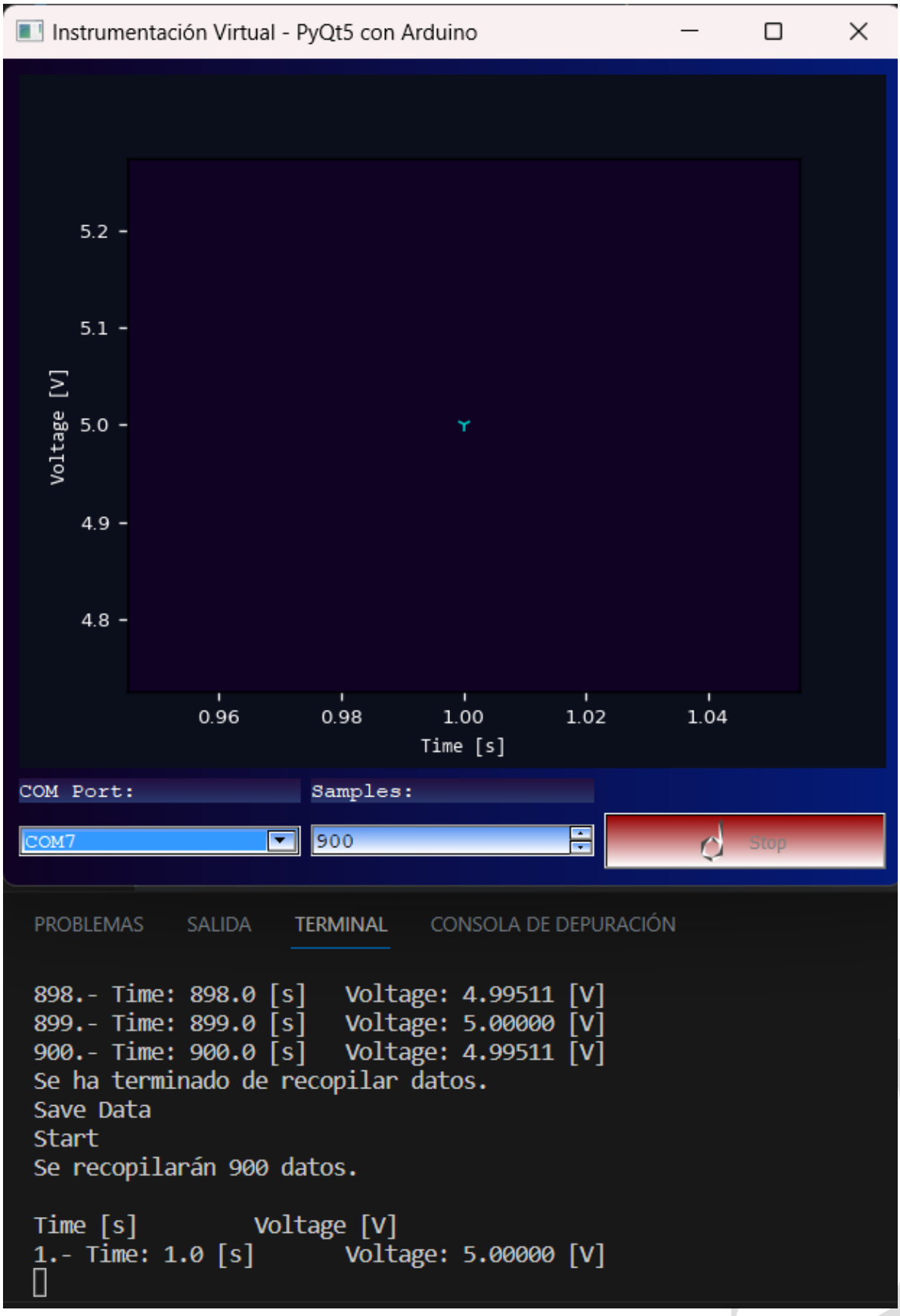

Instrumentación Virtual - PyQt5 con Arduino
Voltage [V]
Time [s]
COM Port:
Samples:
COM7
900
Start
Save
PROBLEMAS
SALIDA
TERMINAL
CONSOLA DE DEPURACIÓN
893.- Time: 893.0 [s] Voltage: 5.00000 [V]
894.- Time: 894.0 [s] Voltage: 4.99511 [V]
895.- Time: 895.0 [s] Voltage: 5.00000 [V]
896.- Time: 896.0 [s] Voltage: 4.99511 [V]
897.- Time: 897.0 [s] Voltage: 5.00000 [V]
898.- Time: 898.0 [s] Voltage: 4.99511 [V]
899.- Time: 899.0 [s] Voltage: 5.00000 [V]
900.- Time: 900.0 [s] Voltage: 4.99511 [V]
Se ha terminado de recopilar datos.
Save Data



Library	PyQt5
Time [s]	Voltage [V]
1	3.32356
2	3.30401
3	3.31378
4	3.29912
5	3.31378
6	3.30401
7	3.31378
8	3.30401
9	3.3089
10	3.30401
11	3.31378
12	3.29912
13	3.3089
14	3.29912
15	3.31378
16	3.30401
17	3.3089
18	3.30401
19	3.31378
20	3.29912
21	3.31378
22	3.30401
23	3.31378
24	3.29912
25	3.31378



874	0
875	0
876	0
877	0
878	0
879	0
880	0
881	0
882	4.99511
883	5
884	4.99511
885	5
886	4.99511
887	5
888	4.99511
889	5
890	4.99511
891	5
892	4.99511
893	5
894	4.99511
895	5
896	4.99511
897	5
898	4.99511
899	5
900	4.99511


Instrumentación Virtual - PyQt5 con Arduino
5.2
5.1
5.0
4.9
4.8
Voltage [V]
0.96
0.98
1.00
1.02
1.04
Time [s]
COM Port:
Samples:
COM7
900
Stop
PROBLEMAS
SALIDA
TERMINAL
CONSOLA DE DEPURACIÓN
898.- Time: 898.0 [s] Voltage: 4.99511 [V]
899.- Time: 899.0 [s] Voltage: 5.00000 [V]
900.- Time: 900.0 [s] Voltage: 4.99511 [V]
Se ha terminado de recopilar datos.
Save Data
Start
Se recopilarán 900 datos.
Time [s] Voltage [V]
1.- Time: 1.0 [s] Voltage: 5.00000 [V]