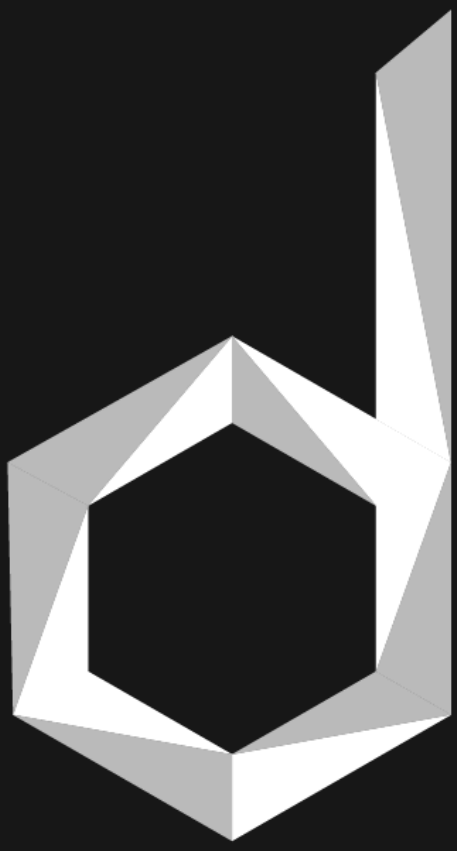
INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

INSTRUMENTACIÓN VIRTUAL

PYTHON 3.9.7, C# & LABVIEW

Graphical User Interface (GUI): wxPython

Contenido

Teoría – GUI (Graphical User Interface): wxPython

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) y se utilizará a continuación para crear las distintas partes de una GUI.

La terminología de las GUI con la librería **wxPython** es:

- **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.
- **Frame:** Es una clase perteneciente a la librería **wxPython** que sirve para crear la ventana de la GUI.
    - **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**.
    - **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.

Ejercicios – GUI:

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

1.- GUIs Básicas: Frame vacío, Frame con Panel, Frame con un Panel que contiene una imagen.

En el primer programa se crearán 3 GUIs:

1. GUI de un Frame vacío con un título que dice Hello World.
2. GUI de un Frame con un Panel que contiene un botón, tanto el Frame como el Panel tienen colores distintos para distinguirse entre sí, además incluye con un botón que cuenta todas las veces que es presionado y lo imprime en consola.
3. GUI de un Frame con un Panel 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

```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.

import wx #wxPython: Librería para crear interfaces de usuario GUI (Graphycal User Interface)

#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.

#GRAPHICAL USER INTERFACES HECHAS CON LA LIBRERÍA WX PYTHON:
#GUI VACÍO CON TÍTULO HELLO WORLD CREADO CON PROGRAMACIÓN ESTRUCTURADA:
#Instancia de la librería wxPython por medio del constructor de la clase App para crear un objeto que funcione
#como la base de un GUI.
app = wx.App(redirect = False)
#Instancia de la librería wxPython por medio del constructor de la clase Frame, frame se refiere a la ventana
#del GUI y a esta se le asigna un título por medio de su segundo parámetro.
frame = wx.Frame(None, title = "Hello World 1: Programación Estructurada")
#wx.Frame.show(): Método aplicado a una instancia de la clase Frame para mostrar la ventana de la GUI.
frame.Show()
```

```
#wx.Frame.SetSize(): Método utilizado para indicar el tamaño en pixeles del frame (ventana), este método recibe
#como parámetro un objeto de la clase Size, perteneciente a la librería wxPython:
# - wx.Size(ancho, alto): Con este atributo se indica el ancho y alto del Frame en pixeles.
frame.SetSize(wx.Size(500, 500))
#wx.App.MainLoop(): Método para que se ejecute en un loop infinito el GUI, logrando 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 frame.
app.MainLoop()




#GUI VACÍO CON TÍTULO HELLO WORLD CREADO CON PROGRAMACIÓN ORIENTADA A OBJETOS:
#myFrame: Esta clase propia recibe como parámetro un objeto de la clase Frame, que pertenece a la librería
#wxPython y representa la ventana del GUI.
class myFrame(wx.Frame):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben de a fuerza de tener un valor.
    def __init__(self):
        #Dentro del constructor de la GUI se declara una instancia de la librería wxPython por medio de la cual
        #se accede al constructor de la clase Frame, que representa la ventana del GUI, a dicha ventana se le
        #asigna un título por medio de su segundo parámetro.
        wx.Frame.__init__(self, None, title = "Hello World 2: POO")
        #wx.Frame.SetBackgroundColour() = self.SetBackgroundColour(): Método aplicado al objeto de la clase
        #Frame que recibe como parámetro el constructor de esta clase, utilizado para cambiar el color del fondo
        #de la ventana del GUI, el método realiza esto recibiendo como parámetro un objeto de la clase Colour,
        #perteneciente a la librería wxPython, indicando el color en formato RGB:
        #  - wx.Colour(R, G, B): Los valores de RGB van de 0 a 255 y su combinación de colores rojo, verde y
        #    azul crean cualquier color existente, el valor (0, 0, 0) corresponde al color negro y
        #    (255, 255, 255) al blanco.
        self.SetBackgroundColour(wx.Colour(200, 0, 0)) #Color de fondo rojo no tan brillante.
        #wx.Frame.show() = self.show(): Método aplicado al objeto de la clase Frame que recibe como parámetro el
        #constructor de esta clase para mostrar la ventana del GUI.
        self.Show()

#__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 wxPython por medio del constructor de la clase App para crear un objeto que
    #funcione como la base de un GUI.
    appPOO = wx.App(redirect = False)
    #Instancia de nuestra clase propia llamada myFrame que fue creada en este mismo programa (frame se refiere
    #a la ventana del GUI), el constructor vacío lo que hace es indicar que se cree y muestre la ventana.
```

```
    frame = myFrame()  #No se indica el título del frame porque ya fue declarado dentro de la clase myFrame.
    #Instancia_myFrame.SetPosition(): Método utilizado para indicar la posición inicial de la ventana dentro de
    #la pantalla del ordenaror, este método recibe como parámetro un objeto de la clase Point, perteneciente a
    #la librería wxPython:
    # - wx.Point(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.
    frame.SetPosition(wx.Point(0, 0))
    #Instancia_myFrame.SetSize(): Método utilizado para indicar el tamaño incial en pixeles del frame (ventana),
    #este método recibe como parámetro un objeto de la clase Size, perteneciente a la librería wxPython:
    # - wx.Size(ancho, alto): Con este atributo se indica el ancho y alto del Frame en pixeles.
    frame.SetSize(wx.Size(300, 300))
    #wx.App.MainLoop(): Método para que se ejecute en un loop infinito el GUI, logrando que no se ejecute una
    #vez y luego cierre por sí solo, sino que solo se cierre al dar clic en el tache del frame.
    appPOO.MainLoop()




#GUI CREADO CON PROGRAMACIÓN ORIENTADA A OBJETOS QUE INCLUYE UN PANEL (CONTENEDOR DE ELEMENTOS) CON UN BOTÓN:
#Ahora vamos a agregar un panel, que es un contenedor donde se pueden incluir varios elementos (widgets) como
#botones, cuadros de texto, imágenes, etc. dentro de un marco (frame).

#Cuando se genere un GUI que incluya un panel se debe crear el código en el siguiente órden:
# - clase Panel: El contenedor incluye un objeto que instancíe la clase de cada widget que se quiera incluir.
#       - Widget: Dentro del constructor de la clase Panel se crea un objeto de cada widget que se quiera
#         incluir en el contenedor, pero si es que alguno de estos elementos realiza una acción, fuera del
#         constructor pero dentro de la clase Panel, se debe crear una función que describa lo que realiza.
#         Los widgets que realizan acciones pueden ser: botones, checkboxes, áreas de texto, comboboxes,
#         radiobuttons, listboxes, ventanas de diálogo, etc.
# - clase Frame: Dentro de la clase frame se declara su título y se instancía la clase panel para agregar el
#   contenedor previamente creado a la ventana.
# - método main: A través del método main se ejecuta la clase frame para mostrar y ejecutar la ventana del GUI.

#myPanel: La clase recibe como parámetro un objeto de la clase Panel, que pertenece a la librería wxPython y
#representa un contenedor.
class MyPanel (wx.Panel):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben de a fuerza de tener un valor.
    def __init__(self, parent):
        #super().__init__(parent): Lo que hace el método super() es heredar todos los métodos y atributos de la
        #clase padre. En este caso es necesario incluirlo porque el constructor de la clase Panel recibe como
        #parámetro al elemento parent.
```

```
        super().__init__(parent)
        #Esta es una práctica rara pero es la única manera que se encontró de cambiar el tamaño del Panel, para
        #ello se debe crear una función interna en el constructor,donde dentro simplemente se usa el método
        #setSize() y luego se utiliza el método wx.CallAfter() para llamar la función cambiaTamañoPanel() en un
        #momento posterior.
        def cambiaTamañoPanel():
            #wx.Panel.SetSize() = self.SetSize(): Método utilizado para indicar el tamaño en pixeles del panel
            #(contenedor), este método recibe como parámetro un objeto de la clase Size, perteneciente a la
            #librería wxPython:
            # - wx.Size(ancho, alto): Con este atributo se indica el ancho y alto del Frame en pixeles.
            self.SetSize(wx.Size(100, 100))
        #wx.CallAfter(): Este método se utiliza comúnmente en entornos de GUI para programar llamadas a
        #las funciones que actualizan la interfaz de usuario de manera segura, el problema con usar esto es que
        #hace muy lento el proceso de cerrar la GUI, por lo cual no es de muy buenas prácticas utilizar este
        #método, ahora solo se utilizó para mostrar la diferencia entre en Panel (contenedor) y Frame (ventasa).
        #Para lograr que el programa cierre más rápido se debe seleccionar la consola y dar clic en las teclas:
        #   CTRL + C:   Al darse clic sobre la consola, forza que el programa termine su ejecución.
        #   cls:        Comando usado para que se borre de la consola el historial de ejecuciones.
        wx.CallAfter(cambiaTamañoPanel)
        #wx.Panel.SetBackgroundColour() = self.SetBackgroundColour(): Método aplicado al objeto de la clase
        #Panel que recibe como parámetro el constructor de esta clase para cambiar el color del fondo del
        #contenedor, el método realiza esto recibiendo un parámetro que instancíe la clase Colour de la librería
        #wxPython, indicando el color en formato RGB:
        #  - wx.Colour(R, G, B): Los valores de RGB van de 0 a 255 y su combinación de colores rojo, verde y
        #    azul crean cualquier color existente, el valor (0, 0, 0) corresponde al color negro y
        #    (255, 255, 255) al blanco.
        self.SetBackgroundColour(wx.Colour(0, 255, 0))

        #CREACIÓN DE LOS WIDGETS: Botón
        #Instancia de la librería wxPython por medio del constructor de la clase Button para crear un widget de
        #tipo botón, en este se debe indicar como parámetro el texto que aparece sobre él y su posición.
        # - label = "": Con este parámetro se indica el texto que aparecerá sobre el botón.
        # - pos = (x, y): Con este atributo se indica la posición fija en pixeles del widget, 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.
        #       - Es importante mencionar que si después se utiliza la clase BoxSizer para posicionar los
        #         elementos de forma relativa, esta posición no es respetada.
        button = wx.Button(self, label = "Press Me", pos = (0, 0))

        #El siguiente código crea una instancia de la clase BoxSizer, la cual permite un posicionamiento
        #relativo, colocando así un elemento respecto a otro, para poder usar esta clase el primer objeto se
        #debe encontrar dentro del segundo. Si no se usa la clase BoxSizer, los elementos se colocarán unos
        #encima de los otros.
        #Al crear la Instancia de la clase BoxSizer perteneciente a la librería wxPython se le puede pasar como
        #parámetro solamente dos posibles atributos para indicar la dirección de la posición del objeto:
```

```
# - wx.HORIZONTAL: Hace que la dirección de la alineación del primer objeto sea horizontal respecto al
#   segundo, esto se refiere a que se empiece indicar la posición del widget desde la esquina izquierda
#   dentro del contenedor.
# - wx.VERTICAL: Hace que la dirección de la alineación del primer objeto sea vertical respecto al
#   segundo, esto se refiere a que se empiece indicar la posición del widget desde la parte superior de
#   en medio, dentro del contenedor.
main_sizer = wx.BoxSizer(wx.HORIZONTAL)

#Ya se mencionó que la clase BoxSizer sirve al posicionar un elemento respecto a otro, para ello uno de
#los elementos se debe encontrar dentro del otro, con el objetivo de indicar cuál es el primer objeto
#(el que tiene posicionamiento relativo) y cuál es el segundo objeto (el que contiene al primer objeto),
#se utilizan los métodos .Add() y .SetSizer() de la siguiente forma:

#Instancia_BoxSizer.Add(): Método utilizado para agregar un elemento al sizer, sizer se refiere al
#elemento que contiene a otro que está colocado dentro de él con posicionamiento relativo, para ello
#dentro de su paréntesis se agregan los siguientes parámetros:
# - primer_parámetro: Con este parámetro se indica qué objeto que será agregado dentro del otro, el
#   objeto contenedor es llamado sizer.
# - proportion: Este parámetro determina cómo se asignará el espacio de todos los elementos que se
#   encuentren dentro del contenedor, aún si el sizer se expande o se reduce.
#           - proportion = 0: El elemento no crecerá ni se encogerá en relación a otros elementos
#             dentro del sizer.
#           - proportion = valor: El elemento se expandirá o se encogerá proporcionalmente en el sizer,
#             dependiendo del valor de los demás elementos.
#                   Por ejemplo, si hay dos elementos en el sizer, ambos con proportion = 1, cada uno
#                   ocupará la mitad del espacio disponible cuando el sizer se expanda. Si uno de los
#                   elementos tiene proportion = 2, ocupará dos tercios del espacio disponible, mientras
#                   que el otro elemento ocupará un tercio cuando el sizer se expanda.
# - flag: El parámetro flag se utiliza para especificar las opciones de posicionamiento y alineación
#   del elemento dentro del sizer por medio de banderas, las acciones de estas banderas se pueden
#   combinar usando la operación lógica OR (|), las flags que se pueden usar son descritas a
#   continuación:
#           - wx.EXPAND: Hace que el elemento se expanda para ocupar todo el espacio disponible en la
#             dirección del sizer.
#           - wx.ALL: Agrega un borde invisible en todos los lados del elemento.
#           - wx.LEFT: Agrega un borde invisible en el lado izquierdo del elemento.
#           - wx.RIGHT: Agrega un borde invisible en el lado derecho del elemento.
#           - wx.TOP: Agrega un borde invisible en la parte superior del elemento.
#           - wx.BOTTOM: Agrega un borde invisible en la parte inferior del elemento.
#           - wx.CENTER: Centra el elemento dentro del espacio asignado por el sizer.
#           - wx.ALIGN_LEFT: Alinea el elemento a la izquierda dentro del espacio asignado por el sizer.
#           - wx.ALIGN_RIGHT: Alinea el elemento a la derecha dentro del espacio asignado por el sizer.
#           - wx.ALIGN_TOP: Alinea el elemento en la parte superior dentro del espacio asignado por el
#             sizer.
#           - wx.ALIGN_BOTTOM: Alinea el elemento en la parte inferior dentro del espacio asignado por
```

```
        #             el sizer.
        # - border: Establece el espacio en píxeles entre el elemento y los bordes del sizer.
        main_sizer.Add(button, proportion = 1,              #Botón agregado al main_sizer, proportion = 1.
                               flag = wx.ALL | wx.CENTER,   #Borde en todos los lados del elemento y Centrado.
                               border = 0,)                 #Borde de 0 pixeles.
        #wx.Panel.SetSizer() = self.SetSizer(): Método aplicado al objeto de la clase Panel que recibe como
        #parámetro esta clase, el cuál recibe como parámetro un objeto de la clase BoxSizer para indicar cuál es
        #el elemento contenedor al que ya se han agregado anteriormente uno o más widgets posicionados
        #relativamente con el método .Add().
        self.SetSizer(main_sizer)

        #ACCIONES DEL BOTÓN: Para ejecutar la acción de un botón en Python, dentro del constructor de la clase
        #Panel se debe mandar a llamar una función (método) que describa la acción a ejecutar, esta función se
        #debe encontrar fuera del constructor pero pertenecer igualmente a la clase Panel.

        #ATRIBUTOS DEL CONSTRUCTOR DE LA CLASE PANEL: Dentro del constructor se deben declarar las variables con
        #las que interactúen los botones para realizar acciones dentro del GUI.
        self.count = 0     #Atributo de la clase Panel para contar cuantas veces se ha presionado un botón.

        #Instancia_Button.Bind(): Este método se utiliza para enlazar un evento a un controlador de eventos,
        #indicando en su primer parámetro el evento que detona el método y en el segundo la función que
        #se ejecutará cuando ese evento ocurra.
        #Las funciones que describen las acciones a realizar por los widgets del Panel se encuentran dentro de
        #esta misma clase, pero fuera de su constructor.
        # - Tipos de Eventos en Python:
        #       - wx.EVT_BUTTON: Evento que se activa cuando se hace clic en un botón.
        #       - wx.EVT_TEXT: Evento que se activa cuando se cambia el contenido de un control de texto.
        #       - wx.EVT_CHECKBOX: Evento que se activa cuando se cambia el estado de una casilla de
        #         verificación.
        #       - wx.EVT_COMBOBOX: Evento que se activa cuando se selecciona un elemento de una lista
        #         desplegable (combobox).
        #       - wx.EVT_LISTBOX: Evento que se activa cuando se selecciona un elemento de una lista (listbox).
        #       - wx.EVT_RADIOBUTTON: Evento que se activa cuando se selecciona un botón de opción (radiobutton)
        #         en un grupo de botones de opción.
        #       - wx.EVT_MENU: Evento que se activa cuando se selecciona una opción de menú.
        #       - wx.EVT_CLOSE: Evento que se activa cuando se intenta cerrar una ventana o diálogo.
        #       - wx.EVT_KEY_DOWN y wx.EVT_KEY_UP: Eventos que se activan cuando se presiona o se suelta una
        #         tecla del teclado, respectivamente.
        #       - wx.EVT_MOUSE_EVENTS: Son una serie de eventos relacionados con las interacciones del ratón,
        #         como clics, movimiento, etc. Estos eventos son descritos a continuación:
        #               - wx.EVT_LEFT_DOWN: Se activa cuando se presiona el botón izquierdo del ratón.
        #               - wx.EVT_LEFT_UP: Se activa cuando se suelta el botón izquierdo del ratón.
        #               - wx.EVT_LEFT_DCLICK: Se activa cuando se hace doble clic con el botón izquierdo del
        #                 ratón.
        #               - wx.EVT_RIGHT_DOWN: Se activa cuando se presiona el botón derecho del ratón.
```

```
        #                - wx.EVT_RIGHT_UP: Se activa cuando se suelta el botón derecho del ratón.
        #                - wx.EVT_RIGHT_DCLICK: Se activa cuando se hace doble clic con el botón derecho del
        #                  ratón.
        #                - wx.EVT_MIDDLE_DOWN: Se activa cuando se presiona el botón central del ratón.
        #                - wx.EVT_MIDDLE_UP: Se activa cuando se suelta el botón central del ratón.
        #                - wx.EVT_MIDDLE_DCLICK: Se activa cuando se hace doble clic con el botón central del
        #                  ratón.
        #                - wx.EVT_MOTION: Se activa cuando se mueve el ratón dentro del área del objeto
        #                  capturador.
        #                - wx.EVT_ENTER_WINDOW: Se activa cuando el ratón entra en el área del objeto
        #                  capturador.
        #                - wx.EVT_LEAVE_WINDOW: Se activa cuando el ratón sale del área del objeto capturador.
        #                - wx.EVT_MOUSEWHEEL: Se activa cuando se desplaza la rueda del ratón.
        #Evento de clic en botón, ejecutado por la función on_button_press de esta clase Panel.
        button.Bind(wx.EVT_BUTTON, self.on_button_press)


    #función on_button_press(): Método creado dentro de la clase propia llamada myPanel 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 un botón y debe contar cuántas veces se ha dado
    #clic en dicho botón.
    def on_button_press(self, event):
        #BARIABLES DE LA FUNCIÓN:
        #La variable count es un atributo declarado en el constructor de la clase Panel, por eso se accede a
        #través de la palabra reservada self.
        self.count = self.count + 1
        #print(): Imprimir un mensaje en consola.
        print("Presionase el botón")
        print(self.count)


#Frame_With_Panel: Esta clase propia recibe como parámetro un objeto de la clase Frame, que pertenece a la
#librería wxPython y representa la ventana del GUI.
#Dentro de ella se crea una instancia de la clase MyPanel para agregar dentro de la ventana del GUI un
#contenedor que incluye widgets (botones, áreas de texto, áreas de imagen, etc.) dentro.
class Frame_With_Panel(wx.Frame):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben de a fuerza de tener un valor.
    def __init__(self):
        #Dentro del constructor de la GUI se declara una instancia de la librería wxPython por medio de la cual
        #se accede al constructor de la clase Frame, que representa la ventana del GUI, a dicha ventana se le
        #asigna un título por medio de su segundo parámetro.
        super().__init__(None, title = "Hello World 3 - POO: Panel con Botón")
        #Instancia de la clase MyPanel para agregar el panel al Frame, osea el contenedor de elementos a la
        #ventana de la GUI, se le pasa como parámetro la instancia del objeto Frame que recibe esta misma clase
```

```
        #como parámetro, por eso se usa la palabra reservada self.
        panel = MyPanel(self)
        #wx.Frame.SetBackgroundColour() = self.SetBackgroundColour(): Método aplicado al objeto de la clase
        #Frame que recibe como parámetro el constructor de esta clase, utilizado para cambiar el color del fondo
        #de la ventana del GUI, el método realiza esto recibiendo como parámetro un objeto de la clase Colour,
        #perteneciente a la librería wxPython, indicando el color en formato RGB:
        #  - wx.Colour(R, G, B): Los valores de RGB van de 0 a 255 y su combinación de colores rojo, verde y
        #    azul crean cualquier color existente, el valor (0, 0, 0) corresponde al color negro y
        #    (255, 255, 255) al blanco.
        self.SetBackgroundColour(wx.Colour(0, 0, 200)) #Color de fondo azul no tan brillante.
        #wx.Frame.show() = self.show(): Método aplicado al objeto de la clase Frame que recibe como
        #parámetro el constructor de esta clase para mostrar la ventana del GUI.
        self.Show()


#__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 wxPython por medio del constructor de la clase App para crear un objeto que
    #funcione como la base de un GUI.
    app2 = wx.App(redirect = False)
    #Instancia de nuestra clase propia llamada Frame_With_Panel que fue creada en este mismo programa (frame se
    #refiere a la ventana del GUI) e incluye una instancia de la clase Panel para agregar un contenedor con
    #elementos dentro, el constructor vacío lo que hace es indicar que se cree y muestre la ventana.
    frame2 = Frame_With_Panel()
    #wx.App.MainLoop(): Método para que se ejecute en un loop infinito el GUI, logrando que no se ejecute una
    #vez y luego cierre por sí solo, sino que solo se cierre al dar clic en el tache del frame.
    app2.MainLoop()




#GUI CREADO CON PROGRAMACIÓN ORIENTADA A OBJETOS QUE INCLUYE UN PANEL (CONTENEDOR DE ELEMENTOS) CON UNA IMAGEN:
#vamos a agregar un panel, que es un contenedor donde se pueden incluir varios elementos (widgets) como
#botones, cuadros de texto, imágenes, etc. dentro de un marco (frame).

#Cuando se genere un GUI que incluya un panel se debe crear el código en el siguiente órden:
# - clase Panel: El contenedor incluye un objeto que instancíe la clase de cada widget que se quiera incluir.
#       - Widget: Dentro del constructor de la clase Panel se crea un objeto de cada widget que se quiera
#         incluir en el contenedor, pero si es que alguno de estos elementos realiza una acción, fuera del
#         constructor pero dentro de la clase Panel, se debe crear una función que describa lo que realiza.
#         Los widgets que realizan acciones pueden ser: botones, checkboxes, áreas de texto, comboboxes,
```

```
#         radiobuttons, listboxes, ventanas de diálogo, etc.
# - clase Frame: Dentro de la clase frame se declara su título y se instancía la clase panel para agregar el
#   contenedor previamente creado a la ventana.
# - método main: A través del método main se ejecuta la clase frame para mostrar y ejecutar la ventana del GUI.

#ImagePanel: La clase recibe como parámetro un objeto de la clase Panel, que pertenece a la librería wxPython y
#representa un contenedor.
class ImagePanel(wx.Panel):
    #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.
    #El parámetro image_size se declaró en el constructor de la clase para que cuando se cree un objeto del
    #panel en la clase Frame, se le tenga que indicar el tamaño del widget donde aparecerá la imagen, que no es
    #igual al tamaño de la imagen misma, solo al de la ventana donde aparece.
    def __init__(self, parent, image_size):
        #super().__init__(parent): Lo que hace el método super() es heredar todos los métodos y atributos de la
        #clase padre. En este caso es necesario incluirlo porque el constructor de la clase Panel recibe como
        #parámetro al elemento parent.
        super().__init__(parent)

        #CREACIÓN DE LOS WIDGETS: Imágen
        #Las clases wx.Image y wx.Bitmap están relacionadas y se utilizan en conjunto para trabajar con
        #imágenes en la interfaz gráfica.
        #   wx.Image: 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.
        #
        #   wx.Bitmap: Es una representación de imagen que puede ser utilizada directamente en los controles y
        #   widgets de la interfaz gráfica de wxPython.

        #Cargar una Imagen: Se crea una instancia de la librería wxPython por medio del constructor de la clase
        #Image para cargar una imagen en memoria con datos en bruto.
        # - filepath (str): 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.
        # - type (int): Indica el tipo de imagen a cargar. Puede ser uno de los siguientes valores predefinidos:
        #           - wx.BITMAP_TYPE_BMP: Bitmap de Windows.
        #           - wx.BITMAP_TYPE_JPEG: JPEG.
        #           - wx.BITMAP_TYPE_GIF: GIF.
        #           - wx.BITMAP_TYPE_PNG: PNG.
        #           - wx.BITMAP_TYPE_TIFF: TIFF.
        #           - wx.BITMAP_TYPE_PNM: PNM (Portable anymap).
        #           - wx.BITMAP_TYPE_PCX: PCX (ZSoft IBM PC Paintbrush file).
        #           - wx.BITMAP_TYPE_TGA: TGA (Truevision TGA file).
        #           - wx.BITMAP_TYPE_ICO: ICO (Icon file).
```

```
        #           - wx.BITMAP_TYPE_CUR: CUR (Cursor file).
        #           - wx.BITMAP_TYPE_XBM: XBM (X11 Bitmap).
        #           - wx.BITMAP_TYPE_XPM: XPM (X11 Pixmap).
        #           - wx.BITMAP_TYPE_WEBP: WEBP (WebP image format).
        #           - wx.BITMAP_TYPE_ANY: Cualquier tipo de imagen compatible.
        # - index (int): Este método se utiliza cuando se carga una imagen que contiene varias subimágenes
        #   (como un archivo ICO o un archivo animado GIF). El valor predeterminado es -1 para cargar la imagen
        #   principal.
        # - data (str o bytes): Datos en bruto de la imagen en formato binario. Se utiliza cuando la imagen no
        #   se carga desde un archivo, sino desde una fuente de datos en memoria.
        # - mask (wx.Image o None): Representa una máscara opcional que se utilizará para darle transparencia
        #   a la imagen. La máscara debe tener el mismo tamaño que la imagen principal y esta especifica cuáles
        #   píxeles están completamente transparentes y cuáles son opacos.
        # - size (tuple): Parámetro opcional que especifica el tamaño deseado para la imagen cargada. Se utiliza
        #   para redimensionar la imagen a un tamaño específico. El tamaño debe ser una tupla de dos elementos
        #   en forma de (ancho, alto).
        #       - Desempaquetar (*): Si es que se quiere indicar el tamaño de una imagen perteneciente a un GUI
        #         fuera de la clase donde fue creada se debe usar el operador * el cual se utiliza en Python
        #         para desempaquetar una secuencia (como una lista, tupla o conjunto) en sus elementos
        #         individuales.
        #         Esto significa que los elementos de la secuencia se extraen en el órden en el que fueron
        #         declarados, se pasan y usan como argumentos separados a una función o constructor que espera
        #         múltiples argumentos.

        #Para indicar el filepath normalmente se utiliza una variable aparte que guarde el directorio y nombre
        #del archivo deseado, donde se deben reemplazar los guiones\ por /:
        #Para leer una imagen con un objeto wx.Image se debe utilizar la dirección absoluta del archivo:
        # - 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.
        image_file = "C:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación
Virtual Aplicada/Img/Iron Man Bullet.jpg"
        img = wx.Image(image_file, wx.BITMAP_TYPE_ANY)

        #Una vez que se haya cargado la imagen y asignado el tamaño del widget donde aparece por medio del
        #constructor, se ejecuta una función declarada dentro de la clase ImagePanel llamada loadImage(), que
        #sirve para ajustar el tamaño de la imagen cargada antes de asignarla al widget y que aparezca en el
        #GUI.
        img = self.scale_image(img, image_size)
```

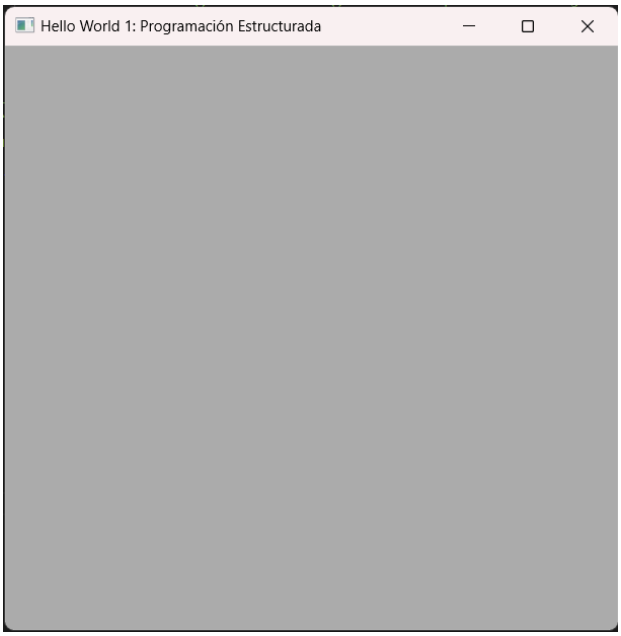
```
        #Crear un Bitmap: Una vez que se tiene una instancia de la clase wx.Image, se puede crear un objeto
        #Bitmap, esto se puede realizar de dos formas:
        #   - wx.Bitmap(wx.Image): El constructor de la clase Bitmap recibe como parámetro un objeto que
        #     instancíe la clase Image, ambos pertenecientes a la librería de wxPython.
        #   - wx.Image.ConvertToBitmap(): Método utilizado para convertir un objeto Image en uno Bitmap.
        #Esto convierte la imagen en un formato adecuado para que pueda ser utilizado en controles como
        #wx.StaticBitmap, wx.Button, wx.BitmapButton, entre otros.
        img_widget_GUI = img.ConvertToBitmap()
        #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 wx.StaticBitmap, que es parte de la biblioteca wxPython y se utiliza para
        #mostrar una imagen estática en un panel. Los parámetros que puede recibir la clase wx.StaticBitmap son
        #los siguientes:
        # - parent: Es el objeto padre al que se añadirá el widget wx.StaticBitmap.
        # - id: Un identificador único para el widget. Puede ser de tipo entero o wx.ID_ANY para permitir que
        #   wxPython asigne automáticamente un ID.
        # - bitmap: El objeto wx.Bitmap que se utilizará como imagen para el wx.StaticBitmap.
        # - pos: Una tupla (x, y) o un objeto wx.Point que indica la posición inicial del widget. Si no se
        #   proporciona, se utilizará la posición predeterminada.
        # - size: Una tupla (width, height) o un objeto wx.Size que indica el tamaño del widget. Si no se
        #   proporciona, se utilizará el tamaño predeterminado.
        # - style: Un estilo adicional para el widget wx.StaticBitmap. Puede incluir combinaciones de banderas:
        #           - wx.ALIGN_LEFT: Alinea el contenido del widget a la izquierda.
        #           - wx.ALIGN_RIGHT: Alinea el contenido del widget a la derecha.
        #           - wx.ALIGN_CENTER_HORIZONTAL: Centra horizontalmente el contenido del widget.
        #           - wx.ALIGN_TOP: Alinea el contenido del widget en la parte superior.
        #           - wx.ALIGN_BOTTOM: Alinea el contenido del widget en la parte inferior.
        #           - wx.ALIGN_CENTER_VERTICAL: Centra verticalmente el contenido del widget.
        #           - wx.ST_NO_AUTORESIZE: Evita que el widget se redimensione automáticamente cuando cambia el
        #             tamaño de su contenido.
        #   Además de estos estilos, hay otros disponibles que permiten modificar aspectos estéticos, se pueden
        #   combinar varios estilos utilizando el operador OR (|) para crear la combinación deseada:
        #           - wx.BORDER_NONE: No muestra ningún borde alrededor del widget.
        #           - wx.BORDER_SIMPLE: Muestra un borde simple alrededor del widget.
        #           - wx.BORDER_RAISED: Muestra un borde elevado alrededor del widget.
        #           - wx.BORDER_SUNKEN: Muestra un borde hundido alrededor del widget.
        #           - wx.BORDER_STATIC: Muestra un borde estático alrededor del widget.
        # - name: El nombre del widget.
        #Es importante destacar que los parámetros pos, size, style y name son opcionales y tienen valores
        #predeterminados si no se proporcionan, el que si es necesario indicarlo es el objeto bitmap.
        #Se declaran como self.nombreObjeto los widgets a los que sí se les vaya a extraer o introducir datos
        #en el transcurso del funcionamiento de la interfaz gráfica, por eso el widget de Bitmap si se declara como
        #self y el widget de Image no se declara como self.
        self.image_ctrl = wx.StaticBitmap(parent = self, bitmap = img_widget_GUI)


    #función scale_image(): Método creado dentro de la clase propia llamada ImagePanel que recibe como
```

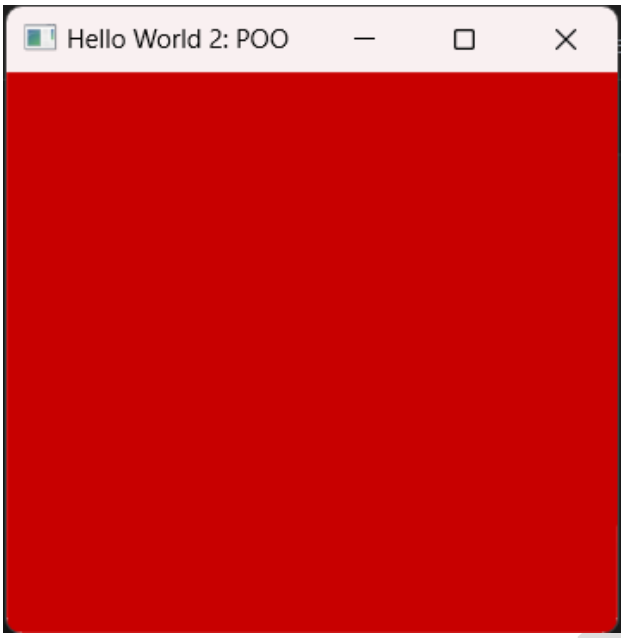
```
#parámetro a la imagen cargada con el objeto wx.Image y el tamaño del widget donde aparecerá dicha imagen,
#para después ajustar el tamaño de la imagen y que se vea completa en el widget del GUI.
def scale_image(self, img, size):
    #De la variable size, la cual es una tupla de dos elementos en forma de (ancho, alto) que se le pasa
    #como parámetro al constructor de esta clase y representa el tamaño del widget que mostrará la imagen,
    #se obtiene y separa en dos variables distintas su ancho y alto para posteriormente ajustar el tamaño
    #del objeto Image, que es la imagen cargada y representada en datos brutos.
    widget_width, widget_height = size  #Ancho y Alto en pixeles del widget.
    #wx.Image.GetWidth(): Método para obtener el ancho en pixeles de una imagen.
    img_width = img.GetWidth()          #Ancho en pixeles de la imagen en datos brutos.
    #wx.Image.GetHeight(): Método para obtener la altura en pixeles de una imagen.
    img_height = img.GetHeight()        #Alto en pixeles de la imagen en datos brutos.

    #Si el ancho de la imagen es mayor que el ancho del widget, entonces se crearán nuevos valores para su
    #ancho y alto, logrando así que se ajuste su tamaño al del widget, manteniendo su forma original:
    if img_width > widget_width:
        #El nuevo ancho de la imagen es igual al ancho del widget
        new_width = widget_width
        #La nueva altura de la imagen se obtiene al realizar una regla de 3 donde se busca obtener la altura
        #proporcional del widget, conociendo ya la altura y el ancho de la imagen original y el ancho del
        #widget al que se quiere llegar, por lo tanto la fórmula de la regla de 3 sería la siguiente:
        #   img_width   ->  widget_width
        #   img_height  ->  widget_height
        #Por lo cual, para obtener el widget_height y asignarlo a el nuevo ancho se utiliza la siguiente
        #fórmula: widget_height = new_height = img_height*(widget_width/img_width)
        #El doble signo de "//" se utiliza en Python para realizar una división entera o "floor division",
        #a diferencia del operador de división normal ("/"), que devuelve un número decimal, el operador
        #"//" devuelve el cociente entero de la división, redondeando hacia abajo al número entero más
        #cercano.
        #No se debe poner paréntesis en la operación matemática porque sino se puede confundir el programa
        #y realizar mal la operación, ya que si primero realiza la división y obtiene un número decimal,
        #esto se redondearía a cero y la operación daría un resultado erróneo.
        new_height = img_height * (widget_width // img_width)
        print("Operación Errónea // y ():\t\t", new_height, "=", img_height, "*(", widget_width, "//", img_width, ")")
        new_height = img_height * widget_width // img_width
        print("Operación Correcta //:\t\t\t", new_height, "=", img_height, "*", widget_width, "//", img_width)
        new_height = int(img_height * (widget_width / img_width))
        print("Operación Correcta método int() y /:\t", new_height, "= int(", img_height, "*(", widget_width, "/",
         img_width, "))")

        #wx.Image.Scale(): Método para redimensionar una imagen. Este método ajusta la imagen al tamaño
        #especificado y devuelve una nueva instancia del objeto wx.Image con las dimensiones modificadas.
        img = img.Scale(new_width, new_height)
    #Si la altura de la imagen es mayor que la altura del widget, entonces se crearán nuevos valores para su
    #ancho y alto, logrando así que se ajuste su tamaño al del widget, manteniendo su forma original:
```

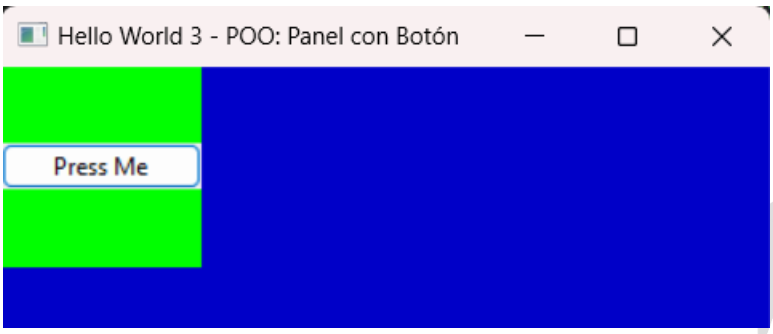
```
        elif img_height > widget_height:
            #La nueva altura de la imagen es igual a la altura del widget
            new_height = widget_height
            #El nuevo ancho de la imagen se obtiene a través de una regla de 3, muy similar a como se realizó en
            #el condicional de arriba.
            new_width = img_width * widget_height // img_height

            #wx.Image.Scale(): Método para redimensionar una imagen. Este método ajusta la imagen al tamaño
            #especificado y devuelve una nueva instancia del objeto wx.Image con las dimensiones modificadas.
            img = img.Scale(new_width, new_height)
        #Al terminar la función se devuelve la nueva img redimensionada para que esa sea puesta en el Bitmap.
        return img

#Frame_With_Panel: La clase hereda de la clase Frame que pertenece a la librería wxPython, representa la ventana
#del GUI y crea una instancia de la clase MyPanel para agregar dentro de la ventana un contenedor con elementos
#dentro.
class Frame_With_ImagePanel(wx.Frame):
    #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):
        #Dentro del constructor de la GUI se declara una instancia de la librería wxPython por medio de la cual
        #se accede al constructor de la clase Frame, que representa la ventana del GUI, a dicha ventana se le
        #asigna un título por medio de su segundo parámetro.
        super().__init__(None, title="Hello World 4 - POO: Imagen en GUI")

        #ATRIBUTOS DEL CONSTRUCTOR DE LA CLASE FRAME: Dentro del constructor se deben declarar las variables con
        #las que interactúen los elementos dentro del Frame.
        #El tamaño de la ventana (widget) donde se muestra la imagen se declara como un atributo propio de la
        #clase Frame_With_ImagePanel llamado img_widget_size, esto indicará el ancho y alto del widget, pero
        #recordemos que como la imagen se reajusta para caber aquí, puede que al final no se vea como un
        #cuadrado y solo se respete el ancho o alto de la ventana.
        self.img_widget_size = 500      #Tamaño de la ventana de imagen, dado en pixeles = (ancho, alto)

        #Instancia de la clase ImagePanel para agregar el panel al Frame, osea el contenedor de elementos
        #a la ventana de la GUI, se le pasa como parámetro a la instancia del objeto Frame que recibe esta misma
        #clase como parámetro, por eso se usa la palabra reservada self.
        #Además se le indica el tamaño que tendrá el área donde aparecerá la imagen en la ventana con el
        #parámetro image_size, que fue declarado como un parámetro del constructor.
        panel3 = ImagePanel(self, image_size=(self.img_widget_size, self.img_widget_size))

        #wx.Frame.show() = self.show(): Método aplicado al objeto de la clase Frame que recibe como
        #parámetro el constructor de esta clase para mostrar la ventana del GUI.
        self.Show()

#__name__ == __main__: Método main, esta función es super importante ya que sirve para instanciar las clases del
```

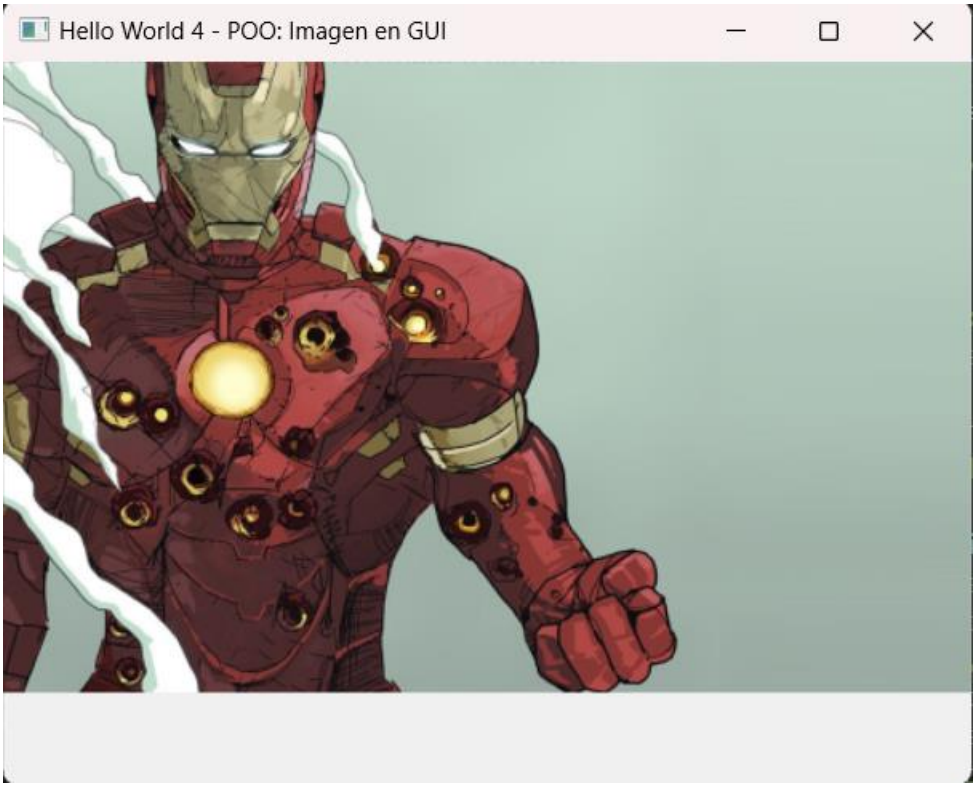
```python
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.
if __name__ == '__main__':
    #Instancia de la librería wxPython por medio del constructor de la clase App para crear un objeto que
    #funcione como la base de un GUI.
    app3 = wx.App(redirect=False)
    #Instancia de nuestra clase propia llamada Frame_With_Panel que fue creada en este mismo programa (frame se
    #refiere a la ventana del GUI) e incluye una instancia de la clase Panel para agregar un contenedor con
    #elementos dentro, el constructor vacío lo que hace es indicar que se cree y muestre la ventana.
    frame3 = Frame_With_ImagePanel()
    #Instancia_myFrame.SetSize(): Método utilizado para indicar el tamaño incial en pixeles del frame (ventana),
    #este método recibe como parámetro un objeto de la clase Size, perteneciente a la librería wxPython:
    # - wx.Size(ancho, alto): Con este atributo se indica el ancho y alto del Frame en pixeles.
    frame3.SetSize(wx.Size(500, 400))
    #wx.App.MainLoop(): Método para que se ejecute en un loop infinito el GUI, logrando que no se ejecute una
    #vez y luego cierre por sí solo, sino que solo se cierre al dar clic en el tache del frame.
    app3.MainLoop()
```

Resultado del Código Python









2.- GUI con Botón de Load Image: Interfaz gráfica que interactúa con el explorador de archivos.

En este programa se creará una sola GUI que permita cargar una imagen usando un botón que abra el explorador de archivos, para luego ajustar esa imagen al tamaño de la GUI y también indicar en un control de texto el directorio de donde fue extraída la imagen.

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.

import wx #wxPython: Librería para crear interfaces de usuario GUI (Graphycal User Interface)

#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.

#GUI CREADO CON PROGRAMACIÓN ORIENTADA A OBJETOS Y LA LIBRERÍA WX PYTHON QUE INCLUYE UN PANEL (CONTENEDOR DE
#ELEMENTOS) CON UN BOTÓN PARA CARGAR UNA IMÁGEN DEL EXPLORADOR DE ARCHIVOS Y MUESTRE SU RUTA EN UNA CAJA DE
#TEXTO: Vamos a agregar un panel, que es un contenedor donde se pueden incluir varios elementos (widgets) como
#botones, cuadros de texto, imágenes, etc. dentro de un marco (frame).

#Cuando se genere un GUI que incluya un panel (contenedor) se debe crear el código en el siguiente órden:
# - clase Panel: El contenedor incluye un objeto que instancíe la clase de cada widget que se quiera incluir.
#       - Widget: Dentro del constructor de la clase Panel se crea un objeto de cada widget que se quiera
#         incluir en el contenedor, pero si es que alguno de estos elementos realiza una acción, fuera del
#         constructor pero dentro de la clase Panel, se debe crear una función que describa lo que realiza.
#         Los widgets que realizan acciones pueden ser: botones, checkboxes, áreas de texto, comboboxes,
#         radiobuttons, listboxes, ventanas de diálogo, etc.
# - clase Frame: Dentro de la clase frame se declara su título y se instancía la clase panel para agregar el
#   contenedor previamente creado a la ventana.
# - método main: A través del método main se ejecuta la clase frame para mostrar y ejecutar la ventana del GUI.

#ImageBrowsePanel: La clase recibe como parámetro un objeto de la clase Panel, que pertenece a la librería
#wxPython y representa un contenedor.
class ImageBrowsePanel(wx.Panel):
    #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.
    #El parámetro image_size se declaró en el constructor de la clase para que cuando se cree un objeto del
    #panel en la clase Frame, se le tenga que indicar el tamaño del widget donde aparecerá la imagen, que no es
    #igual al tamaño de la imagen misma, solo al de la ventana donde aparece.
    def __init__(self, parent, image_size):
        #super().__init__(parent): Lo que hace el método super() es heredar todos los métodos y atributos de la
        #clase padre. En este caso es necesario incluirlo porque el constructor de la clase Panel recibe como
        #parámetro al elemento parent.
        super().__init__(parent)
        #wx.Panel.SetBackgroundColour() = self.SetBackgroundColour(): Método aplicado al objeto de la clase
        #Panel que recibe como parámetro el constructor de esta clase para cambiar el color del fondo del
        #contenedor, el método realiza esto recibiendo un parámetro que instancíe la clase Colour de la librería
        #wxPython, indicando el color en formato RGB:
        #  - wx.Colour(R, G, B): Los valores de RGB van de 0 a 255 y su combinación de colores rojo, verde y
        #    azul crean cualquier color existente, el valor (0, 0, 0) corresponde al color negro y
        #    (255, 255, 255) al blanco.
        self.SetBackgroundColour(wx.Colour(150, 0, 0))

        #CREACIÓN DE LOS WIDGETS: Botón
        #Instancia de la librería wxPython por medio del constructor de la clase Button para crear un widget de
        #tipo botón, en este se debe indicar como parámetro el texto que aparece sobre él y su posición.
        # - label = "": Con este parámetro se indica el texto que aparecerá sobre el botón.
        # - pos = (x, y): Con este atributo se indica la posición fija en pixeles del widget, 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.
#       - Es importante mencionar que si después se utiliza la clase BoxSizer para posicionar los
#         elementos de forma relativa, esta posición no es respetada.
browse_btn = wx.Button(self, label = "Browse", pos = (0,0))
#No se declaran como self.nombreObjeto los widgets a los que no se les vaya a extraer o introducir datos
#en el transcurso del funcionamiento de la interfaz gráfica.

#CREACIÓN DE LOS WIDGETS: Imágen
#Las clases wx.Image y wx.Bitmap están relacionadas y se utilizan en conjunto para trabajar con
#imágenes en la interfaz gráfica.
#   wx.Image: 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.
#
#   wx.Bitmap: Es una representación de imagen que puede ser utilizada directamente en los controles y
#   widgets de la interfaz gráfica de wxPython.

#Cargar una Imagen: Se crea una instancia de la librería wxPython por medio del constructor de la clase
#Image para cargar una imagen en memoria con datos en bruto.
# - filepath (str): 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.
# - type (int): Indica el tipo de imagen a cargar. Puede ser uno de los siguientes valores predefinidos:
#           - wx.BITMAP_TYPE_BMP: Bitmap de Windows.
#           - wx.BITMAP_TYPE_JPEG: JPEG.
#           - wx.BITMAP_TYPE_GIF: GIF.
#           - wx.BITMAP_TYPE_PNG: PNG.
#           - wx.BITMAP_TYPE_TIFF: TIFF.
#           - wx.BITMAP_TYPE_PNM: PNM (Portable anymap).
#           - wx.BITMAP_TYPE_PCX: PCX (ZSoft IBM PC Paintbrush file).
#           - wx.BITMAP_TYPE_TGA: TGA (Truevision TGA file).
#           - wx.BITMAP_TYPE_ICO: ICO (Icon file).
#           - wx.BITMAP_TYPE_CUR: CUR (Cursor file).
#           - wx.BITMAP_TYPE_XBM: XBM (X11 Bitmap).
#           - wx.BITMAP_TYPE_XPM: XPM (X11 Pixmap).
#           - wx.BITMAP_TYPE_WEBP: WEBP (WebP image format).
#           - wx.BITMAP_TYPE_ANY: Cualquier tipo de imagen compatible.
# - index (int): Este método se utiliza cuando se carga una imagen que contiene varias subimágenes
#   (como un archivo ICO o un archivo animado GIF). El valor predeterminado es -1 para cargar la imagen
#   principal.
# - data (str o bytes): Datos en bruto de la imagen en formato binario. Se utiliza cuando la imagen no
#   se carga desde un archivo, sino desde una fuente de datos en memoria.
# - mask (wx.Image o None): Representa una máscara opcional que se utilizará para darle transparencia
```

```
#   a la imagen. La máscara debe tener el mismo tamaño que la imagen principal y esta especifica cuáles
#   píxeles están completamente transparentes y cuáles son opacos.
# - size (tuple): Parámetro opcional que especifica el tamaño deseado para la imagen cargada. Se utiliza
#   para redimensionar la imagen a un tamaño específico. El tamaño debe ser una tupla de dos elementos
#   en forma de (ancho, alto).
#       - Desempaquetar (*): Si es que se quiere indicar el tamaño de una imagen perteneciente a un GUI
#         fuera de la clase donde fue creada se debe usar el operador * el cual se utiliza en Python
#         para desempaquetar una secuencia (como una lista, tupla o conjunto) en sus elementos
#         individuales.
#         Esto significa que los elementos de la secuencia se extraen en el órden en el que fueron
#         declarados, se pasan y usan como argumentos separados a una función o constructor que espera
#         múltiples argumentos.
#En sí existen dos widgets en esta GUI, uno vacío que se crea cuando se ejecuta la GUI sin haber
#seleccionado todavía el botón de Browse para cargar una imagen y el otro es el el que ya incluye la
#imagen cargada. El widget que se crea a continuación es el que se muestra inicialmente vacío,
#posteriormente en la función loadImage() se creará el otro que ya incluye la imagen después de haberse
#cargado.
img = wx.Image(*image_size)
#Crear un Bitmap: Una vez que se tiene una instancia de la clase wx.Image, se puede crear un objeto
#Bitmap, esto se puede realizar de dos formas:
#   - wx.Bitmap(wx.Image): El constructor de la clase Bitmap recibe como parámetro un objeto que
#     instancíe la clase Image, ambos pertenecientes a la librería de wxPython.
#   - wx.Image.ConvertToBitmap(): Método utilizado para convertir un objeto Image en uno Bitmap.
#Esto convierte la imagen en un formato adecuado para que pueda ser utilizado en controles como
#wx.StaticBitmap, wx.Button, wx.BitmapButton, entre otros.

#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 wx.StaticBitmap, que es parte de la biblioteca wxPython y se utiliza para
#mostrar una imagen estática en un panel. Los parámetros que puede recibir la clase wx.StaticBitmap son
#los siguientes:
# - parent: Es el objeto padre al que se añadirá el widget wx.StaticBitmap.
# - id: Un identificador único para el widget. Puede ser de tipo entero o wx.ID_ANY para permitir que
#   wxPython asigne automáticamente un ID.
# - bitmap: El objeto wx.Bitmap que se utilizará como imagen para el wx.StaticBitmap.
# - pos: Una tupla (x, y) o un objeto wx.Point que indica la posición inicial del widget. Si no se
#   proporciona, se utilizará la posición predeterminada.
# - size: Una tupla (width, height) o un objeto wx.Size que indica el tamaño del widget. Si no se
#   proporciona, se utilizará el tamaño predeterminado.
# - style: Un estilo adicional para el widget wx.StaticBitmap. Puede incluir combinaciones de banderas:
#           - wx.ALIGN_LEFT: Alinea el contenido del widget a la izquierda.
#           - wx.ALIGN_RIGHT: Alinea el contenido del widget a la derecha.
#           - wx.ALIGN_CENTER_HORIZONTAL: Centra horizontalmente el contenido del widget.
#           - wx.ALIGN_TOP: Alinea el contenido del widget en la parte superior.
#           - wx.ALIGN_BOTTOM: Alinea el contenido del widget en la parte inferior.
#           - wx.ALIGN_CENTER_VERTICAL: Centra verticalmente el contenido del widget.
```

```
#           - wx.ST_NO_AUTORESIZE: Evita que el widget se redimensione automáticamente cuando cambia el
#             tamaño de su contenido.
#   Además de estos estilos, hay otros disponibles que permiten modificar aspectos estéticos, se pueden
#   combinar varios estilos utilizando el operador OR (|) para crear la combinación deseada:
#           - wx.BORDER_NONE: No muestra ningún borde alrededor del widget.
#           - wx.BORDER_SIMPLE: Muestra un borde simple alrededor del widget.
#           - wx.BORDER_RAISED: Muestra un borde elevado alrededor del widget.
#           - wx.BORDER_SUNKEN: Muestra un borde hundido alrededor del widget.
#           - wx.BORDER_STATIC: Muestra un borde estático alrededor del widget.
# - name: El nombre del widget.
#Es importante destacar que los parámetros pos, size, style y name son opcionales y tienen valores
#predeterminados si no se proporcionan, el que si es necesario indicarlo es el objeto bitmap.
self.image_ctrl = wx.StaticBitmap(parent = self, bitmap = wx.Bitmap(img))
#Se declaran como self.nombreObjeto los widgets a los que sí se les vaya a extraer o introducir datos
#en el transcurso del funcionamiento de la interfaz gráfica.

#CREACIÓN DE LOS WIDGETS: Área de Texto
#Instancia de la librería wxPython por medio del constructor de la clase TextCtrl para crear un widget
#que proporciona una caja de texto de una sola línea o de varias líneas en la que los usuarios pueden
#ingresar y editar texto, en este se deben indicar los siguientes parámetros:
# - parent: El widget padre al que pertenece la caja de texto, en este caso es self porque pertenece
#   al Panel.
# - id: Un identificador único. Puede ser un número entero o el valor wx.ID_ANY para que wxPython
#   seleccione automáticamente un identificador.
# - value: El texto inicial que se mostrará en la caja de texto.
# - pos: La posición de la caja de texto en coordenadas (x, y) relativas al widget padre.
# - size: El tamaño de la caja de texto en píxeles (ancho, alto).
#           - El valor -1 indica que la altura se ajustará automáticamente según el contenido de la
#             caja de texto.
# - style: Estilos que se aplicarán a la caja de texto para modificar su apariencia y comportamiento.
#           - wx.TE_MULTILINE: Permite la edición de texto en varias líneas en lugar de una sola.
#           - wx.TE_PASSWORD: Oculta el texto ingresado, como en el caso de contraseñas, mostrando
#             asteriscos o puntos en su lugar.
#           - wx.TE_READONLY: Hace que la caja de texto sea de solo lectura, lo que impide que los
#             usuarios editen el contenido.
#           - wx.TE_PROCESS_ENTER: Genera un evento wx.EVT_TEXT_ENTER cuando se presiona la tecla
#             Enter en la caja de texto.
#           - wx.TE_CENTER: Centra el texto dentro de la caja de texto.
#           - wx.TE_LEFT: Alinea el texto a la izquierda dentro de la caja de texto.
#           - wx.TE_RIGHT: Alinea el texto a la derecha dentro de la caja de texto.
# - validator: Un objeto wx.Validator que se utiliza para validar y controlar la entrada de texto, esto
#   se utiliza por ejemplo en las cajas de texto donde se ingresa un correo, en donde a fuerza debe
#   existir un @, hotmail.com, gmail.com, etc.
#           - wx.TextValidator: Permite validar la entrada de texto utilizando expresiones regulares
#             y otras restricciones personalizadas.
```

```
#           - wx.IntegerValidator: Permite ingresar solo valores enteros en la caja de texto.
#           - wx.FloatingPointValidator: Permite ingresar solo valores de punto flotante en la caja de
#             texto.
#           - wx.Validator: Clase base para crear validadores personalizados. Permite controlar el
#             formato y las restricciones de entrada.
# - name: El nombre de la caja de texto.
# - maxLength: La longitud máxima de caracteres que conforman el texto que se puede ingresar.
# - initialValue: El valor inicial de la caja de texto antes de que se haya ingresado cualquier texto.
# - autoCompleteMode: El modo de autocompletado que se aplicará a la caja de texto.
# - autoCompleteChoices: Una lista de opciones para el autocompletado.
# - passwordStyle: Si se establece en True, el texto ingresado se ocultará (por ejemplo, para ingresar
#   contraseñas).
# - readonly: Si se establece en True, la caja de texto será de solo lectura y no se permitirá la
#   edición del texto.
# - multiline: Si se establece en True, la caja de texto permitirá la entrada de múltiples líneas de
#   texto.
self.photo_txt = wx.TextCtrl(self, style = wx.TE_READONLY, size = (200, -1))

#POSICIONAMIENTO DE ELEMENTOS:
#El siguiente código crea una instancia de la clase BoxSizer, la cual permite un posicionamiento
#relativo, colocando así un elemento respecto a otro, para poder usar esta clase el primer objeto se
#debe encontrar dentro del segundo. Si no se usa la clase BoxSizer, los elementos se colocarán unos
#encima de los otros.
#Al crear la Instancia de la clase BoxSizer perteneciente a la librería wxPython se le puede pasar como
#parámetro solamente dos posibles atributos para indicar la dirección de la posición del objeto:
# - wx.HORIZONTAL: Hace que la dirección de la alineación del primer objeto sea horizontal respecto al
#   segundo, esto se refiere a que se empiece indicar la posición del widget desde la esquina izquierda
#   dentro del contenedor.
# - wx.VERTICAL: Hace que la dirección de la alineación del primer objeto sea vertical respecto al
#   segundo, esto se refiere a que se empiece indicar la posición del widget desde la parte superior de
#   en medio, dentro del contenedor.
hsizer = wx.BoxSizer(wx.HORIZONTAL)     #Sizer del botón y Área de texto de su ruta.
main_sizer = wx.BoxSizer(wx.VERTICAL)   #Sizer que incluye una imagen y al contenedor hziser.

#Ya se mencionó que la clase BoxSizer sirve al posicionar un elemento respecto a otro, para ello uno de
#los elementos se debe encontrar dentro del otro, con el objetivo de indicar cuál es el primer objeto
#(el que tiene posicionamiento relativo) y cuál es el segundo objeto (el que contiene al primer objeto),
#se utilizan los métodos .Add() y .SetSizer() de la siguiente forma:

#Instancia_BoxSizer.Add(): Método utilizado para agregar un elemento al sizer, sizer se refiere al
#elemento que contiene a otro que está colocado dentro de él con posicionamiento relativo, para ello
#dentro de su paréntesis se agregan los siguientes parámetros:
# - primer_parámetro: Con este parámetro se indica qué objeto que será agregado dentro del otro, el
#   objeto contenedor es llamado sizer.
# - proportion: Este parámetro determina cómo se asignará el espacio de todos los elementos que se
```

```
#   encuentran dentro del contenedor, aún si el sizer se expande o se reduce.
#           - proportion = 0: El elemento no crecerá ni se encogerá en relación a otros elementos
#             dentro del sizer.
#           - proportion = valor: El elemento se expandirá o se encogerá proporcionalmente en el sizer,
#             dependiendo del valor de los demás elementos.
#                   Por ejemplo, si hay dos elementos en el sizer, ambos con proportion = 1, cada uno
#                   ocupará la mitad del espacio disponible cuando el sizer se expanda. Si uno de los
#                   elementos tiene proportion = 2, ocupará dos tercios del espacio disponible, mientras
#                   que el otro elemento ocupará un tercio cuando el sizer se expanda.
# - flag: El parámetro flag se utiliza para especificar las opciones de posicionamiento y alineación
#   del elemento dentro del sizer por medio de banderas, las acciones de estas banderas se pueden
#   combinar usando la operación lógica OR (|), las flags que se pueden usar son descritas a
#   continuación:
#           - wx.EXPAND: Hace que el elemento se expanda para ocupar todo el espacio disponible en la
#             dirección del sizer.
#           - wx.ALL: Agrega un borde en todos los lados del elemento.
#           - wx.LEFT: Agrega un borde en el lado izquierdo del elemento.
#           - wx.RIGHT: Agrega un borde en el lado derecho del elemento.
#           - wx.TOP: Agrega un borde en la parte superior del elemento.
#           - wx.BOTTOM: Agrega un borde en la parte inferior del elemento.
#           - wx.CENTER: Centra el elemento dentro del espacio asignado por el sizer.
#           - wx.ALIGN_LEFT: Alinea el elemento a la izquierda dentro del espacio asignado por el sizer.
#           - wx.ALIGN_RIGHT: Alinea el elemento a la derecha dentro del espacio asignado por el sizer.
#           - wx.ALIGN_TOP: Alinea el elemento en la parte superior dentro del espacio asignado por el
#             sizer.
#           - wx.ALIGN_BOTTOM: Alinea el elemento en la parte inferior dentro del espacio asignado por
#             el sizer.
# - border: Establece el espacio en píxeles entre el elemento y los bordes del sizer.
#Botón agregado al contenedor hsizer
hsizer.Add(browse_btn, proportion = 0,              #Botón agregado al hsizer, proportion = 0
                       flag = wx.ALL | wx.CENTER)  #Borde en todos los lados del elemento y Centrado.
#Área de texto agregada al contenedor hsizer
hsizer.Add(self.photo_txt, proportion = 1,          #Área de texto agregada al hsizer, proportion = 1
                       flag = wx.ALL | wx.CENTER,  #Borde en todos los lados del elemento y Centrado.
                       border = 10)                #Borde de 10 pixeles.
#Imagen agregada al contenedor main_sizer
main_sizer.Add(self.image_ctrl, proportion = 1,     #Imágen agregada al main_sizer, proportion = 0
                       flag = wx.ALL | wx.CENTER | wx.BOTTOM, #Bordes, Centrado y Alineado abajo.
                       border = 10)                #Borde de 10 pixeles.
#Contenedor hsizer agregado al contenedor main_sizer
main_sizer.Add(hsizer, proportion = 0,              #Panel hsizer agregado al main_sizer, proportion = 0
                       flag = wx.ALL | wx.EXPAND,  #Borde en todos los lados del elemento y Expandido.
                       border = 10)                #Borde de 10 pixeles.
#wx.Panel.SetSizer() = self.SetSizer(): Método aplicado al objeto de la clase Panel que recibe como
#parámetro esta clase, el cuál recibe como parámetro un objeto de la clase BoxSizer para indicar cuál es
```

```
        #el elemento contenedor al que ya se han agregado anteriormente uno o más widgets posicionados
        #relativamente con el método .Add().
        self.SetSizer(main_sizer)
        #wx.BoxSizer.Fit() = sizer.SetSizer(): Método aplicado al objeto de la clase BoxSizer, el cual recibe
        #como parámetro al parent del constructor de la clase Panel. El método se utiliza para ajustar
        #automáticamente el tamaño del contenedor al tamaño óptimo para que quepa y se vea todo su contenido.
        main_sizer.Fit(parent)
        #wx.Panel.SetSizer() = self.SetSizer(): Método aplicado al objeto de la clase Panel que recibe como
        #parámetro esta clase, el no recibe nada como parámetro. El método se utiliza para recalcular la
        #disposición y la posición de los elementos dentro del contenedor después de realizar cambios en ellos,
        #osea al haber dado clic sobre un botón y que se haya ejecutado su acción en el GUI o que se haya
        #cambiado de tamaño la ventana.
        #Ambos métodos .Fit() y .Layout() son importantes para garantizar que los elementos se muestren
        #correctamente y se ajusten adecuadamente en la interfaz de usuario.
        self.Layout()

        #ACCIONES DEL BOTÓN: Para ejecutar la acción de un botón en Python, dentro del constructor de la clase
        #Panel se debe mandar a llamar una función (método) que describa la acción a ejecutar, esta función se
        #debe encontrar fuera del constructor pero pertenecer igualmente a la clase Panel.

        #ATRIBUTOS DEL CONSTRUCTOR DE LA CLASE PANEL: Dentro del constructor se deben declarar las variables con
        #las que interactúen los botones para realizar acciones dentro del GUI.
        self.max_img_size = image_size #Atributo de la clase Panel que indica el tamaño del widget imagen.

        #Instancia_Button.Bind(): Este método se utiliza para enlazar un evento a un controlador de eventos,
        #indicando en su primer parámetro el evento que detona el método y en el segundo la función que
        #se ejecutará cuando ese evento ocurra. Normalmente las funciones que describen las acciones a
        #realizar por los elementos del Panel se encuentran dentro de esta misma clase, pero fuera de su
        #constructor.
        # - Tipos de Eventos en Python:
        #       - wx.EVT_BUTTON: Evento que se activa cuando se hace clic en un botón.
        #       - wx.EVT_TEXT: Evento que se activa cuando se cambia el contenido de un control de texto.
        #       - wx.EVT_CHECKBOX: Evento que se activa cuando se cambia el estado de una casilla de
        #         verificación.
        #       - wx.EVT_COMBOBOX: Evento que se activa cuando se selecciona un elemento de una lista
        #         desplegable (combobox).
        #       - wx.EVT_LISTBOX: Evento que se activa cuando se selecciona un elemento de una lista (listbox).
        #       - wx.EVT_RADIOBUTTON: Evento que se activa cuando se selecciona un botón de opción (radiobutton)
        #         en un grupo de botones de opción.
        #       - wx.EVT_MENU: Evento que se activa cuando se selecciona una opción de menú.
        #       - wx.EVT_CLOSE: Evento que se activa cuando se intenta cerrar una ventana o diálogo.
        #       - wx.EVT_KEY_DOWN y wx.EVT_KEY_UP: Evento que se activan cuando se presiona o se suelta una
        #         tecla del teclado, respectivamente.
        #       - wx.EVT_MOUSE_EVENTS: Son una serie de eventos relacionados con las interacciones del ratón,
        #         como clics, movimiento, etc. Estos eventos son descritos a continuación:
```

```
        #               - wx.EVT_LEFT_DOWN: Se activa cuando se presiona el botón izquierdo del ratón.
        #               - wx.EVT_LEFT_UP: Se activa cuando se suelta el botón izquierdo del ratón.
        #               - wx.EVT_LEFT_DCLICK: Se activa cuando se hace doble clic con el botón izquierdo del
        #                 ratón.
        #               - wx.EVT_RIGHT_DOWN: Se activa cuando se presiona el botón derecho del ratón.
        #               - wx.EVT_RIGHT_UP: Se activa cuando se suelta el botón derecho del ratón.
        #               - wx.EVT_RIGHT_DCLICK: Se activa cuando se hace doble clic con el botón derecho del
        #                 ratón.
        #               - wx.EVT_MIDDLE_DOWN: Se activa cuando se presiona el botón central del ratón.
        #               - wx.EVT_MIDDLE_UP: Se activa cuando se suelta el botón central del ratón.
        #               - wx.EVT_MIDDLE_DCLICK: Se activa cuando se hace doble clic con el botón central del
        #                 ratón.
        #               - wx.EVT_MOTION: Se activa cuando se mueve el ratón dentro del área del objeto
        #                 capturador.
        #               - wx.EVT_ENTER_WINDOW: Se activa cuando el ratón entra en el área del objeto
        #                 capturador.
        #               - wx.EVT_LEAVE_WINDOW: Se activa cuando el ratón sale del área del objeto capturador.
        #               - wx.EVT_MOUSEWHEEL: Se activa cuando se desplaza la rueda del ratón.
        #Evento de clic en botón, ejecutado por la función on_browse de esta clase Panel.
        browse_btn.Bind(wx.EVT_BUTTON, self.on_browse)


    #función on_browse(): Método creado dentro de la clase propia llamada ImageBrowsePanel 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 un botón y debe abrir el explorador de archivos
    #para seleccionar la imagen que se va a mostrar en el GUI, además se introduce el directorio de su ubicación
    #en el área de texto de la ventana.
    def on_browse(self, event):
        #VARIABLES DE LA FUNCIÓN:
        #La variable img_type es una variable propia de esta función, esta declara un string para indicar cuál
        #es el tipo de imágenes que se acepta para mostrar en el GUI.
        img_type = "JPEG files (*.jpg)|*.jpg"  #Palabra que se muestra al en el explorador de archivos para jpg.

        #Instancia de la librería wxPython por medio del constructor de la clase FileDialog para ejecutar una
        #ventana de selección de archivos. A través de este cuadro de diálogo, los usuarios pueden seleccionar
        #un archivo.
        # - with: La palabra reservada with se utiliza para garantizar una gestión adecuada de los recursos del
        #   explorador de archivos. Cuando se finaliza el bloque with, se liberan automáticamente los recursos
        #   utilizados por el cuadro de diálogo.
        #   - parent: Si se le pasa un valor None a este argumento lo que es significa que no hay un padre
        #     específico para el cuadro de diálogo.
        #   - message: Con este parámetro se indica el texto que aparecerá sobre el explorador de archivos.
        #   - wildcard: Define el tipo de archivos que se pueden seleccionar, esto se debe ingresar como un
        #     string.
        #   - style: Indica el estilo o comportamiento de la ventana de selección de archivos. Puede tomar
```

```
#     varios valores para personalizar la apariencia y la funcionalidad del cuadro de diálogo.
#         - wx.FD_OPEN: Este es el estilo predeterminado y se utiliza para permitir al usuario
#           seleccionar un archivo existente. El explorador mostrará los archivos y directorios en la
#           ubicación especificada.
#         - wx.FD_SAVE: Este estilo se utiliza cuando se desea permitir al usuario seleccionar una
#           ubicación para guardar un archivo nuevo. El explorador mostrará los archivos y directorios
#           en la ubicación especificada y proporcionará una opción para ingresar un nombre de archivo.
#         - wx.FD_OVERWRITE_PROMPT: Este estilo se utiliza en combinación con wx.FD_SAVE y muestra una
#           advertencia si el archivo seleccionado ya existe. El cuadro de diálogo mostrará un mensaje
#           preguntando al usuario si desea sobrescribir el archivo existente. las acciones de estos
#           estilos se pueden combinar usando la operación lógica OR (|).
#         - wx.FD_MULTIPLE: Este estilo permite la selección de múltiples archivos. En lugar de
#           seleccionar un solo archivo, el usuario puede seleccionar varios archivos a la vez
#           utilizando la tecla de modificación apropiada (como Ctrl o Shift).
#         - wx.FD_CHANGE_DIR: Este estilo indica que el cuadro de diálogo debe cambiar el directorio de
#           trabajo actual según la ubicación seleccionada por el usuario. Esto puede ser útil si desea
#           cambiar automáticamente el directorio de trabajo al directorio del archivo seleccionado.
#         - wx.FD_PREVIEW: Este estilo muestra una vista previa del archivo seleccionado en el cuadro de
#           diálogo, si es posible. La vista previa puede ser una imagen, un documento o cualquier tipo
#           de archivo que pueda ser mostrado en el cuadro de diálogo.
# - dialog: Objeto que permite mostrar cuadros de diálogo.
#   - dialog.ShowModal(): Muestra el cuadro de diálogo y bloquea la ejecución del programa hasta que el
#     usuario seleccione un archivo o cierre el cuadro de diálogo, este método devuelve un código que
#     indica el resultado de la interacción del usuario con el cuadro de diálogo, como lo son:
#         - wx.ID_OK: Indica que el usuario ha seleccionado y confirmado una opción en el explorador de
#           archivos.
#         - wx.ID_CANCEL: Indica que el usuario ha cancelado el cuadro de diálogo sin seleccionar ningún
#           archivo. Puede ocurrir si el usuario hace clic en el botón "Cancelar" o si cierra el
#           explorador de archivos.
#         - wx.ID_YES: Indica que el usuario ha confirmado la selección en el cuadro de diálogo. Este
#           valor puede ser devuelto si se utiliza wx.FD_SAVE en el estilo del cuadro de diálogo y el
#           usuario decide sobrescribir un archivo existente.
#         - wx.ID_NO: Indica que el usuario ha rechazado la selección en el cuadro de diálogo. Esto
#           puede ocurrir si se utiliza wx.FD_SAVE en el estilo del cuadro de diálogo y el usuario
#           decide no sobrescribir un archivo existente.
#         - wx.ID_APPLY: Este valor puede ser utilizado para realizar alguna acción adicional después de
#           seleccionar un archivo en el cuadro de diálogo. No está directamente relacionado con la
#           selección del archivo en sí, y su uso depende de la implementación específica.
#   - dialog.GetPath(): Devuelve la ruta completa del archivo seleccionado por el usuario en el cuadro
#     de diálogo. La ruta incluirá tanto el directorio como el nombre del archivo.
#   - dialog.GetPaths(): Si el cuadro de diálogo permite la selección múltiple de archivos (configurado
#     con wx.FD_MULTIPLE en el argumento style), este método devuelve una lista de las rutas completas
#     de los archivos seleccionados.
#   - dialog.GetDirectory(): Devuelve el directorio seleccionado por el usuario en el cuadro de diálogo.
#     Si el usuario ha seleccionado múltiples archivos de diferentes directorios, esta función devuelve
```

```
        #     el directorio del primer archivo seleccionado.
        #   - dialog.GetFilenames(): Devuelve una lista de los nombres de archivo seleccionados por el usuario
        #     en el cuadro de diálogo. Los nombres de archivo no incluyen la ruta del directorio.
        with wx.FileDialog(parent = None, message = 'Elige una imagen jpg',
                           wildcard = img_type,               #Tipo de archivo aceptado = Imagen .jpg
                           style = wx.FD_OPEN) as dialog:     #Permite seleccionar un archivo
            #Si el usuario ha seleccionado un archivo correctamente, por lo cual el método dialog.ShowModal() ha
            #devuelto un valor wx.ID_OK, se devuelve la ruta completa de su ubicación y se coloca en el área de
            #texto del GUI llamada photo_txt.
            if (dialog.ShowModal() == wx.ID_OK):
                #Setters y Getters: Como los valores de los atributos están encapsulados por haber sido creados
                #dentro del constructor de la clase, existen dos tipos de métodos especiales para poder editar
                #o extraer el valor de estas variables de clase:
                #   - nombre_Atributo.SetValue(): Método setter que permite editar el valor de un atributo.
                #   - nombre_Atributo.GetValue(): Método getter que permite obtener el valor de un atributo.

                self.photo_txt.SetValue(dialog.GetPath())
                #Una vez que se haya abierto la imagen, obtenido su directorio y colocado en el área de texto
                #del GUI llamada photo_txt, se ejecuta una segunda función llamada loadImage() para colocar la
                #imagen en el área de imagen del GUI.
                self.loadImage()


    #función loadImage(): Método creado dentro de la clase propia llamada ImageBrowsePanel que recibe como
    #parámetro a la instancia del objeto Frame que recibe esta misma clase como parámetro, por eso se usa la
    #palabra reservada self. Esta función es llamada cuando termina su ejecución la función on_browse(), que a
    #su vez es llamada cuando ocurre un evento de clic de botón.
    def loadImage(self):
        #Setters y Getters: Como los valores de los atributos están encapsulados por haber sido creados dentro
        #del constructor de la clase, existen dos tipos de métodos especiales para poder editar o extraer el
        #valor de estas variables de clase:
        #   - nombre_Atributo.SetValue(): Método setter que permite editar el valor de un atributo.
        #   - nombre_Atributo.GetValue(): Método getter que permite obtener el valor de un atributo.
        filepath = self.photo_txt.GetValue()
        print("Directorio de la imagen\n", filepath)

        #Cargar una Imagen: Se crea una instancia de la librería wxPython por medio del constructor de la clase
        #Image para cargar una imagen en memoria con datos en bruto.
        # - filepath (str): En este atributo se indica el nombre de archivo o ruta completa de la imagen que se
        #   va a cargar, su tipo de dato debe de ser 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.
        # - type (int): El tipo de imagen a cargar. Puede ser uno de los siguientes valores predefinidos:
        #           - wx.BITMAP_TYPE_BMP: Bitmap de Windows.
        #           - wx.BITMAP_TYPE_JPEG: JPEG.
```

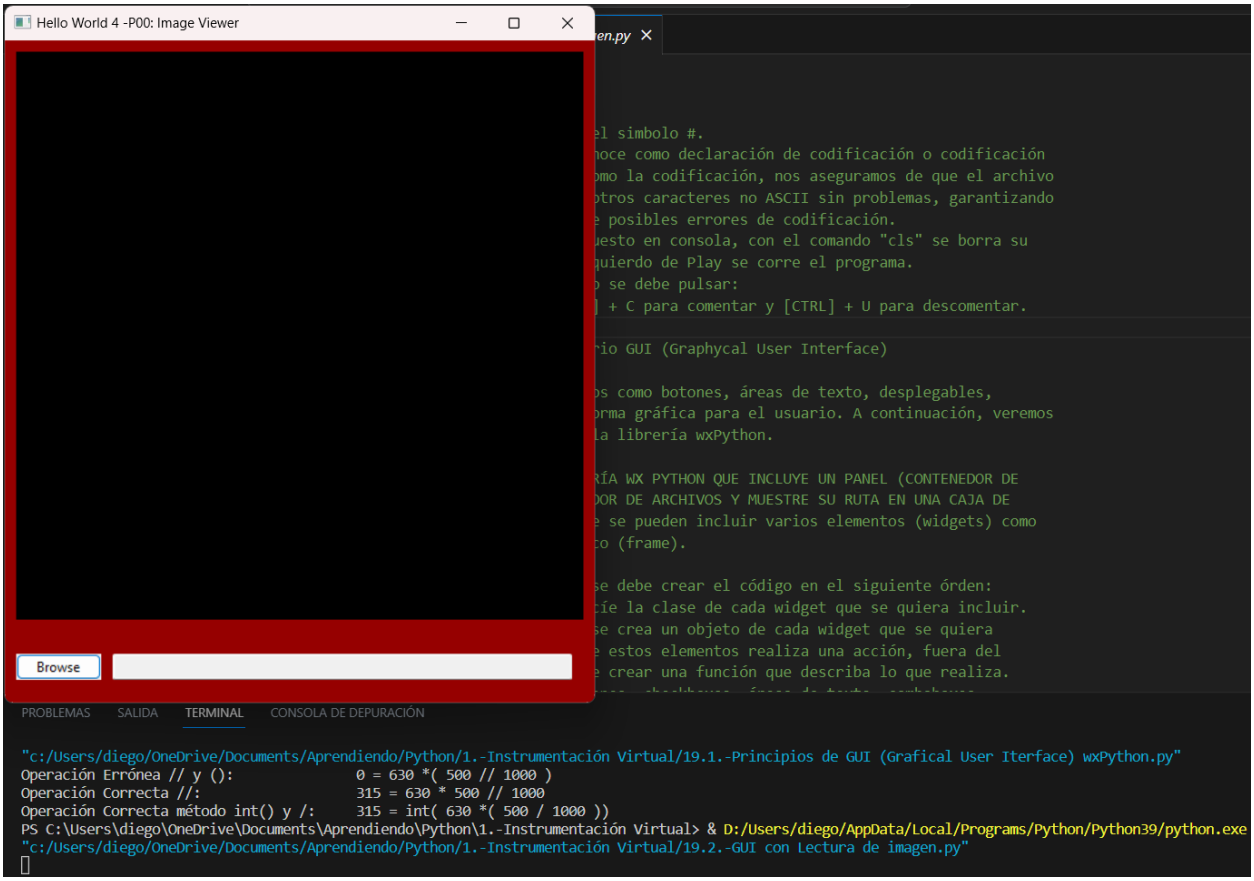
```
#           - wx.BITMAP_TYPE_GIF: GIF.
#           - wx.BITMAP_TYPE_PNG: PNG.
#           - wx.BITMAP_TYPE_TIFF: TIFF.
#           - wx.BITMAP_TYPE_PNM: PNM (Portable anymap).
#           - wx.BITMAP_TYPE_PCX: PCX (ZSoft IBM PC Paintbrush file).
#           - wx.BITMAP_TYPE_TGA: TGA (Truevision TGA file).
#           - wx.BITMAP_TYPE_ICO: ICO (Icon file).
#           - wx.BITMAP_TYPE_CUR: CUR (Cursor file).
#           - wx.BITMAP_TYPE_XBM: XBM (X11 Bitmap).
#           - wx.BITMAP_TYPE_XPM: XPM (X11 Pixmap).
#           - wx.BITMAP_TYPE_WEBP: WEBP (WebP image format).
#           - wx.BITMAP_TYPE_ANY: Cualquier tipo de imagen compatible.
# - index (int): Este método se utiliza cuando se carga una imagen esta que contiene varias subimágenes
#   (como un archivo ICO o un archivo animado GIF). El valor predeterminado es -1 para cargar la imagen
#   principal.
# - data (str o bytes): Datos en bruto de la imagen en formato binario. Se utiliza cuando no se carga
#   desde un archivo, sino desde una fuente de datos en memoria.
# - mask (wx.Image o None): Una imagen de máscara opcional que se utilizará para transparencia. La
#   imagen de máscara debe tener el mismo tamaño que la imagen principal y especifica qué píxeles están
#   completamente transparentes y cuáles son opacos.
# - size (tuple): Parámetro opcional que especifica el tamaño deseado para la imagen cargada. Se utiliza
#   para redimensionar la imagen a un tamaño específico. El tamaño debe ser una tupla de dos elementos
#   en forma de (ancho, alto).
#       - Desempaquetar (*): Si es que se quiere indicar el tamaño de una imagen perteneciente a un GUI
#         fuera de la clase donde fue creada se debe usar el operador * el cual se utiliza en Python
#         para desempaquetar una secuencia (como una lista, tupla o conjunto) en sus elementos
#         individuales.
#         Esto significa que los elementos de la secuencia se extraen en el órden en el que fueron
#         declarados, se pasan y usan como argumentos separados a una función o constructor que espera
#         múltiples argumentos.
img = wx.Image(filepath, type = wx.BITMAP_TYPE_ANY)

#Una vez que se haya cargado la imagen y asignado el tamaño del widget donde aparece por medio del
#constructor, se ejecuta una función declarada dentro de la clase ImagePanel llamada loadImage(), que
#sirve para ajustar el tamaño de la imagen cargada antes de asignarla al widget y que aparezca en el
#GUI.
img = self.scale_image(img, self.max_img_size)

#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 wx.StaticBitmap, que es parte de la biblioteca wxPython y se utiliza para
#mostrar una imagen estática en un panel. Los parámetros que puede recibir la clase wx.StaticBitmap son
#los siguientes:
# - parent: Es el objeto padre al que se añadirá el widget wx.StaticBitmap.
# - id: Un identificador único para el widget. Puede ser de tipo entero o wx.ID_ANY para permitir que
#   wxPython asigne automáticamente un ID.
```

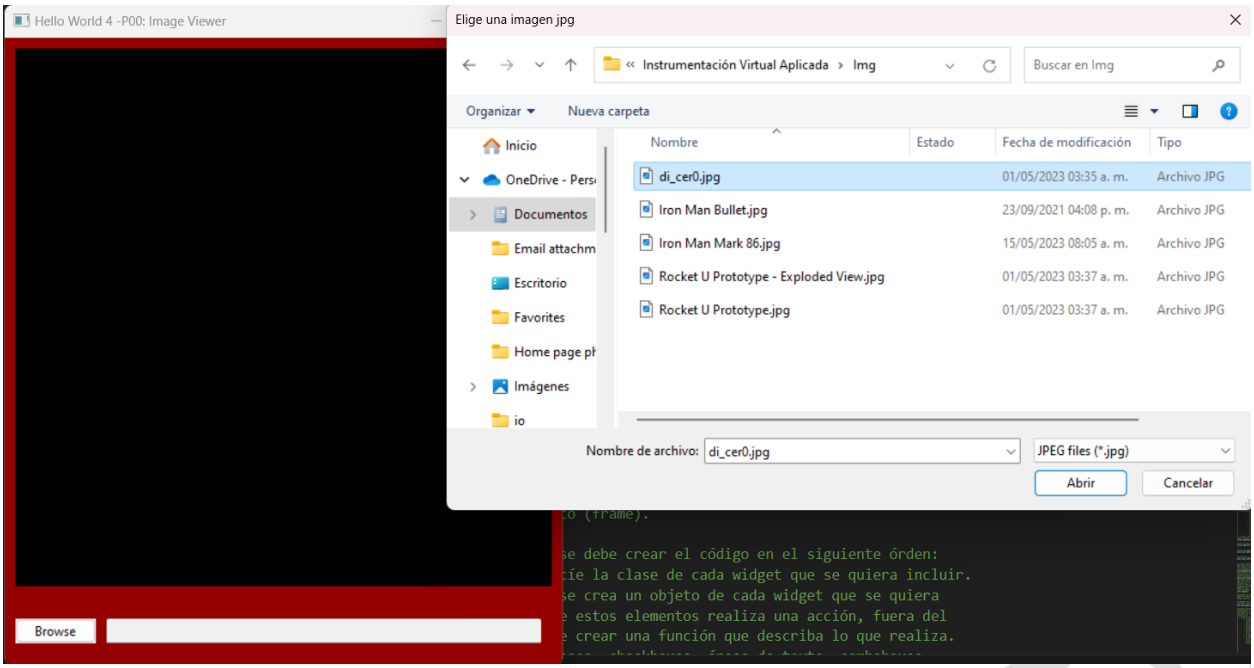
```
        # - bitmap: El objeto wx.Bitmap que se utilizará como imagen para el wx.StaticBitmap.
        # - pos: Una tupla (x, y) o un objeto wx.Point que indica la posición inicial del widget. Si no se
        #   proporciona, se utilizará la posición predeterminada.
        # - size: Una tupla (width, height) o un objeto wx.Size que indica el tamaño del widget. Si no se
        #   proporciona, se utilizará el tamaño predeterminado.
        # - style: Un estilo adicional para el widget wx.StaticBitmap. Puede incluir combinaciones de banderas:
        #           - wx.ALIGN_LEFT: Alinea el contenido del widget a la izquierda.
        #           - wx.ALIGN_RIGHT: Alinea el contenido del widget a la derecha.
        #           - wx.ALIGN_CENTER_HORIZONTAL: Centra horizontalmente el contenido del widget.
        #           - wx.ALIGN_TOP: Alinea el contenido del widget en la parte superior.
        #           - wx.ALIGN_BOTTOM: Alinea el contenido del widget en la parte inferior.
        #           - wx.ALIGN_CENTER_VERTICAL: Centra verticalmente el contenido del widget.
        #           - wx.ST_NO_AUTORESIZE: Evita que el widget se redimensione automáticamente cuando cambia el
        #             tamaño de su contenido.
        #   Además de estos estilos, hay otros disponibles que permiten modificar aspectos estéticos, se pueden
        #   combinar varios estilos utilizando el operador OR (|) para crear la combinación deseada:
        #           - wx.BORDER_NONE: No muestra ningún borde alrededor del widget.
        #           - wx.BORDER_SIMPLE: Muestra un borde simple alrededor del widget.
        #           - wx.BORDER_RAISED: Muestra un borde elevado alrededor del widget.
        #           - wx.BORDER_SUNKEN: Muestra un borde hundido alrededor del widget.
        #           - wx.BORDER_STATIC: Muestra un borde estático alrededor del widget.
        # - name: El nombre del widget.
        #Es importante destacar que los parámetros pos, size, style y name son opcionales y tienen valores
        #predeterminados si no se proporcionan, el que si es necesario indicarlo es el objeto bitmap.
        self.image_ctrl.SetBitmap(wx.Bitmap(img))
        self.Refresh()


    #función scale_image(): Método creado dentro de la clase propia llamada ImagePanel que recibe como
    #parámetro a la imagen cargada con el objeto wx.Image y el tamaño del widget donde aparecerá dicha imagen,
    #para después ajustar el tamaño de la imagen y que se vea completa en el widget del GUI.
    def scale_image(self, img, size):
        #De la variable size, la cual es una tupla de dos elementos en forma de (ancho, alto) que se le pasa
        #como parámetro al constructor de esta clase y representa el tamaño del widget que mostrará la imagen,
        #se obtiene y separa en dos variables distintas su ancho y alto para posteriormente ajustar el tamaño
        #del objeto Image, que es la imagen cargada y representada en datos brutos.
        widget_width, widget_height = size  #Ancho y Alto en pixeles del widget.
        #wx.Image.GetWidth(): Método para obtener el ancho en pixeles de una imagen.
        img_width = img.GetWidth()          #Ancho en pixeles de la imagen en datos brutos.
        print("Ancho imagen:\t", img_width)
        #wx.Image.GetHeight(): Método para obtener la altura en pixeles de una imagen.
        img_height = img.GetHeight()        #Alto en pixeles de la imagen en datos brutos.
        print("Alto imagen:\t", img_height)

        #Si el ancho de la imagen es mayor que el ancho del widget, entonces se crearán nuevos valores para su
```

```
        #ancho y alto, logrando así que se ajuste su tamaño al del widget, manteniendo su forma original:
        if img_width > widget_width:
            #El nuevo ancho de la imagen es igual al ancho del widget
            new_width = widget_width
            #La nueva altura de la imagen se obtiene al realizar una regla de 3 donde se busca obtener la altura
            #proporcional del widget, conociendo ya la altura y el ancho de la imagen original y el ancho del
            #widget al que se quiere llegar, por lo tanto la fórmula de la regla de 3 sería la siguiente:
            #   img_width   ->  widget_width
            #   img_height  ->  widget_height
            #Por lo cual, para obtener el widget_height y asignarlo a el nuevo ancho se utiliza la siguiente
            #fórmula: widget_height = new_height = img_height*(widget_width/img_width)
            #El doble signo de "//" se utiliza en Python para realizar una división entera o "floor division",
            #a diferencia del operador de división normal ("/"), que devuelve un número decimal, el operador
            #"//" devuelve el cociente entero de la división, redondeando hacia abajo al número entero más
            #cercano.
            #No se debe poner paréntesis en la operación matemática porque sino se puede confundir el programa
            #y realizar mal la operación, ya que si primero realiza la división y obtiene un número decimal,
            #esto se redondearía a cero y la operación daría un resultado erróneo.
            new_height = img_height * (widget_width // img_width)
            print("Operación Errónea // y ():\t\t", new_height, "=", img_height, "*(", widget_width, "//", img_width, ")")
            new_height = img_height * widget_width // img_width
            print("Operación Correcta //:\t\t\t", new_height, "=", img_height, "*", widget_width, "//", img_width)
            new_height = int(img_height * (widget_width / img_width))
            print("Operación Correcta método int() y /:\t", new_height, "= int(", img_height, "*(", widget_width, "/",
img_width, "))")

            #wx.Image.Scale(): Método para redimensionar una imagen. Este método ajusta la imagen al tamaño
            #especificado y devuelve una nueva instancia del objeto wx.Image con las dimensiones modificadas.
            img = img.Scale(new_width, new_height, wx.IMAGE_QUALITY_HIGH)
        #Si la altura de la imagen es mayor que la altura del widget, entonces se crearán nuevos valores para su
        #ancho y alto, logrando así que se ajuste su tamaño al del widget, manteniendo su forma original:
        elif img_height > widget_height:
            #La nueva altura de la imagen es igual a la altura del widget
            new_height = widget_height
            #El nuevo ancho de la imagen se obtiene a través de una regla de 3, muy similar a como se realizó en
            #el condicional de arriba.
            new_width = widget_height * img_width // img_height

            #wx.Image.Scale(): Método para redimensionar una imagen. Este método ajusta la imagen al tamaño
            #especificado y devuelve una nueva instancia del objeto wx.Image con las dimensiones modificadas.
            img = img.Scale(new_width, new_height, wx.IMAGE_QUALITY_HIGH)
        #Al terminar la función se devuelve la nueva img redimensionada para que esa sea puesta en el Bitmap.
        return img
```

```
#MainFrame: La clase hereda de la clase Frame que pertenece a la librería wxPython, representa la ventana del
#GUI y crea una instancia de la clase MyPanel para agregar dentro de la ventana un contenedor con elementos
#dentro.
class MainFrame(wx.Frame):
    #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):
        #Dentro del constructor de la GUI se declara una instancia de la librería wxPython por medio de la cual
        #se accede al constructor de la clase Frame, que representa la ventana del GUI, a dicha ventana se le
        #asigna un título por medio de su segundo parámetro.
        super().__init__(None, title = "Hello World 4 -P00: Image Viewer")

        #ATRIBUTOS DEL CONSTRUCTOR DE LA CLASE FRAME: Dentro del constructor se deben declarar las variables con
        #las que interactúen los elementos dentro del Frame.
        #No es lo mismo el tamaño de la imagen que está descrito por el atributo max_img_size dentro de la
        #clase ImageBrowsePanel al tamaño de la ventana donde se muestra la imagen, que ahora se está declarando
        #como un atributo propio de la clase Frame llamado img_widget_size, esto siempre y cuando se quiera que
        #la imagen aparezca en un cuadrado, con ancho y alto del mismo valor.
        self.img_widget_size = 500      #Tamaño de la ventana de imagen, dado en pixeles = (ancho, alto)

        #Instancia de la clase ImageBrowsePanel para agregar el panel al Frame, osea el contenedor de elementos
        #a la ventana de la GUI, se le pasa como parámetro a la instancia del objeto Frame que recibe esta misma
        #clase como parámetro, por eso se usa la palabra reservada self.
        #Además se le indica el tamaño que tendrá el área donde aparecerá la imagen en la ventana con el
        #parámetro image_size, que fue declarado como un parámetro del constructor.
        panel = ImageBrowsePanel(self, image_size = (self.img_widget_size, self.img_widget_size))
        #wxPython.Frame.show() = self.show(): Método aplicado al objeto de la clase Frame que recibe como
        #parámetro el constructor de esta clase para mostrar la ventana del GUI.
        self.Show()


#__name__ == __main__: Método main, esta función es super importante ya que sirve para ejecutar los métodos de
#las clases declaradas en el archivo, en python pueden existir varios métodos main en un solo programa.
if(__name__ == '__main__'):
    #Instancia de la librería wxPython por medio del constructor de la clase App para crear un objeto que
    #funcione como la base de un GUI.
    app = wx.App(redirect= False)
    #Instancia de nuestra clase propia llamada Frame_With_Panel que fue creada en este mismo programa (frame se
    #refiere a la ventana del GUI) e incluye una instancia de la clase Panel para agregar un contenedor con
    #elementos dentro, el constructor vacío lo que hace es indicar que se cree y muestre la ventana.
    frame = MainFrame()
    #wxPython.App.MainLoop(): Método para que se ejecute en un loop infinito el GUI, logrando que no se
    #ejecute una vez y luego cierre por sí solo, sino que solo se cierre al dar clic en el tache del frame.
    app.MainLoop()
```

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 ListBox perteneciente a la librería **wxPython** 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 SpinCtrl perteneciente a la librería **wxPython**.
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 **FigureCanvasWxAgg** perteneciente a la librería **wxPython**.
10. Imprimir en consola todos los valores de tensión recabados de los n valores indicados en el SpinCtrl.
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:
import wx #wxPython: Librería para crear interfaces de usuario GUI (Graphycal User Interface)
#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 - FigureCanvasWxAgg: La clase FigureCanvasWxAgg proporcionada por la biblioteca Matplotlib en el
#módulo matplotlib.backends.backend_wxagg se utiliza para mostrar y manejar gráficos generados por Matplotlib
#dentro de una ventana o panel de wxPython. En este caso se utiliza para mostrar una gráfica que actualice
#sus datos en tiempo real mientras los vaya recopilando de una tarjeta de desarrollo Arduino.
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
import numpy as np #Librería numpy: 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.

#GUI CREADO CON PROGRAMACIÓN ORIENTADA A OBJETOS Y LA LIBRERÍA WX PYTHON QUE INCLUYE 2 PANELES (CONTENEDORES DE
#ELEMENTOS):
#   - PANEL 1: CUENTA CON UN ÁREA DE GRAFICACIÓN (FIGURE MATPLOTLIB) PARA ACTUALIZAR Y MOSTRAR EN TIEMPO REAL
#     (FIGURECANVASWXAGG) LOS DATOS DE TENSIÓN VS. TIEMPO RECOPILADOS DEL PIN ANALÓGICO A0 DE UN ARDUINO.
#   - PANEL 2: CUENTA CON UN LISTBOX QUE MUESTRA LOS PUERTOS DISPONIBLES DE CONEXIÓN (SERIAL), UN SPINCONTROL
#     NUMÉRICO QUE PERMITA INTRODUCIR EL NÚMERO DE MUESTRAS A RECOPILAR, UN BOTÓN DE START/STOP QUE PERMITA
#     EMPEZAR O DETENER EL MUESTREO Y UN BOTÓN DE SAVE QUE GUARDE LOS DATOS RECOPILADOS EN UN ARCHIVO DE EXCEL.
#Se agregarán dos paneles, que son contenedores donde se pueden incluir varios elementos (widgets) como botones,
#cuadros de texto, imágenes, etc. dentro de un frame.

#Cuando se genere un GUI que incluya un panel (contenedor) se debe crear el código en el siguiente órden:
# - clase Panel: El contenedor incluye un objeto que instancíe la clase de cada widget que se quiera incluir.
#       - Widget: Dentro del constructor de la clase Panel se crea un objeto de cada widget que se quiera
#         incluir en el contenedor, pero si es que alguno de estos elementos realiza una acción, fuera del
#         constructor pero dentro de la clase Panel, se debe crear una función que describa lo que realiza.
#         Los widgets que realizan acciones pueden ser: botones, checkboxes, áreas de texto, comboboxes,
#         radiobuttons, listboxes, ventanas de diálogo, gráficas, temporizadores, etc.
# - clase Frame: Dentro de la clase frame se declara su título y se instancía la clase panel para agregar el
```

```
#   contenedor previamente creado a la ventana.
# - método main: A través del método main se ejecuta la clase frame para mostrar y ejecutar la ventana del GUI.

#TopPanel: La clase recibe como parámetro un objeto de la clase Panel, que pertenece a la librería wxPython y
#representa un contenedor. El panel superior solamente incluye la gráfica que recopila y muestra datos en tiempo
#real, actualizandose al transcurrir del tiempo.
class TopPanel(wx.Panel):
    #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, parent):
        #super().__init__(parent): Lo que hace el método super() es heredar todos los métodos y atributos de la
        #clase padre. En este caso es necesario incluirlo porque el constructor de la clase Panel recibe como
        #parámetro al elemento parent.
        super().__init__(parent)

        #GRAFICACIÓN EN PYTHON:
        #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.
        #Este objeto se asigna al constructor de la clase Panel donde nos encontramos actualmente por medio de
        #la instrucción self.
        #Se declaran como self.nombreObjeto los widgets a los que sí se les vaya a extraer o introducir datos
        #en el transcurso del funcionamiento de la interfaz gráfica.
        self.figure = Figure()
        #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.
        self.axes = self.figure.add_subplot(111)
        #FigureCanvas(): La clase FigureCanvas permite mostrar gráficos generados por Matplotlib en una ventana
        #o panel de wxPython, permitiendo manejar eventos de interacción del usuario, como hacer zoom,
        #seleccionar puntos en el gráfico, etc:
        # - parent: Es el objeto padre al que se asociará el lienzo. Puede ser un objeto wx.Window o wx.Panel
        #   que actúa como contenedor del lienzo.
        # - id: Es el identificador numérico del lienzo. Puede ser especificado para identificar de manera única
        #   el lienzo dentro de la aplicación.
        #       - Se puede utilizar id = -1 para indicar que no se ha especificado ningún identificador único
        #         específico, ya que en el objeto FigureCanvasWxAgg a fuerza se debe declarar un id.
        # - figure: Es el objeto Figure de Matplotlib que contiene el gráfico que se desea mostrar en el lienzo.
        #   Es obligatorio pasar este parámetro.
        # - dpi: Especifica la resolución en puntos por pulgada (dots per inch) para el lienzo. El valor default
        #   es None, lo que significa que se utilizará la configuración de resolución predeterminada.
        # - size: Especifica el tamaño del lienzo en píxeles. Por defecto, es wx.DefaultSize, que permite que
```

```
#   el lienzo se ajuste automáticamente al tamaño del objeto padre.
# - name: Es el nombre del lienzo.
self.canvas = FigureCanvas(parent = self, id = -1, figure = self.figure)
#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.
self.axes.set_xlabel(xlabel = "Time [s]", fontname = "Consolas", fontsize = 8)
#matplotlib.Figure().add_subplot().set_xlabel(): Método para indicar el texto que aparece en el eje
#vertical de la gráfica, sus parámetros son los mismos que se describieron en set_xlabel.
self.axes.set_ylabel(ylabel = "Voltage [V]", fontname = "Consolas", fontsize = 8)
#matplotlib.Figure().add_subplot().set_ylim(): Método que indica los valores que abarca el eje vertical
#de la gráfica.
self.axes.set_ylim(-0.1, 5.1)
#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 FigureCanvasWxAgg.
self.canvas.draw()

#POSICIONAMIENTO DE ELEMENTOS:
#El siguiente código crea una instancia de la clase BoxSizer, la cual permite un posicionamiento
#relativo, colocando así un elemento respecto a otro, para poder usar esta clase el primer objeto se
#debe encontrar dentro del segundo. Si no se usa la clase BoxSizer, los elementos se colocarán unos
#encima de los otros.
#Al crear la Instancia de la clase BoxSizer perteneciente a la librería wxPython se le puede pasar como
#parámetro solamente dos posibles atributos para indicar la dirección de la posición del objeto:
# - wx.HORIZONTAL: Hace que la dirección de la alineación del primer objeto sea horizontal respecto al
#   segundo, esto se refiere a que se empiece indicar la posición del widget desde la esquina izquierda
#   dentro del contenedor.
# - wx.VERTICAL: Hace que la dirección de la alineación del primer objeto sea vertical respecto al
#   segundo, esto se refiere a que se empiece indicar la posición del widget desde la parte superior de
#   en medio, dentro del contenedor.
self.Sizer = wx.BoxSizer(wx.HORIZONTAL)
#Se declaran como self.nombreObjeto los widgets a los que sí se les vaya a extraer o introducir datos
#en el transcurso del funcionamiento de la interfaz gráfica.

#Ya se mencionó que la clase BoxSizer sirve al posicionar un elemento respecto a otro, para ello uno de
#los elementos se debe encontrar dentro del otro, con el objetivo de indicar cuál es el primer objeto
```

```
#(el que tiene posicionamiento relativo) y cuál es el segundo objeto (el que contiene al primer objeto),
#se utilizan los métodos .Add() y .SetSizer() de la siguiente forma:

#Instancia_BoxSizer.Add(): Método utilizado para agregar un elemento al sizer, sizer se refiere al
#elemento que contiene a otro que está colocado dentro de él con posicionamiento relativo, para ello
#dentro de su paréntesis se agregan los siguientes parámetros:
# - primer_parámetro: Con este parámetro se indica qué objeto que será agregado dentro del otro, el
#   objeto contenedor es llamado sizer.
# - proportion: Este parámetro determina cómo se asignará el espacio de todos los elementos que se
#   encuentran dentro del contenedor, aún si el sizer se expande o se reduce.
#           - proportion = 0: El elemento no crecerá ni se encogerá en relación a otros elementos
#             dentro del sizer.
#           - proportion = valor: El elemento se expandirá o se encogerá proporcionalmente en el sizer,
#             dependiendo del valor de los demás elementos.
#                   Por ejemplo, si hay dos elementos en el sizer, ambos con proportion = 1, cada uno
#                   ocupará la mitad del espacio disponible cuando el sizer se expanda. Si uno de los
#                   elementos tiene proportion = 2, ocupará dos tercios del espacio disponible, mientras
#                   que el otro elemento ocupará un tercio cuando el sizer se expanda.
# - flag: El parámetro flag se utiliza para especificar las opciones de posicionamiento y alineación
#   del elemento dentro del sizer por medio de banderas, las acciones de estas banderas se pueden
#   combinar usando la operación lógica OR (|), las flags que se pueden usar son descritas a
#   continuación:
#           - wx.EXPAND: Hace que el elemento se expanda para ocupar todo el espacio disponible en la
#             dirección del sizer.
#           - wx.ALL: Agrega un borde en todos los lados del elemento.
#           - wx.LEFT: Agrega un borde en el lado izquierdo del elemento.
#           - wx.RIGHT: Agrega un borde en el lado derecho del elemento.
#           - wx.TOP: Agrega un borde en la parte superior del elemento.
#           - wx.BOTTOM: Agrega un borde en la parte inferior del elemento.
#           - wx.CENTER: Centra el elemento dentro del espacio asignado por el sizer.
#           - wx.ALIGN_LEFT: Alinea el elemento a la izquierda dentro del espacio asignado por el sizer.
#           - wx.ALIGN_RIGHT: Alinea el elemento a la derecha dentro del espacio asignado por el sizer.
#           - wx.ALIGN_TOP: Alinea el elemento en la parte superior dentro del espacio asignado por el
#             sizer.
#           - wx.ALIGN_BOTTOM: Alinea el elemento en la parte inferior dentro del espacio asignado por
#             el sizer.
# - border: Establece el espacio en píxeles entre el elemento y los bordes del sizer.
self.Sizer.Add(self.canvas, proportion = 1,                  #Gráfica agregada al Sizer, proportion = 1
                        flag = wx.CENTER)                    #Elemento centrado.

#wx.Panel.SetSizer() = self.SetSizer(): Método aplicado al objeto de la clase Panel que recibe como
#parámetro esta clase, el cuál recibe como parámetro un objeto de la clase BoxSizer para indicar cuál es
#el elemento contenedor al que ya se han agregado anteriormente uno o más widgets posicionados
#relativamente con el método .Add().
self.SetSizer(self.Sizer)
```

```
    #función update_graph(): Método creado dentro de la clase propia llamada TopPanel que recibe como parámetro
    #los vectores que representan los ejes horizontal (x) y vertical (y) de la gráfica, para que de esta forma
    #se mantengan actualizados y se refresquen constantemente sus valores de tensión vs tiempo.
    def update_graph(self, x, y):
        #matplotlib.Figure.add_subplot().clear(): Método para limpiar el contenido de una gráfica.
        self.axes.clear()
        #matplotlib.Figure.add_subplot().set_xlabel(): Método para indicar el texto que aparece en el eje x.
        self.axes.set_xlabel(xlabel = "Time [s]", fontname = "Consolas", fontsize = 8)
        #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)
        #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, +: símbolo de más, .; puntos, v: Triángulos hacia abajo,
        #   h: Hexágono, etc.
        # - Tipo de Líneas: -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto,
        #   'or': Nada.
        self.axes.plot(x,y,'w.-') #'kv-' significa w: color blanco, .: Símbolo de puntos, -: línea sólida.
        #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('xkcd:aqua')
        #draw(): Método que actualiza y muestra los datos recopilados en tiempo real en la gráfica creada con el
        #objeto que instancía la clase FigureCanvasWxAgg.
        self.canvas.draw()



#BottomPanel: La clase recibe como parámetro un objeto de la clase Panel, que pertenece a la librería wxPython y
#representa un contenedor. El panel inferior incluye un listbox que muestra los puertos disponibles de conexión,
#un Spin control numérico que permite introducir el número de muestras a recopilar del Arduino, botónes de
#Start/Stop que empiezan o detienen el muestreo de datos y un Botón de Save para guardar los datos recopilados
#en un archivo de Excel.
class BottomPanel(wx.Panel):
    #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.
    #El parámetro top se declara en el constructor de la clase BottomPanel para que cuando se cree un objeto del
    #panel inferior en la clase Frame (ventana), se le tenga que pasar como parámetro un objeto que instancíe al
    #panel (contenedor) superior que contiene la gráfica que se actualiza en tiempo real.
    def __init__(self, parent, top):
```

```
        #super().__init__(parent): Lo que hace el método super() es heredar todos los métodos y atributos de la
        #clase padre. En este caso es necesario incluirlo porque el constructor de la clase Panel recibe como
        #parámetro al elemento parent.
        super().__init__(parent)
        self.graph = top #Objeto TopPanel (gráfica), obtenido como parámetro de esta misma clase Panel.

        #CREACIÓN DE LOS WIDGETS: Botón
        #Instancia de la librería wxPython por medio del constructor de la clase Button para crear un widget de
        #tipo botón, en este se deben indicar como parámetros:
        # - parent: Es el objeto padre al que se asociará el botón. Puede ser un objeto wx.Window o wx.Panel
        #   que actúa como contenedor.
        # - label = "": Con este parámetro se indica el texto que aparecerá sobre el botón.
        # - pos = (x, y): Con este atributo se indica la posición fija en pixeles del widget, 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.
        #       - Es importante mencionar que si después se utiliza la clase BoxSizer para posicionar los
        #         elementos de forma relativa, esta posición no es respetada.
        # - name = "": Con este parámetro se indica el nombre del botón.
        # - id: Es el identificador numérico del botón. Puede ser especificado para identificarlo de manera
        #   única en el programa.
        self.buttonStart = wx.Button(parent = self, label = "Start", pos = (330, 20))
        self.buttonStop = wx.Button(parent = self, label = "Stop", pos = (330, 20))
        self.buttonSaveData = wx.Button(parent = self, label = "Save", pos = (460, 20), name = "ButtonSaveData")
        #widget.Hide(): Método que sirve para esconder un widget en la GUI.
        self.buttonStop.Hide()          #Esconde inicialmente el botón de STOP, está en el mismo lugar que START
        self.buttonSaveData.Hide()      #Esconde inicialmente el botón de SAVE

        #CREACIÓN DE LOS WIDGETS: Temporizador
        #Instancia de la librería wxPython por medio del constructor de la clase Timer para crear un widget de
        #tipo temporizador, en este se pueden indicar los siguientes parámetros:
        # - owner (opcional): Especifica el objeto propietario del timer, para que cuando el temporizador genere
        #   eventos, estos se envíen al objeto propietario para su procesamiento.
        #       - Si no se proporciona ningún propietario, se puede establecer utilizando el método SetOwner().
        # - id (opcional): Especifica el identificador único del temporizador. Si no se proporciona, se generará
        #   automáticamente un identificador único.
        self.temporizador = wx.Timer(owner = self)

        #CREACIÓN DE LOS WIDGETS: Texto Estático
        #Instancia de la librería wxPython por medio del constructor de la clase StaticText para crear un widget
        #que muestre un texto estático en una ventana o panel, se le deben indicar los siguientes parámetros:
        # - parent: Es el objeto padre al que se asociará el StaticText.
        # - id: Un identificador único. Puede ser un número entero o el valor wx.ID_ANY para que wxPython
        #   seleccione automáticamente un identificador.
        # - label: Es el texto que se mostrará en el control StaticText.
        # - pos: Es una tupla que indica la posición inicial del StaticText dentro de su padre.
```

```
        # - size: Indica el tamaño de la caja de texto estático en píxeles (ancho, alto).
        #           - El valor -1 indica que la altura se ajustará automáticamente según el contenido de la caja
        #             de texto.
        # - style: Estilos que se aplicarán a la caja de texto para modificar su apariencia y comportamiento.
        #           - wx.ALIGN_CENTER: Centra el texto dentro de la caja de texto.
        #           - wx.ALIGN_LEFT: Alinea el texto a la izquierda dentro de la caja de texto.
        #           - wx.ALIGN_RIGHT: Alinea el texto a la derecha dentro de la caja de texto.
        # - name: El nombre de la caja de texto.
        self.labelPort = wx.StaticText(self,
                                label = "COM Port: ",   #Título del ListBox de los puertos disponibles.
                                pos = (60, 10))
        #wx.StaticText().SetForegroundColour(): Método para cambiar el color de letra de un widget StaticText,
        #recibe como parámetro el nuevo color de letra:
        # - wx.BLACK: Color negro.
        # - wx.WHITE: Color blanco.
        # - wx.RED: Color rojo.
        # - wx.GREEN: Color verde.
        # - wx.BLUE: Color azul.
        # - wx.CYAN: Color cian (mezcla de verde y azul).
        # - wx.MAGENTA: Color magenta (mezcla de rojo y azul).
        # - wx.YELLOW: Color amarillo.
        # - wx.LIGHT_GREY: Color gris claro.
        # - wx.DARK_GREY: Color gris oscuro.
        # - wx.LIGHT_BLUE: Color azul claro.
        # - wx.LIGHT_CYAN: Color cian claro.
        # - wx.LIGHT_GREEN: Color verde claro.
        # - wx.LIGHT_YELLOW: Color amarillo claro.
        # - wx.LIGHT_RED: Color rojo claro.
        # - wx.LIGHT_MAGENTA: Color magenta claro.
        # - wx.DARK_RED: Color rojo oscuro.
        # - wx.DARK_GREEN: Color verde oscuro.
        # - wx.DARK_BLUE: Color azul oscuro.
        self.labelPort.SetForegroundColour(wx.LIGHT_GREY)       #Letra gris claro.
        #Transformar el estilo de la letra en negritas, itálica, subrayada, etc: Para ello se debe hacer uso de:
        # - wx.StaticText().GetFont(): Método que obtiene el tipo de letra del widget StaticText.
        #   - wx.StaticText().GetFont().SetWeight(): Método que indica el peso de la letra, indicando si es
        #     ligera, normal o en negritas.
        #           - wx.FONTWEIGHT_NORMAL: Define el peso normal de la fuente.
        #           - wx.FONTWEIGHT_LIGHT: Define un peso de fuente más ligero que el normal.
        #           - wx.FONTWEIGHT_BOLD: Define un peso de fuente en negrita, entre otras.
        #   - wx.StaticText().GetFont().SetFamily(): Método que indica el tipo de fuente de la letra:
        #           - wx.FONTFAMILY_DEFAULT: Fuente predeterminada del sistema.
        #           - wx.FONTFAMILY_DECORATIVE: Fuente decorativa, como fuentes de estilo artístico.
        #           - wx.FONTFAMILY_ROMAN: Fuentes con estilo similar a las fuentes "serif" tradicionales.
        #           - wx.FONTFAMILY_SCRIPT: Fuentes de estilo manuscrito o de guión.
```

```
#           - wx.FONTFAMILY_SWISS: Fuentes "sans-serif" modernas y sin adornos.
#           - wx.FONTFAMILY_MODERN: Fuentes modernas y geométricas.
#           - wx.FONTFAMILY_TELETYPE: Fuentes de estilo de máquina de escribir o monoespaciadas.
# - wx.StaticText().SetFont(): Método que devuelve al widget el tipo de letra ya modificado.
font1 = self.labelPort.GetFont()
font1.SetWeight(wx.FONTWEIGHT_BOLD)                     #Letra en negritas.
font1.SetFamily(wx.FONTFAMILY_TELETYPE)                 #Letra tipo máquina de escribir.
font1.SetPointSize(10)                                  #Tamaño de letra = 10.
self.labelPort.SetFont(font1)

self.LabelSamples = wx.StaticText(self,
                        label = "Samples: ",    #Título del SpinCtrl del número de muestreos.
                        pos = (210, 10))
self.LabelSamples.SetForegroundColour(wx.LIGHT_GREY)    #Letra gris claro.
font2 = self.LabelSamples.GetFont()
font2.SetWeight(wx.FONTWEIGHT_BOLD)                     #Letra en negritas.
font2.SetFamily(wx.FONTFAMILY_TELETYPE)                 #Letra tipo máquina de escribir.
font1.SetPointSize(10)                                  #Tamaño de letra = 10.
self.LabelSamples.SetFont(font2)

#CREACIÓN DE LOS WIDGETS: List Box
#Instancia de la librería wxPython por medio del constructor de la clase ListBox para crear un widget
#que muestre una lista de elementos seleccionables en una ventana o panel, a continuación se describen
#los parámetros más comunes que puede recibir su constructor:
# - parent: El objeto que actúa como padre o contenedor del ListBox.
# - id: Un identificador único para el ListBox. Puede ser wx.ID_ANY para permitir que wxPython genere
#   automáticamente un identificador.
# - pos: La posición inicial del ListBox en la ventana. Puede ser un objeto wx.Point o una tupla
#   (x, y) que representa las coordenadas del punto.
# - size: Indica el tamaño del ListBox. Puede ser un objeto wx.Size o una tupla (width, height) que
#   represente el ancho y la altura.
# - choices: Es una lista o tupla de cadenas que representa los elementos de la lista que se mostrarán
#   en el ListBox.
# - style: Es un conjunto de estilos que controlan el comportamiento y la apariencia del ListBox.
#   Algunos de los estilos comunes son:
#           - wx.LB_SINGLE: Permite seleccionar un solo elemento a la vez.
#           - wx.LB_MULTIPLE: Permite seleccionar múltiples elementos manteniendo presionada la tecla
#             Ctrl o Shift.
#           - wx.LB_EXTENDED: Permite seleccionar múltiples elementos haciendo clic y arrastrando el
#             cursor.
#           - wx.LB_HSCROLL: Habilita la barra de desplazamiento horizontal si los elementos exceden el
#             ancho del ListBox.
#           - wx.LB_ALWAYS_SB: Muestra siempre la barra de desplazamiento, incluso si no es necesaria.
# - validator: Utiliza un objeto wx.Validator para validar las selecciones en el ListBox.
# - name: Un nombre opcional para el ListBox, que puede ser utilizado para identificarlo en la jerarquía
```

```
#   del control.
#El objeto ListBox ejecuta la función SerialPorts() de esta clase para obtener los puertos disponibles:
self.commPorts = wx.ListBox(parent = self, id = 3, pos = (60, 30), name = "Ports",
                        #En el parámetro choices se ejecuta la función propia SerialPorts() declarada
                        #fuera del constructor pero dentro de la clase BottomPanel para obtener los
                        #puertos disponibles del ordenador actual por medio de la clase serial.
                        choices = self.SerialPorts(),
                        style = wx.LB_ALWAYS_SB)

#CREACIÓN DE LOS WIDGETS: Spin Control
#Instancia de la librería wxPython por medio del constructor de la clase SpinCtrl para crear un widget
#que muestre un selector de números en el panel (contenedor), a continuación se describen los parámetros
#más comunes que puede recibir su constructor:
# - parent: Es el widget padre al que pertenecerá el control SpinCtrl.
# - id: Es el identificador único del control SpinCtrl. Puede ser especificado como wx.ID_ANY para que
#   wxPython asigne automáticamente un identificador único.
# - value: Es el string que inicialmente aparece en el SpinCtrl.
# - pos: Es la posición inicial del control SpinCtrl. Puede ser especificada utilizando un objeto
#   wx.Point o una tupla(x, y).
# - size: Es el tamaño del control SpinCtrl. Puede ser especificado utilizando un objeto wx.Size o una
#   tupla (width, height).
# - style: Es el estilo del control SpinCtrl, que determina su apariencia y comportamiento. Los estilos
#   se especifican utilizando constantes de la clase wx.SpinCtrl y se pueden combinar utilizando el
#   operador lógico OR |.
#           - wx.SP_ARROW_KEYS: Permite el uso de las teclas de flecha para incrementar o decrementar
#             el valor del control.
#           - wx.SP_WRAP: Permite que cuando el SpinCtrl supere su límite mínimo, inmediatamente después
#             se coloce en su límite máximo y viceversa.
#           - wx.SP_HORIZONTAL: Muestra los botones de incremento y decremento en posición horizontal.
#           - wx.SP_VERTICAL: Muestra los botones de incremento y decremento en posición vertical.
#           - wx.SP_NOBORDER: Elimina el borde alrededor del control SpinCtrl.
# - min: Es el valor mínimo permitido en el control SpinCtrl.
# - max: Es el valor máximo permitido en el control SpinCtrl.
# - initial: Es el valor inicial del control SpinCtrl. Este parámetro es equivalente al parámetro value,
#   y se incluye por razones de compatibilidad con versiones anteriores de wxPython.
# - name: Es el nombre del control SpinCtrl.
#El objeto SpinCtrl es un widget de la GUI que muestra números que van del 1 al 1000.
self.spinCtrlTime = wx.SpinCtrl(parent = self, value = "", pos = (210, 30), name = "wxSpinCtrlTime",
                        style = wx.SP_WRAP,
                        min = 1,
                        max = 1000,
                        initial = 1)

#Instancia_Widget.Bind(): Este método se utiliza para enlazar un evento a un controlador de eventos,
#indicando en su primer parámetro el evento que detona el método y en el segundo la función que
```

```
        #se ejecutará cuando ese evento ocurra. Normalmente las funciones que describen las acciones a
        #realizar por los elementos del Panel se encuentran dentro de esta misma clase, pero fuera de su
        #constructor.
        # - Tipos de Eventos en Python:
        #       - wx.EVT_BUTTON: Evento que se activa cuando se hace clic en un botón.
        #       - wx.EVT_TEXT: Evento que se activa cuando se cambia el contenido de un control de texto.
        #       - wx.EVT_CHECKBOX: Evento que se activa cuando se cambia el estado de una casilla de
        #         verificación.
        #       - wx.EVT_COMBOBOX: Evento que se activa cuando se selecciona un elemento de una lista
        #         desplegable (combobox).
        #       - wx.EVT_LISTBOX: Evento que se activa cuando se selecciona un elemento de una lista (listbox).
        #       - wx.EVT_RADIOBUTTON: Evento que se activa cuando se selecciona un botón de opción (radiobutton)
        #         en un grupo de botones de opción.
        #       - wx.EVT_MENU: Evento que se activa cuando se selecciona una opción de menú.
        #       - wx.EVT_CLOSE: Evento que se activa cuando se intenta cerrar una ventana o diálogo.
        #       - wx.EVT_KEY_DOWN y wx.EVT_KEY_UP: Evento que se activan cuando se presiona o se suelta una
        #         tecla del teclado, respectivamente.
        #       - wx.EVT_MOUSE_EVENTS: Son una serie de eventos relacionados con las interacciones del ratón,
        #         como clics, movimiento, etc. Estos eventos son descritos a continuación:
        #               - wx.EVT_LEFT_DOWN: Se activa cuando se presiona el botón izquierdo del ratón.
        #               - wx.EVT_LEFT_UP: Se activa cuando se suelta el botón izquierdo del ratón.
        #               - wx.EVT_LEFT_DCLICK: Se activa cuando se hace doble clic con el botón izquierdo del
        #                 ratón.
        #               - wx.EVT_RIGHT_DOWN: Se activa cuando se presiona el botón derecho del ratón.
        #               - wx.EVT_RIGHT_UP: Se activa cuando se suelta el botón derecho del ratón.
        #               - wx.EVT_RIGHT_DCLICK: Se activa cuando se hace doble clic con el botón derecho del
        #                 ratón.
        #               - wx.EVT_MIDDLE_DOWN: Se activa cuando se presiona el botón central del ratón.
        #               - wx.EVT_MIDDLE_UP: Se activa cuando se suelta el botón central del ratón.
        #               - wx.EVT_MIDDLE_DCLICK: Se activa cuando se hace doble clic con el botón central del
        #                 ratón.
        #               - wx.EVT_MOTION: Se activa cuando se mueve el ratón dentro del área del objeto
        #                 capturador.
        #               - wx.EVT_ENTER_WINDOW: Se activa cuando el ratón entra en el área del objeto
        #                 capturador.
        #               - wx.EVT_LEAVE_WINDOW: Se activa cuando el ratón sale del área del objeto capturador.
        #               - wx.EVT_MOUSEWHEEL: Se activa cuando se desplaza la rueda del ratón.
        #       - wx.EVT_TIMER: Es un evento específico del temporizador en wxPython. Este evento se activa
        #         cuando se cumple un intervalo de tiempo especificado
        #               - Cuando se usa este tipo de evento se debe incluir un tercer parámetro que indique
        #                 cual es el widget timer en específico que se está utilizando para provocar el evento.
        self.buttonStart.Bind(wx.EVT_BUTTON, self.OnStartClick)     #Clic en Botón Start = OnStartClick()
        self.buttonStop.Bind(wx.EVT_BUTTON, self.OnStopClick)       #Clic en Botón Stop  = OnStopClick()
        self.buttonSaveData.Bind(wx.EVT_BUTTON, self.OnStartSaving) #Clic en Botón Save  = OnStartSaving()
        #Cada que pasa un intervalo del timer, se ejecuta la función TimeInterval() y para ello se utiliza el
```

```
        #objeto temporizador, que es una instancia de la clase Timer.
        self.Bind(wx.EVT_TIMER, self.TimeInterval, self.temporizador)

        #ATRIBUTOS DEL CONSTRUCTOR PERTENECIENTE A LA CLASE BottomPanel:
        self.x = np.array([])           #Numpy array que crea el eje horizontal (x = tiempo [s]) de la gráfica.
        self.y = np.array([])           #Numpy array que crea el eje vertical (x = tensión [V]) de la gráfica.
        self.values = []                #Lista que guarda los valores de tiempo [s] y tensión [V] recopilados.

        self.samplingTime = 500         #Intervalo int del conteo del objeto Timer indicado en milisegundos.
        self.time = 0                   #Variable que cuenta cada 1 segundos el tiempo de ejecución de la GUI.

        self.highValueBoard = 5         #Valor de tensión Máxima = 5V
        self.boardResolution = 1023     #Resolución de 10 bits del ADC perteneciente al Arduino: (2^10)-1 = 1023

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

        self.serialConnection = False   #Variable booleana que indica si se ha establecido una conexión serial.
        self.stopAcquisition = False    #Variable booleana que indica si se ha presionado el botón de STOP.

        #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 ListBox.
        self.serialArduino = None       #Datos de tensión del obtenidos del pin A0 del Arduino.


    #función OnStartClick(): Método creado dentro de la clase propia llamada BottomPanel 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 ListBox 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, event):
        print("Start")
        #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.serialArduino != None):
            #myFile.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.
            self.serialArduino.close()
        #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.
```

```
        self.buttonStart.Hide()         #Esconde el botón de START, está en el mismo lugar que STOP.
        self.buttonSaveData.Hide()      #Esconde el botón de SAVE.
        #widget.Show(): Método que sirve para mostrar un widget en la GUI.
        self.buttonStop.Show()          #Muestra el botón de STOP.

        #REINICIO DE LAS VARIABLES: Como el botón de START se puede presionar una vez que se haya terminado de
        #recabar un número de datos y guardado en un archivo de Excel, para permitir que se vuelva a monitorear
        #los datos de tensión recibidos del pin A0 nuevamente, se debe reiniciar el estado de sus variables
        #cada que se dé clic en el botón de START.
        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 (x = tensión [V]).
        self.values = []                #Reinicio de la lista que guarda los valores de tiempo y tensión.

        self.n = 0                      #Reinicio de la variable que cuenta los datos recopilados por la GUI.
        self.time = 0                   #Reinicio de la variable que cuenta cada 1 segundo.

        self.serialConnection = False   #Reinicio de la variable booleana que indica si hay una conexión serial.
        self.stopAcquisition = False    #Reinicio de la variable booleana que indica si se ha presionado STOP.

        #DECLARACIÓN DE UNA NUEVA VARIABLE BOOLEANA: Además debemos ver si antes de dar clic en el botón de
        #START ya se ha seleccionado un Puerto serial con el que se comunique por medio del Arduino, para ello
        #es que se crea una nueva variable que denote eso.
        takeData = False                #Variable booleana para ver si se ha seleccionado una opción del LisBox.

        #Setters y Getters: Como los valores de los atributos están encapsulados por haber sido creados dentro
        #del constructor de la clase, existen dos tipos de métodos especiales para poder editar o extraer el
        #valor de estas variables de clase:
        #   - nombre_Atributo.SetValue(): Método setter que permite editar el valor de un atributo.
        #   - nombre_Atributo.GetValue(): Método getter que permite obtener el valor de un atributo.
        #wx.ListBox.GetSelection(): Método aplicado a un objeto perteneciente a la clase ListBox de la librería
        #wxPython que sirve para devolver un entero que represente el índice de la opción seleccionada en el
        #ListBox, si no hay ninguna opción seleccionada en la lista, el método GetSelection() devolverá -1 para
        #indicar que no hay selección.
        portVal = self.commPorts.GetSelection()

        #Condicional if que checa si se ha seleccionado un puerto del ListBox:
        # - Si no se ha seleccionado ningún elemento del LitBox se indica que la variable booleana takeData
        #   tenga valor False y se ejecuta una función diferida que muestre una ventana que indique que no hay
        #   ningún puerto seleccionado.
        # - Si ya se ha seleccionado un puerto en el ListBox se obtiene el string que indique cual puerto fue
        #   seleccionado y a la variable booleana takeData se le asigna un valor de True.
        if (portVal == -1):             #GetSelection() = -1, si no se ha seleccionado nada en el ListBox.
            takeData = False            #Variable booleana para ver si se ha seleccionado una opción del LisBox.
            #Instancia de la librería wxPython por medio del constructor de la clase CallLater para crear un
            #objeto que haga referencia a una función diferida, donde una "función diferida" se refiere a una
```

```
        #función que se programa para ser llamada después de un cierto período de tiempo especificado. En
        #lugar de ejecutarse de inmediato, la función se coloca en una cola de tareas y se ejecutará una vez
        #que haya transcurrido el tiempo especificado.
        # - millis: Especifica el tiempo en milisegundos después del cual se ejecutará la función. Puede ser
        #   un entero o un valor de coma flotante.
        # - callableObj: Es la función que se va a llamar después de que expire el tiempo especificado.
        #   Puede ser una función propia definida por nosotros o una función predefinida de Python.
        #ShowMessagePort(): Ventana emergente que indica que no hay ningún puerto seleccionado.
        wx.CallLater(millis = 10, callableObj = self.ShowMessagePort)
    else:
        #Setters y Getters: Como los valores de los atributos están encapsulados por haber sido creados
        #dentro del constructor de la clase, existen dos tipos de métodos especiales para poder editar o
        #extraer el valor de estas variables de clase:
        #   - nombre_Atributo.SetValue(): Método setter que permite editar el valor de un atributo.
        #   - nombre_Atributo.GetValue(): Método getter que permite obtener el valor de un atributo.
        #wx.ListBox.GetString(): Método aplicado a un objeto perteneciente a la clase ListBox de la librería
        #wxPython que sirve para devolver el string perteneciente a un ListBox que se encuentre en cierta
        #posición (índice) indicada como parámetro del método.
        portSelected = self.commPorts.GetString(portVal)
        takeData = True             #Variable booleana para ver si se ha seleccionado una opción del LisBox.

    #Condicional que checa si todavía no existe una comunicación serial, pero si se ha seleccionado ya del
    #ListBox incluido en el GUI a qué puerto serial nos queremos conectar.
    if (self.serialConnection == False and takeData == True):
        #INICIACIÓN DEL TEMPORIZADOR:
        #wx.Timer.Start(intervalo): Método que inicia el conteo de un temporizador, con su intervalo de
        #conteo indicado en milisegundos. El parámetro oneShot indica si el temporizador debe ejecutarse una
        #sola vez (True) o repetidamente (False, valor predeterminado).
        self.temporizador.Start(self.samplingTime, oneShot = False)     #Inicialización del temporizador.
        #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.serialArduino = serial.Serial(port = str(portSelected), 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)
            #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.serialArduino.reset_input_buffer()     #Limpiar el buffer de la comunicación serial.
            self.serialConnection = True    #Variable booleana que indica que si hay una conexión serial.
        except:
            self.serialConnection = False   #Variable booleana que indica que no hay una conexión serial.
            #Instancia de la librería wxPython por medio del constructor de la clase CallLater para crear un
            #objeto que haga referencia a una función diferida, donde una "función diferida" se refiere a
            #una función que se programa para ser llamada después de un cierto período de tiempo
            #especificado. En lugar de ejecutarse de inmediato, la función se coloca en una cola de tareas y
            #se ejecutará una vez que haya transcurrido el tiempo especificado.
            # - milliseconds: Especifica el tiempo en milisegundos después del cual se ejecutará la función.
            #   Puede ser un entero o un valor de coma flotante.
            # - callable: Es la función que se va a llamar después de que expire el tiempo especificado.
            #   Puede ser una función propia definida por nosotros o una función predefinida de Python.
            #ShowMessageError(): Ventana emergente que indica un error al intentar establecer la
            #comunicación serial.
            wx.CallLater(50, self.ShowMessageError)
```

```
#función ShowMessagePort(): Método creado dentro de la clase propia llamada BottomPanel que recibe como
#parámetro a la instancia del objeto Panel que recibe esta misma clase como parámetro, por eso se usa la
#palabra reservada self. Esta función es llamada en el condicional que checa si se ha seleccionado un puerto
#del ListBox, donde luego de haber confirmado que no se ha elegido una opción del ListBox, a través de un
#objeto de la clase CallLater se manda a llamar una función diferida que muestre una ventana emergente que
#muestre el mensaje: No COM Port Selected.
def ShowMessagePort(self):
    #wx.MessageBox(): Método de la librería wxPython 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 o cancelar:
    # - parent: El objeto padre al que se asociará la ventana de diálogo. Puede ser None para indicar que no
    #   tiene padre.
    # - message: Indica el mensaje que se mostrará en la ventana de diálogo.
    # - caption: Indica el título o encabezado de la ventana de diálogo.
    # - style: Declara el estilo de la ventana de diálogo, que determina qué botones se muestran y cómo se
    #   comportan, los estilos se pueden combinar usando la operación lógica OR (|):
    #           - wx.OK: Muestra un botón "OK" para cerrar la ventana de diálogo.
    #           - wx.YES_NO: Muestra botones "Sí" y "No".
    #           - wx.YES_DEFAULT: Hace que el botón "Sí" sea el predeterminado.
    #           - wx.NO_DEFAULT: Hace que el botón "No" sea el predeterminado.
    #           - wx.CANCEL: Muestra un botón "Cancelar".
    #           - wx.OK_CANCEL: Muestra botones "OK" y "Cancelar".
    #           - wx.YES_NO_CANCEL: Muestra botones "Sí", "No" y "Cancelar".
    #           - wx.ICON_NONE: No muestra ningún icono.
    #           - wx.ICON_EXCLAMATION: Muestra un icono de advertencia.
    #           - wx.ICON_HAND: Muestra un icono de error.
    #           - wx.ICON_ERROR: Sinónimo de wx.ICON_HAND.
    #           - wx.ICON_QUESTION: Muestra un icono de pregunta.
    #           - wx.ICON_INFORMATION: Muestra un icono de información.
    #           - wx.ICON_WARNING: Sinónimo de wx.ICON_EXCLAMATION.
    #           - wx.ICON_STOP: Sinónimo de wx.ICON_HAND.
    #           - wx.STAY_ON_TOP: Hace que la ventana de diálogo se mantenga siempre en la parte superior.
    # - pos: Es la posición inicial del control SpinCtrl. Puede ser especificada utilizando un objeto
    #   wx.Point o una tupla(x, y).
    # - size: Es el tamaño del control SpinCtrl. Puede ser especificado utilizando un objeto wx.Size o una
    #   tupla (width, height).
    wx.MessageBox(message = 'No COM Port Selected', caption = 'Serial Communication',
                style = wx.OK | wx.ICON_EXCLAMATION)
    #widget.Show(): Método que sirve para mostrar un widget en la GUI.
    self.buttonStart.Show()          #Muestra el botón de STOP.


#función ShowMessagePort(): Método creado dentro de la clase propia llamada BottomPanel que recibe como
#parámetro a la instancia del objeto Panel que recibe esta misma clase como parámetro, por eso se usa la
#palabra reservada self. Esta función es llamada en el condicional que checa si se ha seleccionado un puerto
```

```
#del ListBox, donde luego de haber confirmado que no se pudo establecer una comunicación serial con la placa
#de desarrollo Arduino, a través de un objeto de la clase CallLater se manda a llamar una función diferida
#que muestre una ventana emergente que muestre el mensaje: Communication with the board failed.
def ShowMessageError(self):
    #wx.MessageBox(): Método de la librería wxPython 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 o cancelar:
    # - parent: El objeto padre al que se asociará la ventana de diálogo. Puede ser None para indicar que no
    #   tiene padre.
    # - message: Indica el mensaje que se mostrará en la ventana de diálogo.
    # - caption: Indica el título o encabezado de la ventana de diálogo.
    # - style: Declara el estilo de la ventana de diálogo, que determina qué botones se muestran y cómo se
    #   comportan, los estilos se pueden combinar usando la operación lógica OR (|):
    #           - wx.OK: Muestra un botón "OK" para cerrar la ventana de diálogo.
    #           - wx.YES_NO: Muestra botones "Sí" y "No".
    #           - wx.YES_DEFAULT: Hace que el botón "Sí" sea el predeterminado.
    #           - wx.NO_DEFAULT: Hace que el botón "No" sea el predeterminado.
    #           - wx.CANCEL: Muestra un botón "Cancelar".
    #           - wx.OK_CANCEL: Muestra botones "OK" y "Cancelar".
    #           - wx.YES_NO_CANCEL: Muestra botones "Sí", "No" y "Cancelar".
    #           - wx.ICON_NONE: No muestra ningún icono.
    #           - wx.ICON_EXCLAMATION: Muestra un icono de advertencia.
    #           - wx.ICON_HAND: Muestra un icono de error.
    #           - wx.ICON_ERROR: Sinónimo de wx.ICON_HAND.
    #           - wx.ICON_QUESTION: Muestra un icono de pregunta.
    #           - wx.ICON_INFORMATION: Muestra un icono de información.
    #           - wx.ICON_WARNING: Sinónimo de wx.ICON_EXCLAMATION.
    #           - wx.ICON_STOP: Sinónimo de wx.ICON_HAND.
    #           - wx.STAY_ON_TOP: Hace que la ventana de diálogo se mantenga siempre en la parte superior.
    # - pos: Es la posición inicial del control SpinCtrl. Puede ser especificada utilizando un objeto
    #   wx.Point o una tupla(x, y).
    # - size: Es el tamaño del control SpinCtrl. Puede ser especificado utilizando un objeto wx.Size o una
    #   tupla (width, height).
    wx.MessageBox('Communication with the board failed', 'Communication error',
                  wx.OK | wx.ICON_ERROR)
    #widget.Show(): Método que sirve para mostrar un widget en la GUI.
    self.buttonStart.Show()          #Muestra el botón de STOP.


#función OnStartClick(): Método creado dentro de la clase propia llamada BottomPanel 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
#TimeInterval(), deteniendo la ejecución del temporizador y logrando así que se detenga la recopilación de
```

```
    #datos.
    def OnStopClick(self, event):
        print("Stop")
        #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.buttonStop.Hide()          #Esconde el botón de STOP, está en el mismo lugar que START.
        self.stopAcquisition = True     #Variable booleana que indica si se ha presionado el botón de STOP.


    #función TimeInterval(): Método creado dentro de la clase propia llamada BottomPanel que recibe como
    #parámetro el evento del temporizador que lo activa, para posteriormente ejecutar cierta acción.
    #En este caso el evento es activado cada vez que transcurre el intervalo de tiempo definido en el
    #temporizador y lo que hace es esconder los botones de START y SAVE, mostrar el botón de STOP y checar el
    #estado de la variable booleana serialConnection para ver si es que el número de datos muestreados es menor
    #al indicado en el SpinCtrl, si este es el caso se recopilan los datos de tensión del puerto A0, se
    #convierten de datos binarios que van de 0 a 1023 a valores de tensión de 0 a 5V, para posteriormente
    #guardarse en 3 vectores, uno que represente el eje x de la gráfica (tiempo [s]), otro que represente el
    #eje y de la gráfica (tensión [V]) y uno tercero que guarde ambos valores para que se puedan almacenar en
    #un archivo Excel al usar la función OnStartSaving().
    def TimeInterval(self, event):
        #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.buttonStart.Hide()         #Esconde el botón de START, está en el mismo lugar que STOP.
        self.buttonSaveData.Hide()      #Esconde el botón de SAVE
        #widget.Show(): Método que sirve para mostrar un widget en la GUI.
        self.buttonStop.Show()          #Muestra el botón de STOP.

        #Condicional if que checa si la conexión serial se ha establecido correctamente, si es así obtiene el
        #valor del objeto SpinCtrl para saber cuál es el número de muestreos del Arduino que se busca recopilar.
        if (self.serialConnection == True):
            #Setters y Getters: Como los valores de los atributos están encapsulados por haber sido creados
            #dentro del constructor de la clase, existen dos tipos de métodos especiales para poder editar o
            #extraer el valor de estas variables de clase:
            #   - nombre_Atributo.SetValue(): Método setter que permite editar el valor de un atributo.
            #   - nombre_Atributo.GetValue(): Método getter que permite obtener el valor de un atributo.
            #int(): Método que convierte un tipo de dato cualquiera en un numérico entero.
            samples = int(self.spinCtrlTime.GetValue()) #Obtiene el valor del objeto SpinCtrl.

            #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
            print("Se recopilarán ", samples, " datos.")

            #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.serialArduino.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("\r\n", "")


    #CONVERSIÓN DE NUMEROS BINARIOS NUMÉRICOS DE TENSIÓN A VALORES DE TENSIÓN REALES:
```

```
            #float(): Método que convierte un tipo de dato cualquiera en número 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.highValueBoard)/self.boardResolution)

            #IMPRIMIR EN CONSOLA TIEMPO Y TENSIÓN:
            printConsole = str(self.n) + ".- Time: " + str(self.time) + " (s)"+"\t" #Tiempo [segundos].
            printConsole += "Voltage: " + str(value) + " (V)"                       #Tensión [V].
            print(printConsole)                                     #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) + "," + str("{0:.3f}".format(value)))
            self.x = np.append(self.x,self.time)                    #Vector tiempo [segundos].
            self.y = np.append(self.y,value)                        #Vector tensión [V].

            #TopPanel.update_graph(): Método creado dentro de la clase propia llamada TopPanel que recibe
            #como parámetro dos vectores que representan los ejes horizontal (x) y vertical (y) del gráfico,
            #para que de esta forma se puedan refrescar y mantener actualizados constantemente los valores
            #de tensión vs tiempo en la gráfica de forma dinámica.
            self.graph.update_graph(self.x, self.y)
        except:
            print("Ocurrió un error al leer los datos de tensión del pin A0 en el Arduino")
            #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)
            self.stopAcquisition = True    #Variable booleana que indica si se ha presionado STOP.

        #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 delay que se declarará
        #después del except debe ser el mismo.
        self.time = self.time + 1       #Variable intermedia que cuenta el tiempo de ejecución del programa.
        self.n = self.n + 1             #Variable que cuenta los datos recopilados por la GUI.

        #Condicional if que checa si se ha llegado al límite impuesto por el número de muestreos indicados
        #en el SpinCtrl, 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.n >= samples or self.stopAcquisition == 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.buttonStop.Hide()  #Esconde el botón de START, está en el mismo lugar que STOP.
            #widget.Show(): Método que sirve para mostrar un widget en la GUI.
            self.buttonStart.Show() #Muestra el botón de START.
            self.buttonSaveData.Show()  #Muestra el botón de SAVE.

            #wx.Timer.Stop(): Método que detiene el conteo de un temporizador previamente empezado con el
            #método Start().
            self.temporizador.Stop()

            #Al cambiar el valor de la variable self.serialConnection se evita que se sigan recopilando
            #datos del pin A0 en el Arduino.
            self.serialConnection = False   #Variable booleana que indica que no hay una conexión serial.


#función OnStartSaving(): Método creado dentro de la clase propia llamada BottomPanel 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 TimeInterval().
def OnStartSaving(self, event):
    print("Save")
    #Instancia de la librería wxPython por medio del constructor de la clase FileDialog para ejecutar una
    #ventana de selección de archivos. A través de este cuadro de diálogo, los usuarios pueden seleccionar
    #un archivo.
    # - parent: Si se le pasa un valor None a este argumento lo que es significa que no hay un padre
    #   específico para el cuadro de diálogo.
    # - message: Con este parámetro se indica el texto que aparecerá sobre el explorador de archivos.
    # - defaultDir: Es una cadena de texto que especifica el directorio inicial en el que se abre el
    #   explorador de archivos. Si se deja en blanco (""), el diálogo de archivo se abrirá en el último
    #   directorio utilizado.
    # - defaultFile: Es una cadena de texto que representa el nombre de archivo inicial que se muestra en
    #   el explorador de archivos. Puede ser útil si se desea que se muestre un nombre de archivo
    #   predeterminado para el usuario.
    # - wildcard: Define el tipo de archivos que se pueden seleccionar, esto se debe ingresar como un
    #   string.
    # - style: Indica el estilo o comportamiento de la ventana de selección de archivos. Puede tomar varios
    #   valores para personalizar la apariencia y la funcionalidad del cuadro de diálogo.
    #       - wx.FD_OPEN: Este es el estilo predeterminado y se utiliza para permitir al usuario
    #         seleccionar un archivo existente. El explorador mostrará los archivos y directorios en la
    #         ubicación especificada.
    #       - wx.FD_SAVE: Este estilo se utiliza cuando se desea permitir al usuario seleccionar una
    #         ubicación para guardar un archivo nuevo. El explorador mostrará los archivos y directorios
    #         en la ubicación especificada y proporcionará una opción para ingresar un nombre de archivo.
```

```
        #       - wx.FD_OVERWRITE_PROMPT: Este estilo se utiliza en combinación con wx.FD_SAVE y muestra una
        #         advertencia si el archivo seleccionado ya existe. El cuadro de diálogo mostrará un mensaje
        #         preguntando al usuario si desea sobrescribir el archivo existente. las acciones de estos
        #         estilos se pueden combinar usando la operación lógica OR (|).
        #       - wx.FD_MULTIPLE: Este estilo permite la selección de múltiples archivos. En lugar de
        #         seleccionar un solo archivo, el usuario puede seleccionar varios archivos a la vez
        #         utilizando la tecla de modificación apropiada (como Ctrl o Shift).
        #       - wx.FD_CHANGE_DIR: Este estilo indica que el cuadro de diálogo debe cambiar el directorio de
        #         trabajo actual según la ubicación seleccionada por el usuario. Esto puede ser útil si desea
        #         cambiar automáticamente el directorio de trabajo al directorio del archivo seleccionado.
        #       - wx.FD_PREVIEW: Este estilo muestra una vista previa del archivo seleccionado en el cuadro de
        #         diálogo, si es posible. La vista previa puede ser una imagen, un documento o cualquier tipo
        #         de archivo que pueda ser mostrado en el cuadro de diálogo.
        fdlg = wx.FileDialog(parent = self, message = "Save your electric data",
                             defaultDir= "",
                             defaultFile = "",
                             wildcard = "CSV files(*.csv)|*.*", #Tipo de archivo aceptado = Excel .csv
                             style = wx.FD_SAVE)               #Permite guardar un archivo.
        #El objeto dialog que instancía la clase FileDialog perteneciente a la librería wxPython cuenta con los
        #siguientes métodos.
        # - dialog: Objeto que permite mostrar cuadros de diálogo.
        #   - dialog.ShowModal(): Muestra el cuadro de diálogo y bloquea la ejecución del programa hasta que el
        #     usuario seleccione un archivo o cierre el cuadro de diálogo, este método devuelve un código que
        #     indica el resultado de la interacción del usuario con el cuadro de diálogo, como lo son:
        #         - wx.ID_OK: Indica que el usuario ha seleccionado y confirmado una opción en el explorador de
        #           archivos.
        #         - wx.ID_CANCEL: Indica que el usuario ha cancelado el cuadro de diálogo sin seleccionar ningún
        #           archivo. Puede ocurrir si el usuario hace clic en el botón "Cancelar" o si cierra el
        #           explorador de archivos.
        #         - wx.ID_YES: Indica que el usuario ha confirmado la selección en el cuadro de diálogo. Este
        #           valor puede ser devuelto si se utiliza wx.FD_SAVE en el estilo del cuadro de diálogo y el
        #           usuario decide sobrescribir un archivo existente.
        #         - wx.ID_NO: Indica que el usuario ha rechazado la selección en el cuadro de diálogo. Esto
        #           puede ocurrir si se utiliza wx.FD_SAVE en el estilo del cuadro de diálogo y el usuario
        #           decide no sobrescribir un archivo existente.
        #         - wx.ID_APPLY: Este valor puede ser utilizado para realizar alguna acción adicional después de
        #           seleccionar un archivo en el cuadro de diálogo. No está directamente relacionado con la
        #           selección del archivo en sí, y su uso depende de la implementación específica.
        #   - dialog.GetPath(): Devuelve la ruta completa del archivo seleccionado por el usuario en el cuadro
        #     de diálogo. La ruta incluirá tanto el directorio como el nombre del archivo.
        #   - dialog.GetPaths(): Si el cuadro de diálogo permite la selección múltiple de archivos (configurado
        #     con wx.FD_MULTIPLE en el argumento style), este método devuelve una lista de las rutas completas
        #     de los archivos seleccionados.
        #   - dialog.GetDirectory(): Devuelve el directorio seleccionado por el usuario en el cuadro de diálogo.
        #     Si el usuario ha seleccionado múltiples archivos de diferentes directorios, esta función devuelve
```

```
        #     el directorio del primer archivo seleccionado.
        #   - dialog.GetFilenames(): Devuelve una lista de los nombres de archivo seleccionados por el usuario
        #     en el cuadro de diálogo. Los nombres de archivo no incluyen la ruta del directorio.
        if (fdlg.ShowModal() == wx.ID_OK):
            #Setters y Getters: Como los valores de los atributos están encapsulados por haber sido creados
            #dentro del constructor de la clase, existen dos tipos de métodos especiales para poder editar
            #o extraer el valor de estas variables de clase:
            #   - nombre_Atributo.SetValue(): Método setter que permite editar el valor de un atributo.
            #   - nombre_Atributo.GetValue(): Método getter que permite obtener el valor de un atributo.
            #Ya que se haya guardado el archivo, se obtiene su path para que ahora se le introduzcan los datos
            #recopilados y guardados en el vector (lista) values.
            self.savePath = fdlg.GetPath()+".csv"
            #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:
                #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.
                myFile = open(self.savePath,'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.
                myFile.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)):
                    myFile.write(self.values[i]+"\n")
                #myFile.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.
                myFile.close()
            except:
                print("No se pudo guardar los datos recopilados en un archivo Excel")



    #función OnStartSaving(): Método creado dentro de la clase propia llamada BottomPanel que sirve para
```

```
#rellenar los elementos del ListBox 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.
```

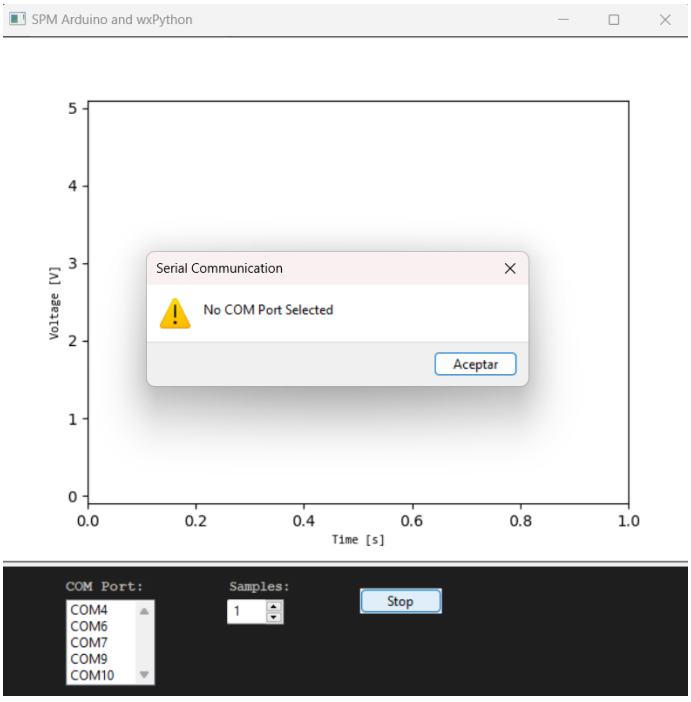
```
#               - Si el número total de bits es par, se deja el bit de paridad en 0.
#                   - Por ejemplo, supongamos que se desea transmitir el byte 11010110. El bit
#                     de paridad en la transmisión de la comunicación se calcularía contando el
#                     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.decode(): 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)
```

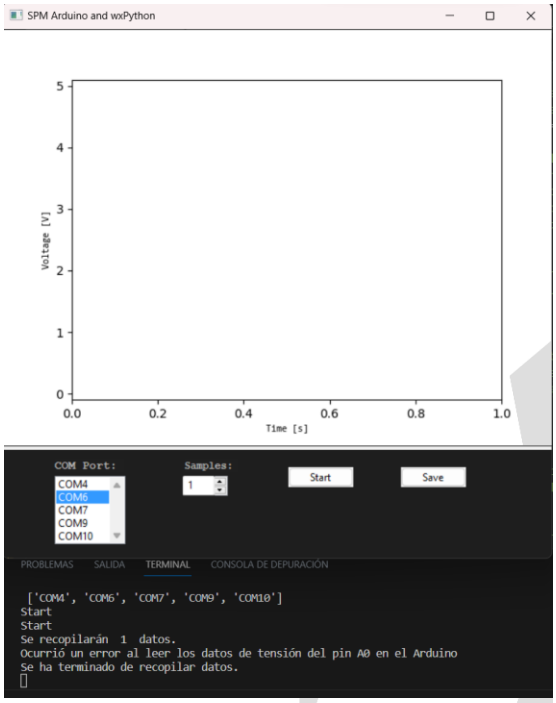
```
            #Para identificar el tipo de excepción que ha ocurrido y utilizarlo en la instrucción except, se
            #puede utilizar la clase Exception, que es una clase incorporada en Python utilizada para describir
            #todos los tipos de excepciones, luego de colocar el nombre de la clase Exception se usa la palabra
            #reservada "as" seguida de un nombre de variable, esto te 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.
                #Luego, __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.

        #Se devuelve la variable result, que es una lista como se mencionó en la anotación de la función.
        print("Los puertos encontrados a los que se pudo conectar el programa fueron: \n", result)
        return result



#MainFrame: La clase hereda de la clase Frame que pertenece a la librería wxPython, representa la ventana del
#GUI y crea una instancia de la clase BottomPanel donde a su vez ya está incluida la clase TopPanel para
#agregar dentro de la ventana un contenedor con elementos dentro y otro contenedor con otros elementos dentro.
class MainFrame(wx.Frame):
    def __init__(self):
        #Dentro del constructor de la GUI se declara una instancia de la librería wxPython por medio de la cual
        #se accede al constructor de la clase Frame, que representa la ventana del GUI, a dicha ventana se le
        #asigna un título por medio de su segundo parámetro y un tamaño a través de su tercer parámetro.
        super().__init__(None, title = "SPM Arduino and wxPython", size = (650, 650))

        #Instancia de la librería wxPython por medio del constructor de la clase SplitterWindow, usada para
        #dividir el espacio de una ventana en dos paneles ajustables, con una barra divisoria (splitter) que se
        #puede arrastrar para cambiar el tamaño de los paneles. Es útil cuando se desea tener una interfaz de
        #usuario con dos áreas separadas que pueden contener diferentes contenidos o widgets.
        # - parent: Es el objeto padre al que se asociará el widget. Puede ser un objeto wx.Panel o wx.Frame
        splitter = wx.SplitterWindow(self)

        #Instancia de la clase TopPanel para agregar el panel al SplitterWindow, osea el contenedor que divide
        #en varias partes el espacio de un Frame (ventana) en paneles ajustables, por ello es que se le
        #pasa como parámetro al objeto SplitterWindow.
```

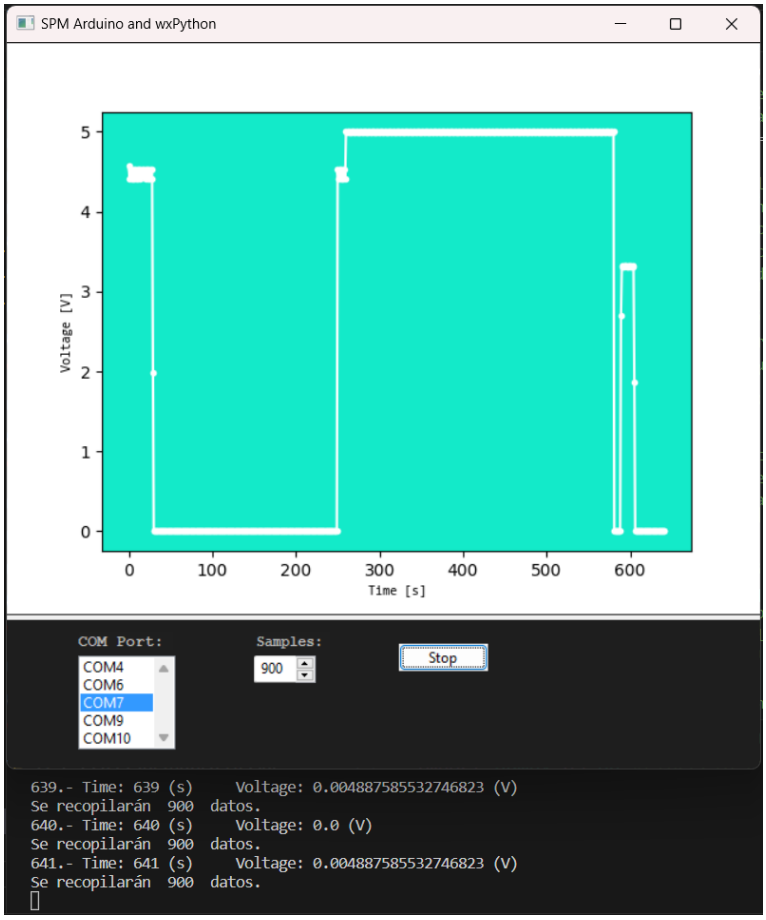
```
        top = TopPanel(splitter)

        #Instancia de la clase BottomPanel para agregar el panel al SplitterWindow, osea el contenedor que
        #divide en varias partes el espacio de un Frame (ventana) en paneles ajustables, por ello es que se
        #le pasa como parámetro al objeto SplitterWindow y a la instancia del panel TopPanel, porque además
        #dentro del constructor de la clase BottomPanel se indicó que se le debe pasar como parámetro un
        #objeto del panel superior.
        bottom = BottomPanel(splitter, top)

        #wx.SplitterWindow().SplitHorizontally(): Método que se utiliza para dividir el área del SplitterWindow
        #en dos paneles verticalmente, colocando de forma horizontal la línea de separación entre ambos:
        # - window1: El primer objeto ventana o panel que se colocará en la parte superior de la división
        #   horizontal.
        # - window2: El segundo objeto ventana o panel que se colocará en la parte inferior de la división
        #   horizontal.
        # - sashPosition: (Opcional) Indica la posición inicial de la barra de separación. Se especifica como
        #   un valor entero que representa la posición en píxeles desde el borde superior del SplitterWindow.
        #   Si no se proporciona un valor, se utiliza una posición predeterminada.
        # - sashPercent: (Opcional) El tamaño relativo de los paneles expresado como un porcentaje. Por ejemplo,
        #   un valor de 70 indica que el primer panel ocupará el 70% de la altura total del SplitterWindow,
        #   mientras que el segundo panel ocupará el 30% restante. Si no se proporciona un valor, se utiliza un
        #   tamaño predeterminado.
        # - reCalc: (Opcional) E un valor booleano que indica si se debe recalcular automáticamente los tamaños
        #   de los paneles después de la división. El valor predeterminado es True.
        splitter.SplitHorizontally(window1 = top, window2 = bottom, sashPosition = 482)

        #wx.SplitterWindow().SetMinimumPaneSize(): Método que se utiliza para establecer el tamaño mínimo
        #permitido para los paneles contenidos en el SplitterWindow.
        # - minSize: representa el tamaño mínimo en píxeles que se permite para los paneles. Este valor puede
        #   ser un entero o una tupla de dos enteros que representan el ancho mínimo y el alto mínimo
        #   respectivamente. El nombre del parámetro no se indica u obtendré un error de compilación.
        splitter.SetMinimumPaneSize(450)

        #wx.Frame.SetBackgroundColour() = self.SetBackgroundColour(): Método aplicado al objeto de la clase
        #Frame que recibe como parámetro el constructor de esta clase, utilizado para cambiar el color del fondo
        #de la ventana del GUI, el método realiza esto recibiendo como parámetro un objeto de la clase Colour,
        #perteneciente a la librería wxPython, indicando el color en formato RGB:
        #  - wx.Colour(R, G, B): Los valores de RGB van de 0 a 255 y su combinación de colores rojo, verde y
        #    azul crean cualquier color existente, el valor (0, 0, 0) corresponde al color negro y
        #    (255, 255, 255) al blanco.
        self.SetBackgroundColour(wx.Colour(30, 30, 30)) #Color de fondo negro no tan obscuro.

        #wx.Frame.show() = self.show(): Método aplicado al objeto de la clase Frame que recibe como
        #parámetro el constructor de esta clase para mostrar la ventana del GUI.
        self.Show()
```

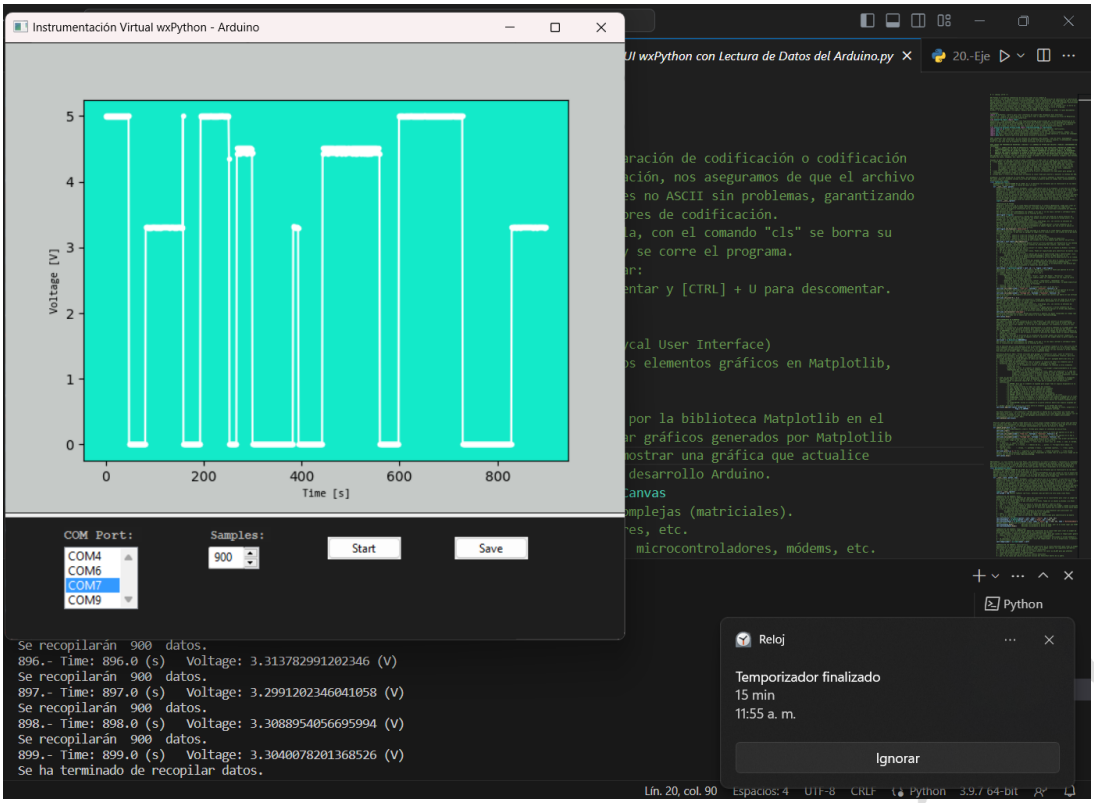
```python
#__name__ == __main__: Método main, esta función es super importante ya que sirve para instanciar las clases del
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.
if (__name__ == "__main__"):
    #Instancia de la librería wxPython por medio del constructor de la clase App para crear un objeto que
    #funcione como la base de un GUI.
    app = wx.App(redirect=False)
    #Instancia de nuestra clase propia llamada MainFrame que fue creada en este mismo programa (frame se
    #refiere a la ventana del GUI) e incluye una instancia de la clase Panel para agregar un contenedor con
    #elementos dentro, el constructor vacío lo que hace es indicar que se cree y muestre la ventana.
    frame = MainFrame()
    #Instancia_myFrame.SetPosition(): Método utilizado para indicar la posición inicial de la ventana dentro de
    #la pantalla del ordenaror, este método recibe como parámetro un objeto de la clase Point, perteneciente a
    #la librería wxPython:
    # - wx.Point(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.
    frame.SetPosition(wx.Point(100, 10))
    #wx.App.MainLoop(): Método para que se ejecute en un loop infinito el GUI, logrando que no se ejecute una
    #vez y luego cierre por sí solo, sino que solo se cierre al dar clic en el tache del frame.
    app.MainLoop()
```

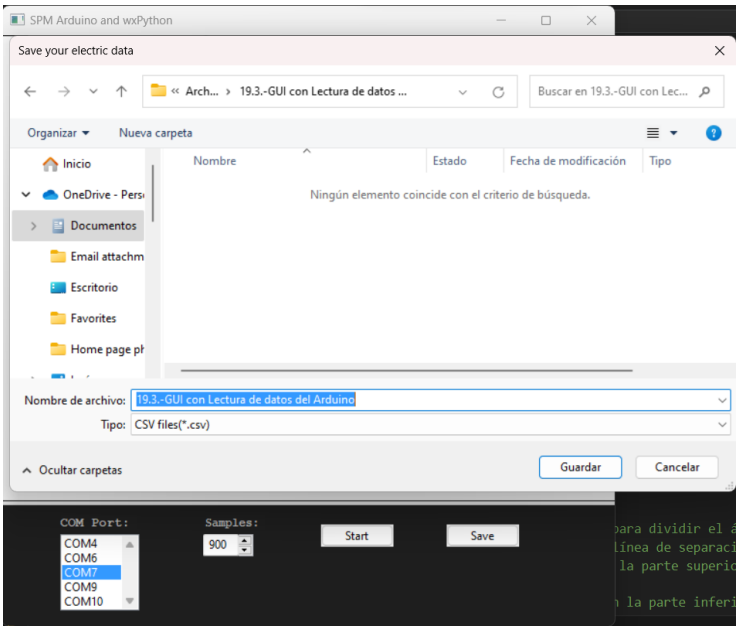
Resultado del Código Python

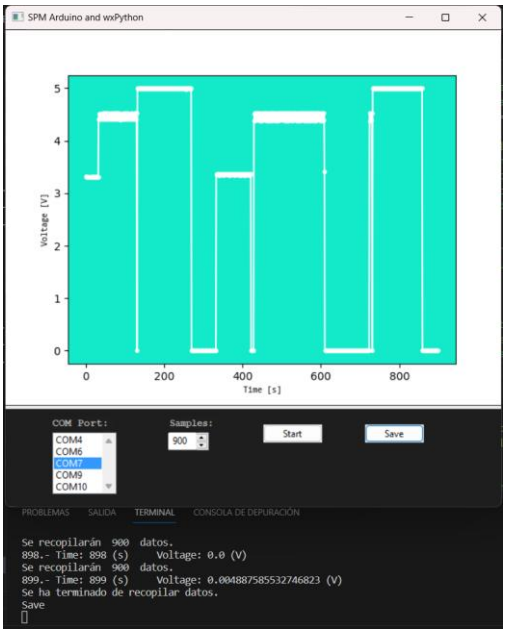


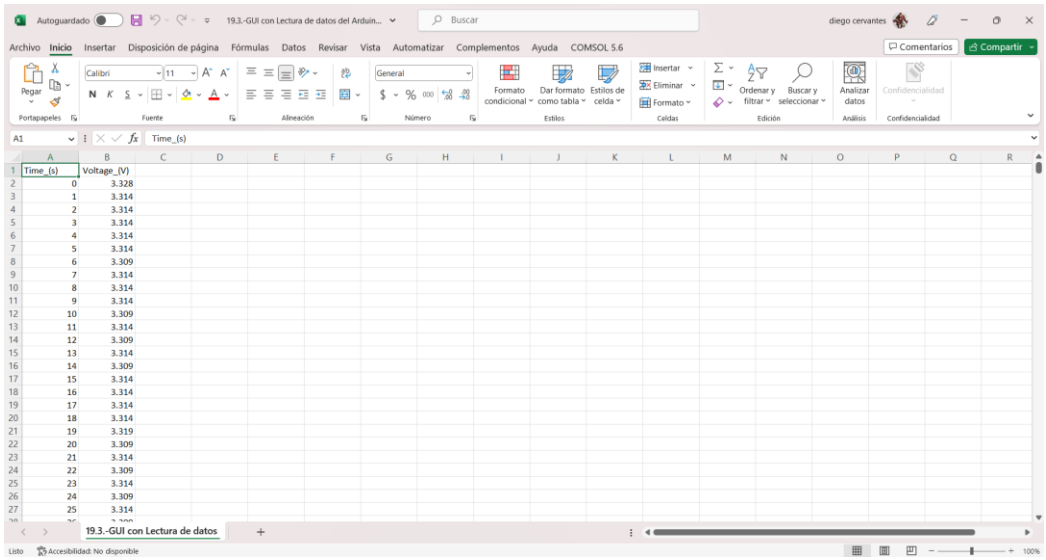


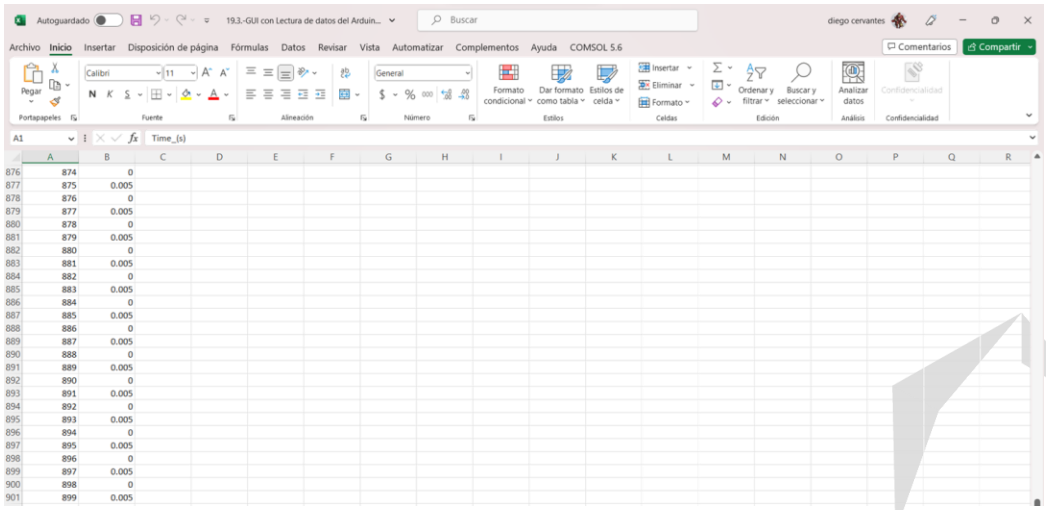

SPM Arduino and wxPython
Voltage [V]
Time [s]
COM Port:
COM4
COM6
COM7
COM9
COM10
Samples:
900
Stop
639.- Time: 639 (s) Voltage: 0.004887585532746823 (V)
Se recopilarán 900 datos.
640.- Time: 640 (s) Voltage: 0.0 (V)
Se recopilarán 900 datos.
641.- Time: 641 (s) Voltage: 0.004887585532746823 (V)
Se recopilarán 900 datos.

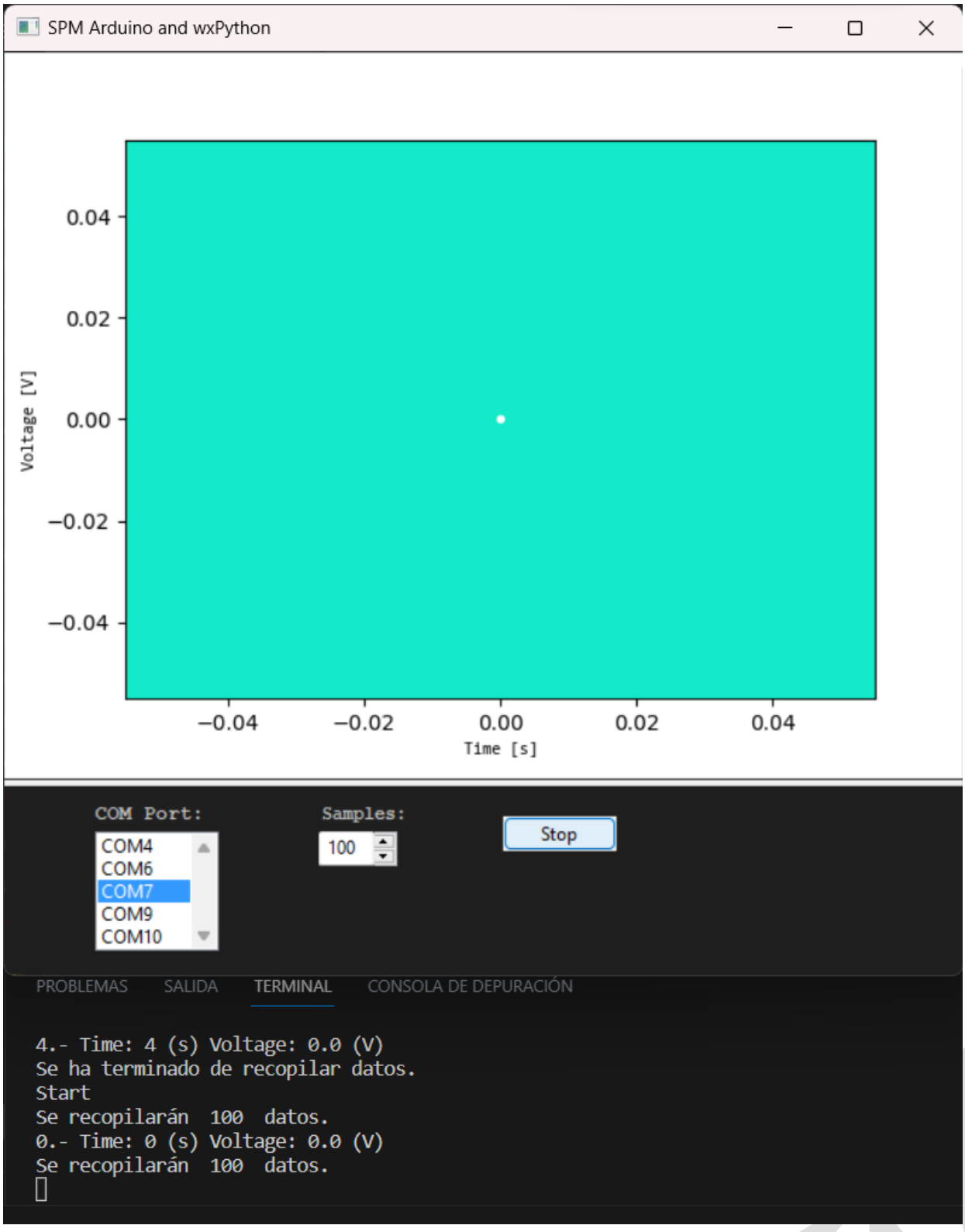

Instrumentación Virtual wxPython - Arduino
Voltage [V]
Time [s]
COM Port:
COM4
COM6
COM7
COM9
Samples:
900
Start
Save
Se recopilarán 900 datos.
896.- Time: 896.0 (s) Voltage: 3.313782991202346 (V)
Se recopilarán 900 datos.
897.- Time: 897.0 (s) Voltage: 3.2991202346041058 (V)
Se recopilarán 900 datos.
898.- Time: 898.0 (s) Voltage: 3.3088954056695994 (V)
Se recopilarán 900 datos.
899.- Time: 899.0 (s) Voltage: 3.3040078201368526 (V)
Se ha terminado de recopilar datos.
Reloj
Temporizador finalizado
15 min
11:55 a. m.
Ignorar
Python


SPM Arduino and wxPython
Save your electric data
Arch... › 19.3.-GUI con Lectura de datos ...
Buscar en 19.3.-GUI con Lec...
Organizar
Nueva carpeta
Inicio
OneDrive - Pers
Documentos
Email attachm
Escritorio
Favorites
Home page pl
Nombre
Estado
Fecha de modificación
Tipo
Ningún elemento coincide con el criterio de búsqueda.
Nombre de archivo: 19.3.-GUI con Lectura de datos del Arduino
Tipo: CSV files(*.csv)
Ocultar carpetas
Guardar
Cancelar
COM Port:
COM4
COM6
COM7
COM9
COM10
Samples:
900
Start
Save


SPM Arduino and wxPython
Voltage [V]
Time [s]
COM Port:
COM4
COM6
COM7
COM9
COM10
Samples:
900
Start
Save
PROBLEMAS
SALIDA
TERMINAL
CONSOLA DE DEPURACIÓN
Se recopilarán 900 datos.
898.- Time: 898 (s) Voltage: 0.0 (V)
Se recopilarán 900 datos.
899.- Time: 899 (s) Voltage: 0.004887585532746823 (V)
Se ha terminado de recopilar datos.
Save


19.3.-GUI con Lectura de datos del Arduin...
Archivo
Inicio
Insertar
Disposición de página
Fórmulas
Datos
Revisar
Vista
Automatizar
Complementos
Ayuda
COMSOL 5.6
Comentarios
Compartir
Time_(s)
Voltage_(V)
19.3.-GUI con Lectura de datos


19.3.-GUI con Lectura de datos del Arduin...
Archivo
Inicio
Insertar
Disposición de página
Fórmulas
Datos
Revisar
Vista
Automatizar
Complementos
Ayuda
COMSOL 5.6
Comentarios
Compartir
Time_(s)


SPM Arduino and wxPython
0.04
0.02
0.00
−0.02
−0.04
Voltage [V]
−0.04
−0.02
0.00
0.02
0.04
Time [s]
COM Port:
COM4
COM6
COM7
COM9
COM10
Samples:
100
Stop
PROBLEMAS
SALIDA
TERMINAL
CONSOLA DE DEPURACIÓN
4.- Time: 4 (s) Voltage: 0.0 (V)
Se ha terminado de recopilar datos.
Start
Se recopilarán 100 datos.
0.- Time: 0 (s) Voltage: 0.0 (V)
Se recopilarán 100 datos.