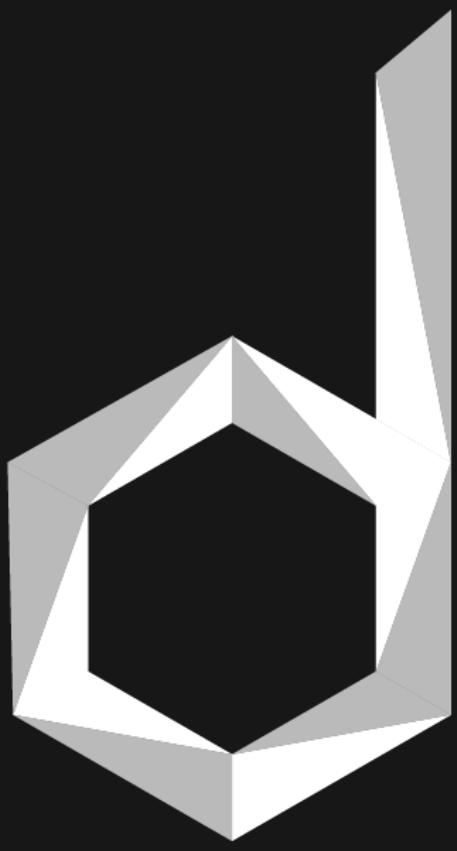
INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

INSTRUMENTACIÓN VIRTUAL

PYTHON 3.9.7, C# & LABVIEW

GUI de Instrumentación Virtual Bidireccional con wxPython y PyQt5

Contenido

Teoría – Instrumentación Virtual:

La instrumentación virtual es el uso de una computadora como instrumento de medición, en vez de utilizar herramientas físicas como osciloscopios, principalmente para bajar costos de operación en algún proceso. Esto se logra haciendo uso de instrumentos que no son tangibles para desarrollar aplicaciones con interfaz de usuario que realicen adquisición de datos a través de lenguajes de programación, protocolos de comunicación, etc.

Teoría – GUI (Graphical User Interface): wxPython y PyQt5

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

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

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

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

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

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

- **Método main:** Es un método en el lenguaje de programación Python a través del cual se ejecutan los métodos de todas las clases incluidas en el programa. **Se declara hasta el final del programa.**
- **Frame:** Es una clase perteneciente a la librería **wxPython** que sirve para crear la ventana de la GUI. **Se debe crear después de las clases que crean los contenedores de la interfaz gráfica.**
    - **SplitterWindow:** La mayoría de los widgets se colocan dentro de los Panel para que se puedan acomodar correctamente, pero existe este widget especial que sirve para cuando existen 2 contenedores principales en vez de uno solo y lo que hace es dividir el espacio de la ventana en dos partes ajustables. Con ajustable nos referimos a que

existirá una línea de separación entre las dos partes del **Frame** que permitirá hacerlas más grandes o chicas cuando se arrastre dicha línea.

- **Panel:** Es una clase perteneciente a la librería **wxPython** que sirve para crear los contenedores que se encuentran dentro del **Frame** de la GUI y que a su vez contienen los elementos gráficos con los que va a interactuar el usuario llamados **Widgets**. **Se debe crear de forma separada y antes de la clase que crea la ventana de la interfaz gráfica.**
    - **Widgets:** Son los botones, áreas de texto, controles de texto, checkbox, radio buttons, desplegables, listas, controles numéricos, imágenes, gráficas, etc. con los que interactúa el usuario en la GUI.

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

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

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

Instrucciones – Instrumentación Virtual Arduino:

Escriba dos programas que realicen una operación de instrumentación virtual con Arduino, una GUI hecha con wxPython y otra hecha con PyQt5, pudiendo recibir y mandar datos a la tarjeta de desarrollo.

En este caso lo que se busca es leer los datos de tensión de dos pines analógicos del Arduino (**A0** y **A1**) y escribir en dos pines digitales (**12** y **13**) para encender y apagar dichos leds dependiendo de si el valor recibido en los pines analógicos cruza el valor de un umbral dado por el programa o no, se describirá el comportamiento de los leds a continuación:

- Tensión analógica **A0**:
    - Si la tensión del pin **A0** es **mayor** **al umbral** **se enciende el led del pin 13**.

    - Si la tensión del pin **A0** es **menor al umbral no se enciende el led del pin 13**.
- Tensión analógica **A1**:
    - Si la tensión del pin **A1**es **mayor al umbral se enciende el led del pin 12**.
    - Si la tensión del pin **A1** es **menor al umbral no se enciende el led del pin 12**.

Estos datos se almacenarán en una variable de Python, se graficarán actualizando sus datos en tiempo real y finalmente serán guardados en un archivo de Excel. Para ello primero se debe haber ejecutado un programa en el IDE de Arduino que suba la librería Standard Firmata a la tarjeta de desarrollo, además de indicar cuál es el puerto de conexión serial que se debe utilizar.

Pseudocódigo.

1. Ejecutar un código en el IDE de Arduino que suba a la tarjeta de desarrollo la librería Standard Firmata de Arduino, para que de esa forma se pueda controlar y/o monitorear el estado de los pines digitales y analógicos del Arduino, luego se recaben 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. Iniciar la comunicación serial con el puerto de conexión previamente utilizado en el IDE del Arduino.
3. Indicar el número de datos a recabar del Arduino.
4. Indicar la tensión de umbral en unidad de mini Volts para encender los pines **12** y **13**.
5. Leer los datos de los pines analógicos **A0** y **A1** 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 podrá estar en el rango de 0 a $2^{10} - 1 = 1023$, algo importante a mencionar del método read() de la librería pyfirmata es que indica los datos binarios de tensión en un rango de valores de 0 a 1, por lo cual la conversión entonces se realiza a través de la siguiente ecuación:

$$Tensión = Tensión_{Binaria} * Tensión_{Max} = Tensión_{Binaria} * 5\ [V]$$

6. Detectar si los valores de tensión analógicos superan el valor de la tensión de umbral para ver si encienden su led correspondiente o no.
7. Almacenar los datos de tensión en una lista, tupla o diccionario.
8. Graficar los datos recabados de tiempo vs. tensión para los pines **A0**, **A1** y la tensión de umbral.
9. Actualizar dinámicamente la gráfica para que se actualicen sus valores y se muestren en tiempo real.
    a. Esto se realiza utilizando la librería **FigureCanvasWxAgg** de cuando se crea la GUI con la librería **wxPython**.
    b. Esto se realiza utilizando la librería **FigureCanvasQtAgg** de cuando se crea la GUI con la librería **PyQt5**.
10. Imprimir en consola el vector que haya almacenado todos los valores de tensión recabados.
11. Dar la opción de guardar los datos recabados en un archivo de Excel.

Código IDE Arduino – Librería Standard Firmata:

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.

```
/*La librería de Arduino que estás viendo es una biblioteca llamada "Firmata". Firmata es un protocolo
de comunicación que permite controlar un Arduino desde un software en una computadora o dispositivo
compatible. La biblioteca implementa el protocolo Firmata en Arduino, lo que permite establecer una
comunicación bidireccional entre el Arduino y el software de control, permitiendo controlar los pines
de Arduino, incluidos los pines analógicos y digitales, los pines PWM, los servos y la comunicación I2C.
En este caso ese software de control estará hecho con Python en Visual Studio Code.*/

#include <Servo.h>
#include <Wire.h>
#include <Firmata.h>

#define I2C_WRITE                   B00000000
#define I2C_READ                    B00001000
#define I2C_READ_CONTINUOUSLY       B00010000
#define I2C_STOP_READING            B00011000
#define I2C_READ_WRITE_MODE_MASK    B00011000
#define I2C_10BIT_ADDRESS_MODE_MASK B00100000
#define I2C_END_TX_MASK             B01000000
#define I2C_STOP_TX                 1
#define I2C_RESTART_TX              0
#define I2C_MAX_QUERIES             8
#define I2C_REGISTER_NOT_SPECIFIED  -1

// the minimum interval for sampling analog input
#define MINIMUM_SAMPLING_INTERVAL   1


/*==============================================================================
 * GLOBAL VARIABLES
 *============================================================================*/

#ifdef FIRMATA_SERIAL_FEATURE
SerialFirmata serialFeature;
#endif

/* analog inputs */
int analogInputsToReport = 0; // bitwise array to store pin reporting

/* digital input ports */
byte reportPINs[TOTAL_PORTS];       // 1 = report this port, 0 = silence
byte previousPINs[TOTAL_PORTS];     // previous 8 bits sent

/* pins configuration */
byte portConfigInputs[TOTAL_PORTS]; // each bit: 1 = pin in INPUT, 0 = anything else

/* timer variables */
unsigned long currentMillis;        // store the current value from millis()
unsigned long previousMillis;       // for comparison with currentMillis
unsigned int samplingInterval = 19; // how often to run the main loop (in ms)

/* i2c data */
struct i2c_device_info {
  byte addr;
  int reg;
  byte bytes;
  byte stopTX;
};

/* for i2c read continuous more */
i2c_device_info query[I2C_MAX_QUERIES];

byte i2cRxData[64];
boolean isI2CEnabled = false;
signed char queryIndex = -1;
// default delay time between i2c read request and Wire.requestFrom()
unsigned int i2cReadDelayTime = 0;

Servo servos[MAX_SERVOS];
byte servoPinMap[TOTAL_PINS];
byte detachedServos[MAX_SERVOS];
byte detachedServoCount = 0;
```

```
byte servoCount = 0;

boolean isResetting = false;

// Forward declare a few functions to avoid compiler errors with older versions
// of the Arduino IDE.
void setPinModeCallback(byte, int);
void reportAnalogCallback(byte analogPin, int value);
void sysexCallback(byte, byte, byte*);

/* utility functions */
void wireWrite(byte data)
{
#if ARDUINO >= 100
  Wire.write((byte)data);
#else
  Wire.send(data);
#endif
}

byte wireRead(void)
{
#if ARDUINO >= 100
  return Wire.read();
#else
  return Wire.receive();
#endif
}

/*==============================================================================
 * FUNCTIONS
 *============================================================================*/

void attachServo(byte pin, int minPulse, int maxPulse)
{
  if (servoCount < MAX_SERVOS) {
    // reuse indexes of detached servos until all have been reallocated
    if (detachedServoCount > 0) {
      servoPinMap[pin] = detachedServos[detachedServoCount - 1];
      if (detachedServoCount > 0) detachedServoCount--;
    } else {
      servoPinMap[pin] = servoCount;
      servoCount++;
    }
    if (minPulse > 0 && maxPulse > 0) {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin), minPulse, maxPulse);
    } else {
      servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin));
    }
  } else {
    Firmata.sendString("Max servos attached");
  }
}

void detachServo(byte pin)
{
  servos[servoPinMap[pin]].detach();
  // if we're detaching the last servo, decrement the count
  // otherwise store the index of the detached servo
  if (servoPinMap[pin] == servoCount && servoCount > 0) {
    servoCount--;
  } else if (servoCount > 0) {
    // keep track of detached servos because we want to reuse their indexes
    // before incrementing the count of attached servos
    detachedServoCount++;
    detachedServos[detachedServoCount - 1] = servoPinMap[pin];
  }

  servoPinMap[pin] = 255;
}

void enableI2CPins()
{
  byte i;
  // is there a faster way to do this? would probaby require importing
  // Arduino.h to get SCL and SDA pins
  for (i = 0; i < TOTAL_PINS; i++) {
    if (IS_PIN_I2C(i)) {
      // mark pins as i2c so they are ignore in non i2c data requests
      setPinModeCallback(i, PIN_MODE_I2C);
    }
```

```
  }

  isI2CEnabled = true;

  Wire.begin();
}

/* disable the i2c pins so they can be used for other functions */
void disableI2CPins() {
  isI2CEnabled = false;
  // disable read continuous mode for all devices
  queryIndex = -1;
}

void readAndReportData(byte address, int theRegister, byte numBytes, byte stopTX) {
  // allow I2C requests that don't require a register read
  // for example, some devices using an interrupt pin to signify new data available
  // do not always require the register read so upon interrupt you call Wire.requestFrom()
  if (theRegister != I2C_REGISTER_NOT_SPECIFIED) {
    Wire.beginTransmission(address);
    wireWrite((byte)theRegister);
    Wire.endTransmission(stopTX); // default = true
    // do not set a value of 0
    if (i2cReadDelayTime > 0) {
      // delay is necessary for some devices such as WiiNunchuck
      delayMicroseconds(i2cReadDelayTime);
    }
  } else {
    theRegister = 0;  // fill the register with a dummy value
  }

  Wire.requestFrom(address, numBytes);  // all bytes are returned in requestFrom

  // check to be sure correct number of bytes were returned by slave
  if (numBytes < Wire.available()) {
    Firmata.sendString("I2C: Too many bytes received");
  } else if (numBytes > Wire.available()) {
    Firmata.sendString("I2C: Too few bytes received");
  }

  i2cRxData[0] = address;
  i2cRxData[1] = theRegister;

  for (int i = 0; i < numBytes && Wire.available(); i++) {
    i2cRxData[2 + i] = wireRead();
  }

  // send slave address, register and received bytes
  Firmata.sendSysex(SYSEX_I2C_REPLY, numBytes + 2, i2cRxData);
}

void outputPort(byte portNumber, byte portValue, byte forceSend)
{
  // pins not configured as INPUT are cleared to zeros
  portValue = portValue & portConfigInputs[portNumber];
  // only send if the value is different than previously sent
  if (forceSend || previousPINs[portNumber] != portValue) {
    Firmata.sendDigitalPort(portNumber, portValue);
    previousPINs[portNumber] = portValue;
  }
}

/* -----------------------------------------------------------------------------
 * check all the active digital inputs for change of state, then add any events
 * to the Serial output queue using Serial.print() */
void checkDigitalInputs(void)
{
  /* Using non-looping code allows constants to be given to readPort().
   * The compiler will apply substantial optimizations if the inputs
   * to readPort() are compile-time constants. */
  if (TOTAL_PORTS > 0 && reportPINs[0]) outputPort(0, readPort(0, portConfigInputs[0]), false);
  if (TOTAL_PORTS > 1 && reportPINs[1]) outputPort(1, readPort(1, portConfigInputs[1]), false);
  if (TOTAL_PORTS > 2 && reportPINs[2]) outputPort(2, readPort(2, portConfigInputs[2]), false);
  if (TOTAL_PORTS > 3 && reportPINs[3]) outputPort(3, readPort(3, portConfigInputs[3]), false);
  if (TOTAL_PORTS > 4 && reportPINs[4]) outputPort(4, readPort(4, portConfigInputs[4]), false);
  if (TOTAL_PORTS > 5 && reportPINs[5]) outputPort(5, readPort(5, portConfigInputs[5]), false);
  if (TOTAL_PORTS > 6 && reportPINs[6]) outputPort(6, readPort(6, portConfigInputs[6]), false);
  if (TOTAL_PORTS > 7 && reportPINs[7]) outputPort(7, readPort(7, portConfigInputs[7]), false);
  if (TOTAL_PORTS > 8 && reportPINs[8]) outputPort(8, readPort(8, portConfigInputs[8]), false);
  if (TOTAL_PORTS > 9 && reportPINs[9]) outputPort(9, readPort(9, portConfigInputs[9]), false);
  if (TOTAL_PORTS > 10 && reportPINs[10]) outputPort(10, readPort(10, portConfigInputs[10]), false);
```

```
  if (TOTAL_PORTS > 11 && reportPINs[11]) outputPort(11, readPort(11, portConfigInputs[11]), false);
  if (TOTAL_PORTS > 12 && reportPINs[12]) outputPort(12, readPort(12, portConfigInputs[12]), false);
  if (TOTAL_PORTS > 13 && reportPINs[13]) outputPort(13, readPort(13, portConfigInputs[13]), false);
  if (TOTAL_PORTS > 14 && reportPINs[14]) outputPort(14, readPort(14, portConfigInputs[14]), false);
  if (TOTAL_PORTS > 15 && reportPINs[15]) outputPort(15, readPort(15, portConfigInputs[15]), false);
}

// -----------------------------------------------------------------------------
/* sets the pin mode to the correct state and sets the relevant bits in the
 * two bit-arrays that track Digital I/O and PWM status
 */
void setPinModeCallback(byte pin, int mode)
{
  if (Firmata.getPinMode(pin) == PIN_MODE_IGNORE)
    return;

  if (Firmata.getPinMode(pin) == PIN_MODE_I2C && isI2CEnabled && mode != PIN_MODE_I2C) {
    // disable i2c so pins can be used for other functions
    // the following if statements should reconfigure the pins properly
    disableI2CPins();
  }
  if (IS_PIN_DIGITAL(pin) && mode != PIN_MODE_SERVO) {
    if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
      detachServo(pin);
    }
  }
  if (IS_PIN_ANALOG(pin)) {
    reportAnalogCallback(PIN_TO_ANALOG(pin), mode == PIN_MODE_ANALOG ? 1 : 0); // turn on/off reporting
  }
  if (IS_PIN_DIGITAL(pin)) {
    if (mode == INPUT || mode == PIN_MODE_PULLUP) {
      portConfigInputs[pin / 8] |= (1 << (pin & 7));
    } else {
      portConfigInputs[pin / 8] &= ~(1 << (pin & 7));
    }
  }
  Firmata.setPinState(pin, 0);
  switch (mode) {
    case PIN_MODE_ANALOG:
      if (IS_PIN_ANALOG(pin)) {
        if (IS_PIN_DIGITAL(pin)) {
          pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
          // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
          digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        }
        Firmata.setPinMode(pin, PIN_MODE_ANALOG);
      }
      break;
    case INPUT:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
#if ARDUINO <= 100
        // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
        digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
#endif
        Firmata.setPinMode(pin, INPUT);
      }
      break;
    case PIN_MODE_PULLUP:
      if (IS_PIN_DIGITAL(pin)) {
        pinMode(PIN_TO_DIGITAL(pin), INPUT_PULLUP);
        Firmata.setPinMode(pin, PIN_MODE_PULLUP);
        Firmata.setPinState(pin, 1);
      }
      break;
    case OUTPUT:
      if (IS_PIN_DIGITAL(pin)) {
        if (Firmata.getPinMode(pin) == PIN_MODE_PWM) {
          // Disable PWM if pin mode was previously set to PWM.
          digitalWrite(PIN_TO_DIGITAL(pin), LOW);
        }
        pinMode(PIN_TO_DIGITAL(pin), OUTPUT);
        Firmata.setPinMode(pin, OUTPUT);
      }
      break;
    case PIN_MODE_PWM:
      if (IS_PIN_PWM(pin)) {
        pinMode(PIN_TO_PWM(pin), OUTPUT);
        analogWrite(PIN_TO_PWM(pin), 0);
```

```
        Firmata.setPinMode(pin, PIN_MODE_PWM);
      }
      break;
    case PIN_MODE_SERVO:
      if (IS_PIN_DIGITAL(pin)) {
        Firmata.setPinMode(pin, PIN_MODE_SERVO);
        if (servoPinMap[pin] == 255 || !servos[servoPinMap[pin]].attached()) {
          // pass -1 for min and max pulse values to use default values set
          // by Servo library
          attachServo(pin, -1, -1);
        }
      }
      break;
    case PIN_MODE_I2C:
      if (IS_PIN_I2C(pin)) {
        // mark the pin as i2c
        // the user must call I2C_CONFIG to enable I2C for a device
        Firmata.setPinMode(pin, PIN_MODE_I2C);
      }
      break;
    case PIN_MODE_SERIAL:
#ifdef FIRMATA_SERIAL_FEATURE
      serialFeature.handlePinMode(pin, PIN_MODE_SERIAL);
#endif
      break;
    default:
      Firmata.sendString("Unknown pin mode"); // TODO: put error msgs in EEPROM
  }
  // TODO: save status to EEPROM here, if changed
}

/*
 * Sets the value of an individual pin. Useful if you want to set a pin value but
 * are not tracking the digital port state.
 * Can only be used on pins configured as OUTPUT.
 * Cannot be used to enable pull-ups on Digital INPUT pins.
 */
void setPinValueCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS && IS_PIN_DIGITAL(pin)) {
    if (Firmata.getPinMode(pin) == OUTPUT) {
      Firmata.setPinState(pin, value);
      digitalWrite(PIN_TO_DIGITAL(pin), value);
    }
  }
}

void analogWriteCallback(byte pin, int value)
{
  if (pin < TOTAL_PINS) {
    switch (Firmata.getPinMode(pin)) {
      case PIN_MODE_SERVO:
        if (IS_PIN_DIGITAL(pin))
          servos[servoPinMap[pin]].write(value);
        Firmata.setPinState(pin, value);
        break;
      case PIN_MODE_PWM:
        if (IS_PIN_PWM(pin))
          analogWrite(PIN_TO_PWM(pin), value);
        Firmata.setPinState(pin, value);
        break;
    }
  }
}

void digitalWriteCallback(byte port, int value)
{
  byte pin, lastPin, pinValue, mask = 1, pinWriteMask = 0;

  if (port < TOTAL_PORTS) {
    // create a mask of the pins on this port that are writable.
    lastPin = port * 8 + 8;
    if (lastPin > TOTAL_PINS) lastPin = TOTAL_PINS;
    for (pin = port * 8; pin < lastPin; pin++) {
      // do not disturb non-digital pins (eg, Rx & Tx)
      if (IS_PIN_DIGITAL(pin)) {
        // do not touch pins in PWM, ANALOG, SERVO or other modes
        if (Firmata.getPinMode(pin) == OUTPUT || Firmata.getPinMode(pin) == INPUT) {
          pinValue = ((byte)value & mask) ? 1 : 0;
          if (Firmata.getPinMode(pin) == OUTPUT) {
            pinWriteMask |= mask;
```

```
          } else if (Firmata.getPinMode(pin) == INPUT && pinValue == 1 && Firmata.getPinState(pin) != 1) {
            // only handle INPUT here for backwards compatibility
#if ARDUINO > 100
            pinMode(pin, INPUT_PULLUP);
#else
            // only write to the INPUT pin to enable pullups if Arduino v1.0.0 or earlier
            pinWriteMask |= mask;
#endif
          }
          Firmata.setPinState(pin, pinValue);
        }
      }
      mask = mask << 1;
    }
    writePort(port, (byte)value, pinWriteMask);
  }
}


// -----------------------------------------------------------------------------
/* sets bits in a bit array (int) to toggle the reporting of the analogIns
 */
//void FirmataClass::setAnalogPinReporting(byte pin, byte state) {
//}
void reportAnalogCallback(byte analogPin, int value)
{
  if (analogPin < TOTAL_ANALOG_PINS) {
    if (value == 0) {
      analogInputsToReport = analogInputsToReport & ~ (1 << analogPin);
    } else {
      analogInputsToReport = analogInputsToReport | (1 << analogPin);
      // prevent during system reset or all analog pin values will be reported
      // which may report noise for unconnected analog pins
      if (!isResetting) {
        // Send pin value immediately. This is helpful when connected via
        // ethernet, wi-fi or bluetooth so pin states can be known upon
        // reconnecting.
        Firmata.sendAnalog(analogPin, analogRead(analogPin));
      }
    }
  }
  // TODO: save status to EEPROM here, if changed
}

void reportDigitalCallback(byte port, int value)
{
  if (port < TOTAL_PORTS) {
    reportPINs[port] = (byte)value;
    // Send port value immediately. This is helpful when connected via
    // ethernet, wi-fi or bluetooth so pin states can be known upon
    // reconnecting.
    if (value) outputPort(port, readPort(port, portConfigInputs[port]), true);
  }
  // do not disable analog reporting on these 8 pins, to allow some
  // pins used for digital, others analog.  Instead, allow both types
  // of reporting to be enabled, but check if the pin is configured
  // as analog when sampling the analog inputs.  Likewise, while
  // scanning digital pins, portConfigInputs will mask off values from any
  // pins configured as analog
}

/*==============================================================================
 * SYSEX-BASED commands
 *============================================================================*/

void sysexCallback(byte command, byte argc, byte *argv)
{
  byte mode;
  byte stopTX;
  byte slaveAddress;
  byte data;
  int slaveRegister;
  unsigned int delayTime;

  switch (command) {
    case I2C_REQUEST:
      mode = argv[1] & I2C_READ_WRITE_MODE_MASK;
      if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {
        Firmata.sendString("10-bit addressing not supported");
        return;
      }
```

```
else {
  slaveAddress = argv[0];
}

// need to invert the logic here since 0 will be default for client
// libraries that have not updated to add support for restart tx
if (argv[1] & I2C_END_TX_MASK) {
  stopTX = I2C_RESTART_TX;
}
else {
  stopTX = I2C_STOP_TX; // default
}

switch (mode) {
  case I2C_WRITE:
    Wire.beginTransmission(slaveAddress);
    for (byte i = 2; i < argc; i += 2) {
      data = argv[i] + (argv[i + 1] << 7);
      wireWrite(data);
    }
    Wire.endTransmission();
    delayMicroseconds(70);
    break;
  case I2C_READ:
    if (argc == 6) {
      // a slave register is specified
      slaveRegister = argv[2] + (argv[3] << 7);
      data = argv[4] + (argv[5] << 7);  // bytes to read
    }
    else {
      // a slave register is NOT specified
      slaveRegister = I2C_REGISTER_NOT_SPECIFIED;
      data = argv[2] + (argv[3] << 7);  // bytes to read
    }
    readAndReportData(slaveAddress, (int)slaveRegister, data, stopTX);
    break;
  case I2C_READ_CONTINUOUSLY:
    if ((queryIndex + 1) >= I2C_MAX_QUERIES) {
      // too many queries, just ignore
      Firmata.sendString("too many queries");
      break;
    }
    if (argc == 6) {
      // a slave register is specified
      slaveRegister = argv[2] + (argv[3] << 7);
      data = argv[4] + (argv[5] << 7);  // bytes to read
    }
    else {
      // a slave register is NOT specified
      slaveRegister = (int)I2C_REGISTER_NOT_SPECIFIED;
      data = argv[2] + (argv[3] << 7);  // bytes to read
    }
    queryIndex++;
    query[queryIndex].addr = slaveAddress;
    query[queryIndex].reg = slaveRegister;
    query[queryIndex].bytes = data;
    query[queryIndex].stopTX = stopTX;
    break;
  case I2C_STOP_READING:
    byte queryIndexToSkip;
    // if read continuous mode is enabled for only 1 i2c device, disable
    // read continuous reporting for that device
    if (queryIndex <= 0) {
      queryIndex = -1;
    } else {
      queryIndexToSkip = 0;
      // if read continuous mode is enabled for multiple devices,
      // determine which device to stop reading and remove it's data from
      // the array, shifiting other array data to fill the space
      for (byte i = 0; i < queryIndex + 1; i++) {
        if (query[i].addr == slaveAddress) {
          queryIndexToSkip = i;
          break;
        }
      }

      for (byte i = queryIndexToSkip; i < queryIndex + 1; i++) {
        if (i < I2C_MAX_QUERIES) {
          query[i].addr = query[i + 1].addr;
          query[i].reg = query[i + 1].reg;
          query[i].bytes = query[i + 1].bytes;
```

```
            query[i].stopTX = query[i + 1].stopTX;
          }
        }
        queryIndex--;
      }
      break;
    default:
      break;
  }
  break;
case I2C_CONFIG:
  delayTime = (argv[0] + (argv[1] << 7));

  if (argc > 1 && delayTime > 0) {
    i2cReadDelayTime = delayTime;
  }

  if (!isI2CEnabled) {
    enableI2CPins();
  }

  break;
case SERVO_CONFIG:
  if (argc > 4) {
    // these vars are here for clarity, they'll optimized away by the compiler
    byte pin = argv[0];
    int minPulse = argv[1] + (argv[2] << 7);
    int maxPulse = argv[3] + (argv[4] << 7);

    if (IS_PIN_DIGITAL(pin)) {
      if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
        detachServo(pin);
      }
      attachServo(pin, minPulse, maxPulse);
      setPinModeCallback(pin, PIN_MODE_SERVO);
    }
  }
  break;
case SAMPLING_INTERVAL:
  if (argc > 1) {
    samplingInterval = argv[0] + (argv[1] << 7);
    if (samplingInterval < MINIMUM_SAMPLING_INTERVAL) {
      samplingInterval = MINIMUM_SAMPLING_INTERVAL;
    }
  } else {
    //Firmata.sendString("Not enough data");
  }
  break;
case EXTENDED_ANALOG:
  if (argc > 1) {
    int val = argv[1];
    if (argc > 2) val |= (argv[2] << 7);
    if (argc > 3) val |= (argv[3] << 14);
    analogWriteCallback(argv[0], val);
  }
  break;
case CAPABILITY_QUERY:
  Firmata.write(START_SYSEX);
  Firmata.write(CAPABILITY_RESPONSE);
  for (byte pin = 0; pin < TOTAL_PINS; pin++) {
    if (IS_PIN_DIGITAL(pin)) {
      Firmata.write((byte)INPUT);
      Firmata.write(1);
      Firmata.write((byte)PIN_MODE_PULLUP);
      Firmata.write(1);
      Firmata.write((byte)OUTPUT);
      Firmata.write(1);
    }
    if (IS_PIN_ANALOG(pin)) {
      Firmata.write(PIN_MODE_ANALOG);
      Firmata.write(10); // 10 = 10-bit resolution
    }
    if (IS_PIN_PWM(pin)) {
      Firmata.write(PIN_MODE_PWM);
      Firmata.write(DEFAULT_PWM_RESOLUTION);
    }
    if (IS_PIN_DIGITAL(pin)) {
      Firmata.write(PIN_MODE_SERVO);
      Firmata.write(14);
    }
    if (IS_PIN_I2C(pin)) {
```

```
          Firmata.write(PIN_MODE_I2C);
          Firmata.write(1);  // TODO: could assign a number to map to SCL or SDA
        }
#ifdef FIRMATA_SERIAL_FEATURE
        serialFeature.handleCapability(pin);
#endif
        Firmata.write(127);
      }
      Firmata.write(END_SYSEX);
      break;
    case PIN_STATE_QUERY:
      if (argc > 0) {
        byte pin = argv[0];
        Firmata.write(START_SYSEX);
        Firmata.write(PIN_STATE_RESPONSE);
        Firmata.write(pin);
        if (pin < TOTAL_PINS) {
          Firmata.write(Firmata.getPinMode(pin));
          Firmata.write((byte)Firmata.getPinState(pin) & 0x7F);
          if (Firmata.getPinState(pin) & 0xFF80) Firmata.write((byte)(Firmata.getPinState(pin) >> 7) & 0x7F);
          if (Firmata.getPinState(pin) & 0xC000) Firmata.write((byte)(Firmata.getPinState(pin) >> 14) & 0x7F);
        }
        Firmata.write(END_SYSEX);
      }
      break;
    case ANALOG_MAPPING_QUERY:
      Firmata.write(START_SYSEX);
      Firmata.write(ANALOG_MAPPING_RESPONSE);
      for (byte pin = 0; pin < TOTAL_PINS; pin++) {
        Firmata.write(IS_PIN_ANALOG(pin) ? PIN_TO_ANALOG(pin) : 127);
      }
      Firmata.write(END_SYSEX);
      break;

    case SERIAL_MESSAGE:
#ifdef FIRMATA_SERIAL_FEATURE
      serialFeature.handleSysex(command, argc, argv);
#endif
      break;
  }
}

/*==============================================================================
 * SETUP()
 *============================================================================*/

void systemResetCallback()
{
  isResetting = true;

  // initialize a defalt state
  // TODO: option to load config from EEPROM instead of default

#ifdef FIRMATA_SERIAL_FEATURE
  serialFeature.reset();
#endif

  if (isI2CEnabled) {
    disableI2CPins();
  }

  for (byte i = 0; i < TOTAL_PORTS; i++) {
    reportPINs[i] = false;    // by default, reporting off
    portConfigInputs[i] = 0;  // until activated
    previousPINs[i] = 0;
  }

  for (byte i = 0; i < TOTAL_PINS; i++) {
    // pins with analog capability default to analog input
    // otherwise, pins default to digital output
    if (IS_PIN_ANALOG(i)) {
      // turns off pullup, configures everything
      setPinModeCallback(i, PIN_MODE_ANALOG);
    } else if (IS_PIN_DIGITAL(i)) {
      // sets the output to 0, configures portConfigInputs
      setPinModeCallback(i, OUTPUT);
    }

    servoPinMap[i] = 255;
  }
  // by default, do not report any analog inputs
```

```
  analogInputsToReport = 0;

  detachedServoCount = 0;
  servoCount = 0;

  /* send digital inputs to set the initial state on the host computer,
   * since once in the loop(), this firmware will only send on change */
  /*
  TODO: this can never execute, since no pins default to digital input
        but it will be needed when/if we support EEPROM stored config
  for (byte i=0; i < TOTAL_PORTS; i++) {
    outputPort(i, readPort(i, portConfigInputs[i]), true);
  }
  */
  isResetting = false;
}

void setup()
{
  Firmata.setFirmwareVersion(FIRMATA_FIRMWARE_MAJOR_VERSION, FIRMATA_FIRMWARE_MINOR_VERSION);

  Firmata.attach(ANALOG_MESSAGE, analogWriteCallback);
  Firmata.attach(DIGITAL_MESSAGE, digitalWriteCallback);
  Firmata.attach(REPORT_ANALOG, reportAnalogCallback);
  Firmata.attach(REPORT_DIGITAL, reportDigitalCallback);
  Firmata.attach(SET_PIN_MODE, setPinModeCallback);
  Firmata.attach(SET_DIGITAL_PIN_VALUE, setPinValueCallback);
  Firmata.attach(START_SYSEX, sysexCallback);
  Firmata.attach(SYSTEM_RESET, systemResetCallback);

  // to use a port other than Serial, such as Serial1 on an Arduino Leonardo or Mega,
  // Call begin(baud) on the alternate serial port and pass it to Firmata to begin like this:
  // Serial1.begin(57600);
  // Firmata.begin(Serial1);
  // However do not do this if you are using SERIAL_MESSAGE

  Firmata.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for ATmega32u4-based boards and Arduino 101
  }

  systemResetCallback();  // reset to default config
}

/*==============================================================================
 * LOOP()
 *============================================================================*/
void loop()
{
  byte pin, analogPin;

  /* DIGITALREAD - as fast as possible, check for changes and output them to the
   * FTDI buffer using Serial.print()  */
  checkDigitalInputs();

  /* STREAMREAD - processing incoming messagse as soon as possible, while still
   * checking digital inputs.  */
  while (Firmata.available())
    Firmata.processInput();

  // TODO - ensure that Stream buffer doesn't go over 60 bytes

  currentMillis = millis();
  if (currentMillis - previousMillis > samplingInterval) {
    previousMillis += samplingInterval;
    /* ANALOGREAD - do all analogReads() at the configured sampling interval */
    for (pin = 0; pin < TOTAL_PINS; pin++) {
      if (IS_PIN_ANALOG(pin) && Firmata.getPinMode(pin) == PIN_MODE_ANALOG) {
        analogPin = PIN_TO_ANALOG(pin);
        if (analogInputsToReport & (1 << analogPin)) {
          Firmata.sendAnalog(analogPin, analogRead(analogPin));
        }
      }
    }
    // report i2c data for all device with read continuous mode enabled
    if (queryIndex > -1) {
      for (byte i = 0; i < queryIndex + 1; i++) {
        readAndReportData(query[i].addr, query[i].reg, query[i].bytes, query[i].stopTX);
      }
    }
  }
```

```
#ifdef FIRMATA_SERIAL_FEATURE
  serialFeature.update();
#endif
}
```

GUI **wxPython** – Visual Studio Code (Logo Azul):

```
# -*- 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 #numpy: Librería que realiza operaciones matemáticas complejas (matriciales).
import time #time: Librería del manejo de tiempos, como retardos, contadores, etc.
import serial #serial: Librería que establece una comunicación serial con  microcontroladores, módems, etc.
import sys #sys: Librería que permite interactuar directamente con el sistema operativo y consola del ordenador.
import glob #glob: Librería que sirve para buscar archivos o directorios.
#pyfirmata: Librería que permite la comunicación bidireccional entre Python y Arduino, brindando control y
#monitoreo de sus pines digitales y analógicos, envío de señales PWM, lectura y escritura de datos y establecer
#una comunicación a través de los protocolos I2C y Serial. La comunicación entre Arduino y Python se realiza
#utilizando el protocolo Firmata, el cual permite controlar y monitorear dispositivos conectados a una placa
#Arduino desde un software de computadora, para habilitarlo, primero se debe subir el programa:
#21.-Stardard_Firmata_Mandar_Datos_Arduino.ino a la placa de desarrollo a través del IDE de Arduino.
import pyfirmata

#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 hereda 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(Llamada al constructor heredado).__init__(Parámetros que se le asignan): Lo que hace el método
        #super() es llamar al constructor de la clase padre de la clase actual (si es que no se le indica ningún
        #parámetro) o a cualquier clase que se le indique en su parámetro, después la instrucción .__init__()
        #asigna valores default a los parámetros del constructor de la clase padre (si es que no se indica
        #ningún parámetro), aunque de igual manera se pueden asignar parámetros adicionales, cualquier parámetro
        #incluido en el método init, será considerado como adicional. En conclusión, lo que está realizando la
        #línea de código es primero llamar al constructor de la superclase para realizar las tareas de
        #inicialización requeridas antes de indicar parámetros adicionales a la instancia de la clase actual.
        super().__init__(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.set_facecolor(): Método para indicar el color de fondo de la parte exterior de
        #la gráfica, el cual puede ser indicado por los mismos colores que se aplican al 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
        #Es importante mencionar que el método set_facecolor() se puede aplicar a varios elementos de la
        #gráfica, en este caso se está aplicando directamente al objeto figure, por eso es que se aplica el
        #color a la parte que está fuera de la gráfica.
        self.figure.set_facecolor('xkcd:silver')
        #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.Figure().add_subplot().tick_params(): Método para acceder a las propiedades gráficas de los
#números que aparecen en los ejes de la gráfica creada con la librería matplotlib.
# - axis: Indica el eje que se quiere afectar, ya sea 'x', 'y', 'both' o 'all'.
# - colors: Indica el color de los números colocados en el eje x o y, para ello es válido usar colores:
#       - Básicos CSS: como "red", "blue", "green", etc.
#       - Colores hexadecimales: "#FF0000", "#00FF00", etc.
#       - Colores RGB: "rgb(255,255,255)" para el color blanco y "rgb(0,0,0)" para el negro.
# - labelsize: Permite especificar el tamaño de letra en pixeles.
# - pad: espacio entre las marcas y las etiquetas en puntos.
self.axes.tick_params(axis = "x", colors = "white", labelsize = 8)
self.axes.tick_params(axis = "y", colors = "white", labelsize = 8)
#matplotlib.Figure.add_subplot().set_facecolor(): Método para indicar el color de fondo de la gráfica,
#el cual puede ser indicado por los mismos colores previamente mencionados en el método plot() o se
#pueden usar los siguientes con el código xkcd:
# - Colores: xkcd:aqua, xkcd:aquamarine, xkcd:azure, xkcd:beige, etc. Los colores se obtienen de:
#https://matplotlib.org/stable/tutorials/colors/colors.html
#Es importante mencionar que el método set_facecolor() se puede aplicar a varios elementos de la
#gráfica, en este caso se está aplicando al objeto axes, obtenido de aplicar el método add_subplot(),
#por eso es que se aplica el color a la parte que interna de la gráfica.
self.axes.set_facecolor('xkcd:aqua')
#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, y0, y1, u):
        #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
            #horizontal de la gráfica, recibe los siguientes parámetros:
            # - xlabel: Especifica el texto que se mostrará en el eje x.
            # - fontname: Indica el estilo de la fuente:
            #       - Nombres de tipos de letra estándar: "Arial", "Times New Roman", "Helvetica", "Courier",
            #         "Monospaced", "Consolas", etc. Estos nombres deben ser compatibles con los tipos de letra
            #         instalados en el sistema operativo.
            #       - Nombres de tipos de letra genéricos: "serif", "sans-serif", "monospace", etc.
            #       - Rutas de archivo: Si se tiene un archivo de tipo de letra personalizado, se puede
            #         especificar la ruta del archivo como el valor de fontname.
            # - fontsize: Indica el tamaño de la fuente.
            # - labelpad: Especifica el espaciado entre la etiqueta del eje x y el eje en sí
            # - color: Indica el color del texto colocado en el eje x, para ello es válido usar colores:
            #       - Básicos CSS: como "red", "blue", "green", etc.
            #       - Colores hexadecimales: "#FF0000", "#00FF00", etc.
            #       - Colores RGB: "rgb(255,255,255)" para el color blanco y "rgb(0,0,0)" para el negro.
        self.axes.set_xlabel(xlabel = "Time [s]", fontname = "Consolas", fontsize = 8, color = "white")
        #matplotlib.Figure.add_subplot().set_ylabel(): Método para indicar el texto que aparece en el eje y.
        self.axes.set_ylabel(ylabel = "Voltage [V]", fontname = "Consolas", fontsize = 8, color = "white")
        #matplotlib.Figure.add_subplot().plot(): Método usado para gráficar, indicando como primer parámetro su
        #eje horizontal, luego su eje vertical y finalmente el estilo de la gráfica:
        # - Colores:             C1: color naranja, r: color rojo, b: color azul, g: verde, c: cyan, m: morado,
        #   y: amarillo, k: negro, w: blanco.
        # - Tipo de marcadores:  o: círculos, +: símbolos de más, .; puntos, v: Triángulo hacia abajo, h:
        #   Hexágono, etc.
        # - Tipo de Líneas:      -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto,
        #   'or': Nada.
        self.axes.plot(x, y0, 'y1:') #'y:' c: color amarillo, 1: tri_down, :: línea punteada.
        self.axes.plot(x, y1, 'r2--') #'r.--' c: color rojo, 2: tri_up, --: linea punteada.
```

```
        self.axes.plot(x, u, 'w,-') #'w,-' c: color cyan, .: pixel, -: línea solida.
        #matplotlib.Figure().add_subplot().legend(): La leyenda de un gráfico es un componente que proporciona
        #información individualmente sobre los diferentes elementos o series presentes en el gráfico, el método
        #permite personalizar dicha leyenda a través de los siguientes parámetros:
        # - labels: Se refiere a los nombres que se les puede poner para las leyendas de los distintos elementos
        #   del gráfico.
        #           - Las etiquetas se generan automáticamente pero se pueden colocar manualmente de la
        #             siguiente manera:
        #               - labels=['Etiqueta 1', 'Etiqueta 2']
        # - loc: Indica la ubicación de la leyenda en el gráfico. Se puede indicar con una cadena de texto o un
        #   código numérico que represente una posición específica.
        #           - "best": Coloca la leyenda en la mejor ubicación posible, evitando superponerse con otros
        #             elementos.
        #           - "upper right": Coloca la leyenda en la esquina superior derecha del gráfico.
        #           - "upper left": Coloca la leyenda en la esquina superior izquierda del gráfico.
        #           - "lower right": Coloca la leyenda en la esquina inferior derecha del gráfico.
        #           - "lower left": Coloca la leyenda en la esquina inferior izquierda del gráfico.
        #           - "right": Coloca la leyenda en el lado derecho del gráfico, centrada verticalmente.
        #           - "center left": Coloca la leyenda en el lado izquierdo del gráfico, centrada verticalmente.
        #           - "center right": Coloca la leyenda en el lado derecho del gráfico, centrada verticalmente.
        #           - "center": Coloca la leyenda en el centro del gráfico.
        self.axes.legend(labels=['Pin A0', 'Pin A1', 'Umbral Leds 13/12'], loc = "best")
        #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 hereda 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(Llamada al constructor heredado).__init__(Parámetros que se le asignan): Lo que hace el método
        #super() es llamar al constructor de la clase padre de la clase actual (si es que no se le indica ningún
        #parámetro) o a cualquier clase que se le indique en su parámetro, después la instrucción .__init__()
        #asigna valores default a los parámetros del constructor de la clase padre (si es que no se indica
        #ningún parámetro), aunque de igual manera se pueden asignar parámetros adicionales, cualquier parámetro
        #incluido en el método init, será considerado como adicional. En conclusión, lo que está realizando la
```

```
        #línea de código es primero llamar al constructor de la superclase para realizar las tareas de
        #inicialización requeridas antes de indicar parámetros adicionales a la instancia de la clase actual.
        super().__init__(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 = (400, 40))
        self.buttonStop = wx.Button(parent = self, label = "Stop", pos = (400, 40))
        self.buttonSaveData = wx.Button(parent = self, label = "Save", pos = (500, 40), 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 = "Puerto COM:",   #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 = "Número de Muestras:",                  #Número de Muestras.
                        pos = (190, 20))
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)

self.LabelUmbral = wx.StaticText(self,
                        label = "Umbral Encendido LED [mV]:",           #Umbral de encendido de leds.
                        pos = (190, 70))
self.LabelUmbral.SetForegroundColour(wx.LIGHT_GREY)     #Letra gris claro.
font3 = self.LabelUmbral.GetFont()
font3.SetWeight(wx.FONTWEIGHT_BOLD)                     #Letra en negritas.
font3.SetFamily(wx.FONTFAMILY_TELETYPE)                 #Letra tipo máquina de escribir.
font1.SetPointSize(10)                                  #Tamaño de letra = 10.
self.LabelUmbral.SetFont(font3)

#CREACIÓN DE LOS WIDGETS: List Box, lista de elementos
#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 = (40, 40), 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, control numérico
#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: Indica el valor mínimo permitido en el control numérico SpinCtrl.
# - max: Indica el valor máximo permitido en el control numérico 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 = (190, 40), name = "wxSpinCtrlTime",
                                        style = wx.SP_WRAP,
                                        min = 1,
                                        max = 32000,
                                        initial = 1)
        self.spinCtrlUmbral = wx.SpinCtrl(parent = self, value = "", pos = (190, 90), name = "wxSpinCtrlUmbral",
                                        style = wx.SP_WRAP,
                                        min = 0,
                                        max = 5000,
                                        initial = 0)

        #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.y0 = np.array([])          #Numpy array del eje vertical perteneciente al pin A0 (y1 = tensión [V])
        self.y1 = np.array([])          #Numpy array del eje vertical perteneciente al pin A1 (y2 = tensión [V])
        self.u = np.array([])           #Numpy array del eje vertical perteneciente al umbral (u = tensión [mV])
        self.values = []                #Lista que guarda los valores de tiempo [s] y tensión [V] recopilados

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

        self.highValueBoard = 5.0       #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.micro_board = None

        self.umbral=0


    #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(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
    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.micro_board != None):
        #pyserial.Arduino.exit(): Método que cierra la comunicación serial. Es muy importante mencionar
        #que si no se ejecuta este método, el puerto serial se va a quedar bloqueado y no se podrá usar.
        self.micro_board.exit()
    #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 y A1 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.y0 = np.array([])          #Reinicio del eje vertical perteneciente al pin A0 (y1 = tensión [V]).
    self.y1 = np.array([])          #Reinicio del eje vertical perteneciente al pin A1 (y2 = tensión [V]).
    self.u = np.array([])           #Reinicio del eje vertical perteneciente al umbral (u = tensión [mV]).
    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):
            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:
            #PYFIRMATA: CONEXIÓN SERIAL, CONTROL Y MONITOREO DE PINES DE UNA TARJETA DE DESARROLLO
            #Instancia de la clase Arduino, que pertenece a la librería pyfirmata, dicho objeto proporciona
            #métodos para comunicarse con placas Arduino utilizando el protocolo Firmata, permitiendo así el
            #control y monitoreo de sus pines digitales y analógicos, realizar el envío de señales PWM,
            #lectura y/o escritura de datos y establecer una comunicación a través de los protocolos I2C y
            #Serial, la conexión serial con los microcontroladores se realiza utilizando la clase
            #pyfirmata.Arduino(), independientemente del tipo de microcontrolador que se esté utilizando, ya
            #que la librería pyfirmata proporciona una interfaz común para comunicarse con diferentes placas
            #de desarrollo, incluyendo Arduino, Raspberry Pi, Intel Galileo, PyCARD, etc. Para ello su
            #constructor recibe los siguientes parámetros:
            # - port (obligatorio): Especifica el puerto de comunicación a través del cual se conectará la
            #   placa Arduino. Puede ser una cadena de texto que representa el nombre del puerto, como por
            #   ejemplo 'COM3' en Windows o '/dev/ttyACM0' en Linux. El nombre del parámetro no se indica
            #   explícitamente.
            # - timeout (opcional): Especifica el tiempo máximo de espera (en segundos) para establecer la
            #   conexión con la placa Arduino. Si no se especifica, se utilizará un valor predeterminado.
            # - baudrate: Este parámetro establece la velocidad de comunicación en baudios para la
            #   comunicación entre la computadora y el microcontrolador. Los baudios representan la cantidad
            #   de bits que se pueden transmitir por segundo.
            #       - El valor que utiliza la librería Standard Firmata por default es de 57600, pero
            #         también se puede usar otros valores como 9600, 115200, etc. Pero esto debería ser
            #         cambiado igual en el código Arduino de la librería que se sube al Arduino.
            # - bytesize, parity y stopbits (opcionales): Estos parámetros permiten configurar la
            #   transmisión serial y se utilizan en conjunto para establecer cómo se transmiten los datos
            #   entre la computadora y la placa de desarrollo.
            self.micro_board = pyfirmata.Arduino(portSelected)
            print("La conexión Pyfirmata ha sido exitosa: ", self.micro_board)
            #Instancia de la clase Iterator, que hereda de la clase util y ambas pertenecen a la librería
            #pyfirmata, dicho objeto permite al programa percibir, capturar y procesar los cambios que
            #ocurran en los pines de entrada de la placa, para ello su constructor recibe como parámetro un
            #objeto pyfirmata.Board que ya haya realizado una conexión serial entre la computadora y la
            #tarjeta de desarrollo.
```

```
            self.it = pyfirmata.util.Iterator(self.micro_board)
            #pyfirmata.util.Iterator.start(): Método utilizado para iniciar el proceso de lectura de datos
            #entrantes desde la placa de desarrollo previamente conectada al ordenador de forma serial con
            #el constructor pyfirmata.Arduino().
            self.it.start()
            #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)
            #SELECCIÓN DE LOS PINES QUE SE QUIERE CONTROLAR Y/O MONITOREAR:
            #pyfirmata.Arduino.get_pin(): Método utilizado para acceder a un pin específico de la placa de
            #desarrollo con la que ya se ha realizado una conexión serial. Indicando si este es analógico o
            #digital, su número y si será utilizado como entrada o salida siguiendo la sintaxis descrita a
            #continuación:
            # - 'analogico_o_digital:   numero_pin:     entrada_o_salida'
            #       - 'analogico:       numero_pin:     entrada'            = 'a: numero_pin: i'
            #       - 'analogico:       numero_pin:     salida'             = 'a: numero_pin: o'
            #       - 'digital:         numero_pin:     entrada'            = 'd: numero_pin: i'
            #       - 'digital:         numero_pin:     salida'             = 'd: numero_pin: o'
            #El número y asignación de pin analógico o digital varía dependiendo de la tarjeta de
            #desarrollo.
            self.analog_0 = self.micro_board.get_pin('a:0:i')                       #Pin A0: Entrada.
            self.analog_1 = self.micro_board.get_pin('a:1:i')                       #Pin A1: Entrada.
            self.digital_13 = self.micro_board.get_pin('d:13:o')                    #Pin 13: Salida.
            self.digital_12 = self.micro_board.get_pin('d:12:o')                    #Pin 12: Salida.
            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 SerialPorts(): 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 each 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.
        s = serial.Serial(port)     #Inicio de comunicación serial.
        #serial.Serial.close(): Método que cierra la comunicación serial. Es muy importante mencionar
        #que si no se ejecuta este método, el puerto serial se va a quedar bloqueado y no se podrá usar.
        s.close()                   #Terminación de la comunicación serial.
        #append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
        result.append(port)
    #Para identificar el tipo de excepción que ha ocurrido y utilizarlo en la instrucción except, se
    #puede utilizar la clase Exception, que es una clase incorporada en Python utilizada para describir
    #todos los tipos de excepciones, luego de colocar el nombre de la clase Exception se usa la palabra
```

```
            #reservada "as" seguida de un nombre de variable, esto nos permitirá acceder a la instancia de la
            #excepción y utilizarla dentro del except.
            except Exception as error:
                #type(clase).__name__: Esta instrucción no es un método, sino una expresión que se utiliza para
                #obtener el nombre de la clase de un objeto en Python, donde type(error) devuelve el tipo de
                #excepción en este caso ya que error es un objeto de una clase de excepción.
                # - __name__: Es un atributo especial en Python que se utiliza para obtener el nombre de la
                #   clase del objeto.
                print("Ocurrió el siguiente tipo de error al intentar conectarse a todos los puertos disponibles: ",
type(error).__name__)
                print("Este es el mensaje del error: ", error)
                #Aunque ocurra un error al tratar de encontrar todos los tipos de puertos, esto no significa
                #que el programa no vaya a funcionar, solo significa que no se ha podido conectar con todos los
                #puertos seriales que encontró en la computadora, muy seguramente porque puede que estos estén
                #siendo ya usados en otra cosa.
        #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


#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 ShowMessageError(): 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(message = 'Communication with the board failed', caption = 'Communication error',
                    style = 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 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 númerico entero.
            samples = int(self.spinCtrlTime.GetValue())
            umbralLed = int(self.spinCtrlUmbral.GetValue())
```

```
        #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:
            #pyfirmata.Arduino.get_pin().read(): Método que se utiliza para leer el valor actual de un pin
            #en la placa. Permite obtener el estado del pin, que puede ser un valor analógico o digital,
            #dependiendo de cómo esté configurado.
            # - Pin digital de entrada: Devolverá un valor booleano (True o False), que indica si el pin
            #   está en alto (encendido) o en bajo (apagado) respectivamente.
            # - Pin analógico de entrada: Devolverá un valor numérico que representa la lectura analógica en
            #   el pin.
            #       - Placa Arduino: Este valor suele estar en un rango de 0 a 1, donde 0 coresponde al
            #         valor binario 0 y 1 al valor binario 1023, que hace referencia a la resolución de 10
            #         bits del conversor analógico a digital (ADC) en la placa Arduino:
            #               - Resolución de 10 bits del ADC del Arduino: (2^10)-1 = 1023
            #Si se utiliza el método read() en un pin que está configurado como salida, el resultado puede
            #ser impredecible o no tener sentido.
            tensionBinariaA0 = self.analog_0.read()                         #Lee el pin analógico A0.
            tensionBinariaA1 = self.analog_1.read()                         #Lee el pin analógico A1.

            #CONVERSIÓN DE NUMEROS BINARIOS NUMÉRICOS DE TENSIÓN A VALORES DE TENSIÓN REALES:
            #float(): Método que convierte un tipo de dato cualquiera en númerico decimal.
            #Se realiza esta operación porque como el ADC del arduino lee de 0 a 5V y como tiene una
            #resolución de 10 bits permitiendo que en el ADC los valores de tensión se interpreten como
            #valores numéricos enteros que valen de 0 a (2^10)-1 = 1023, se hace una regla de 3 para que se
            #imprima el valor de la tensión en consola en vez del valor decimal binario. En esta operación
            #no es necesario dividir el resultado entre la resolución del ADC porque el valor que retorna el
            #método read ya está normalizado en el rango de 0 a 1 (correspondiente a los 1024 niveles)
            #Tensión = Tensión_decimal_read*(ValorMáximoTensión) = Tensión_decimal_read*(5)
            VoltsA0 = (float(tensionBinariaA0)*(self.highValueBoard))       #Tensión real pin A0.
            VoltsA1 = (float(tensionBinariaA1)*(self.highValueBoard))       #Tensión real pin A1.
            umbral = umbralLed/1000.0                                       #Tensión de umbral [V].

            #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.
            self.time = self.time + 1.0     #Variable intermedia que cuenta el tiempo de ejecución del programa.
            self.n = self.n + 1             #Variable que cuenta los datos recopilados por la GUI.
```

```
        #IMPRIMIR EN CONSOLA TIEMPO Y TENSIÓN:
        msg_console = str(self.n) + ".- Time: " + str(self.time) + " [s]" + "\t"    #Tiempo [s].
        msg_console+= "Voltage Pin A0: " + "{0:.5f}".format(VoltsA0)+" [V]\t"      #Tensión real pin A0.
        msg_console+= "Voltage Pin A1: " + "{0:.5f}".format(VoltsA1)+" [V]\t"      #Tensión real pin A1.
        msg_console+= "Umbral: " + "{0:.5f}".format(umbral)+" [V]"               #Tensión de umbral [V].
        print(msg_console)


        #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) + ", " + "{0:05f}".format(VoltsA0) + ", " +
                                                   "{0:05f}".format(VoltsA1) + ", " +
                                                   "{0:05f}".format(umbral))
        self.x = np.append(self.x, self.time)                #Vector tiempo [segundos].
        self.y0 = np.append(self.y0, VoltsA0)                #Vector tensión A0 [V].
        self.y1 = np.append(self.y1, VoltsA1)                #Vector tensión A1 [V].
        self.u = np.append(self.u, umbral)                   #Vector tensión umbral [mV].
        self.graph.update_graph(self.x, self.y0, self.y1, self.u)


        #Si la tensión que ingresa en el pin A0 es mayor al umbral, se enciende un led con el pin 13.
        if(VoltsA0 > umbral):
            #pyfirmata.Arduino.get_pin().write(): Método que se utiliza para escribir un valor en algún
            #pin digital especificado de la placa, controlando si su estado está en alto (1 o True) o
            #bajo (0 o False).
            # - Encender LED:   write(True)  o write(1).
            # - Apagar LED:     write(False) o write(0).
            #El método write() solamente se puede usar con pines de salida digital, si se quiere
            #controlar la salida de un pin analógico se debe usa el método analog_write().
            self.digital_13.write(1)                         #Led encendido en el pin 13.
            print("Led en el pin 13 Encendido")
        else:
            self.digital_13.write(0)                         #Led apagado en el pin 13.
        #Si la tensión que ingresa en el pin A1 es mayor al umbral, se enciende un led con el pin 12.
        if(VoltsA1 > umbral):
            self.digital_12.write(True)                      #Led encendido en el pin 12.
            print("Led en el pin 12 Encendido")
        else:
            self.digital_12.write(False)                     #Led apagado en el pin 12.
    except:
        print("No se pudo actualizar la gráfica")

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

            #pyfirmata.Arduino.exit(): Método que cierra la comunicación serial. Es muy importante mencionar
            #que si no se ejecuta este método, el puerto serial se va a quedar bloqueado y no se podrá usar.
            self.micro_board.exit()


#función OnStopClick(): 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.
    #widget.Show(): Método que sirve para mostrar un widget en la GUI.
    self.buttonStart.Show()         #Muestra el botón de START.
    self.stopAcquisition = True     #Variable booleana que indica si se ha presionado el botón de STOP.


#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 Data")

    #Instancia de la librería wxPython por medio del constructor de la clase FileDialog para ejecutar un
    #método que cree una ventana, 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.
```

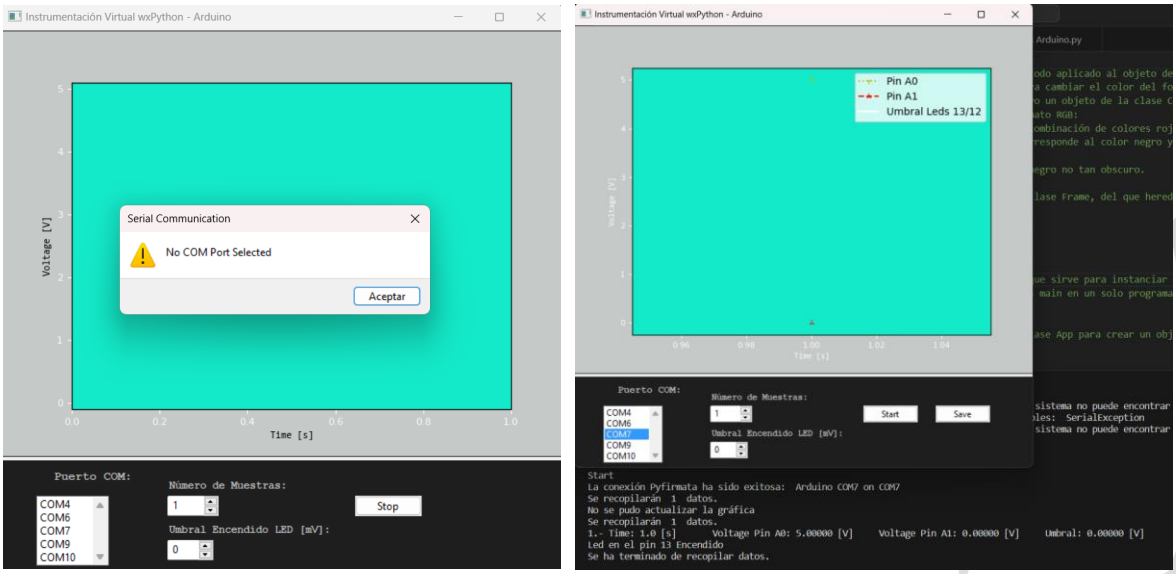
```
        #El objeto dialog que instancía la clase FileDialog perteneciente a la librería wxPython cuenta con los
        #siguientes métodos.
        # - dialog: Objeto que permite mostrar cuadros de diálogo.
        #   - dialog.ShowModal(): Muestra el cuadro de diálogo y bloquea la ejecución del 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
```

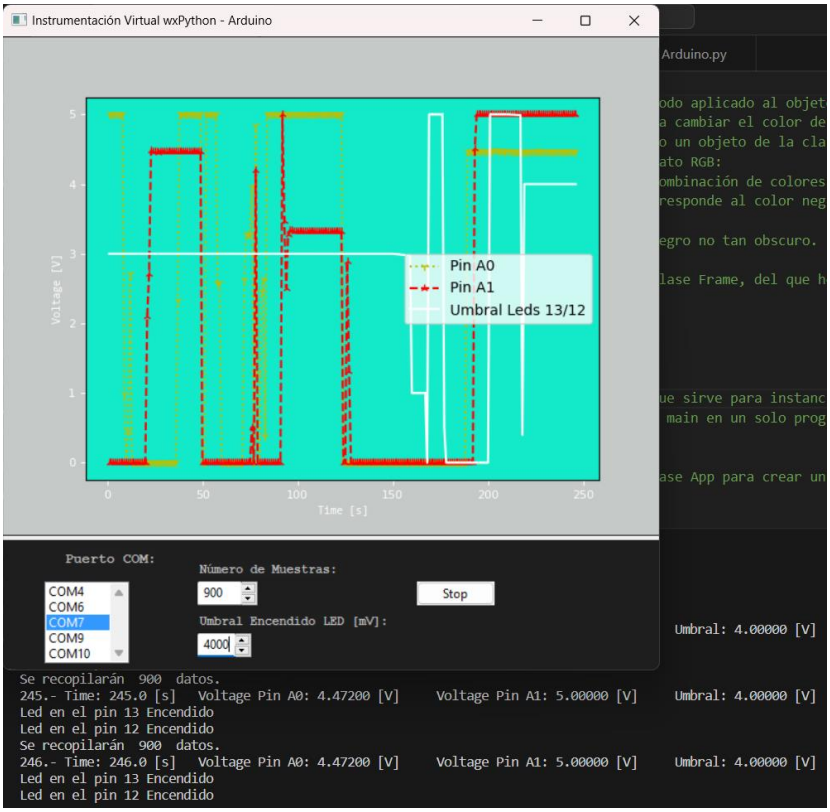
```
            #pueda ocurrir un error durante su ejecución.
            try:
                #open(): Método que sirve para abrir un archivo cualquiera, para ello es necesario indicar dos
                #parámetros, el primero se refiere a la ruta relativa o absoluta del archivo previamente creado
                #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("Library, wxPython, Pyfirmata" + "\n")
                myFile.write("Time [s], Voltage Pin A0 [V], Voltage Pin A1 [V], Umbral [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")


#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 Main(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 y un tamaño a través de su tercer parámetro.
        super().__init__(None, title = "Instrumentación Virtual wxPython - Arduino", 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 del que hereda 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.
```

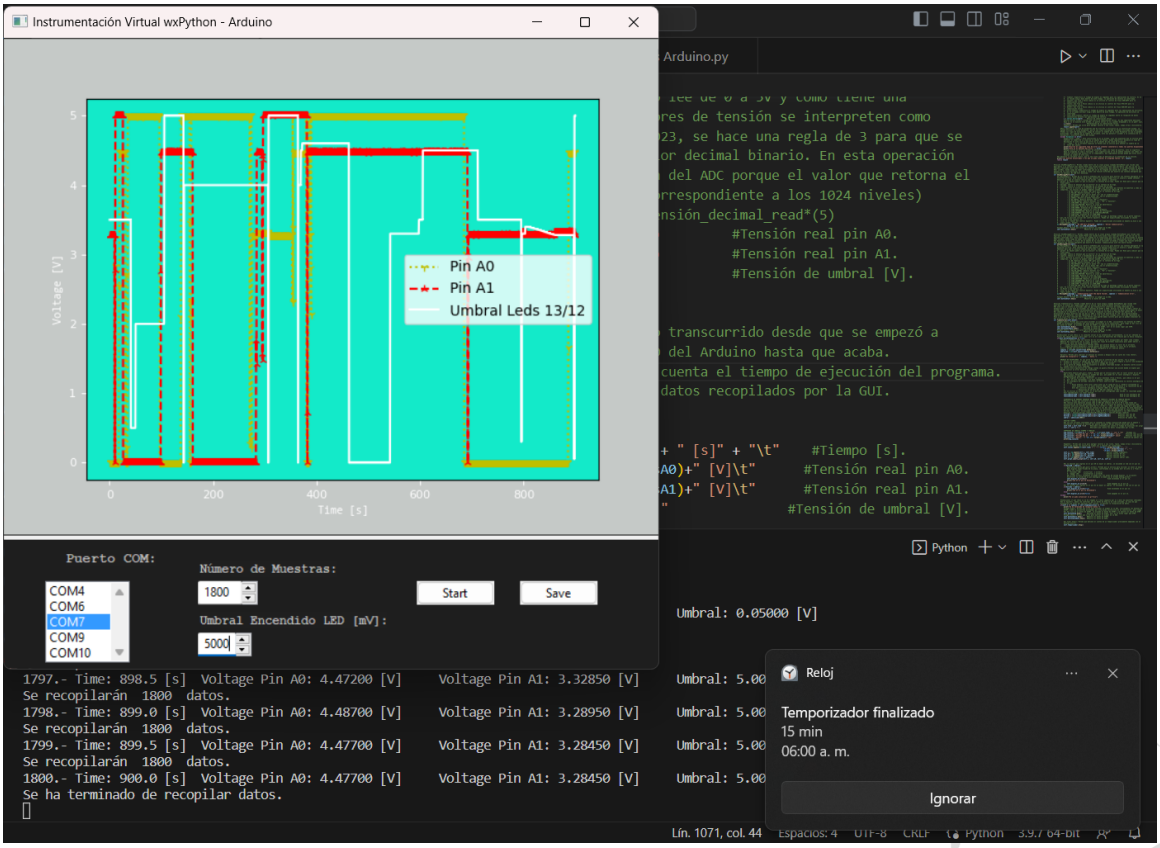
```
        #wx.Frame.show() = self.show(): Método aplicado al objeto de la clase Frame, del que hereda esta clase
        #propia para mostrar la ventana del GUI.
        self.Show()



#__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.
    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 = Main()
    #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()
```

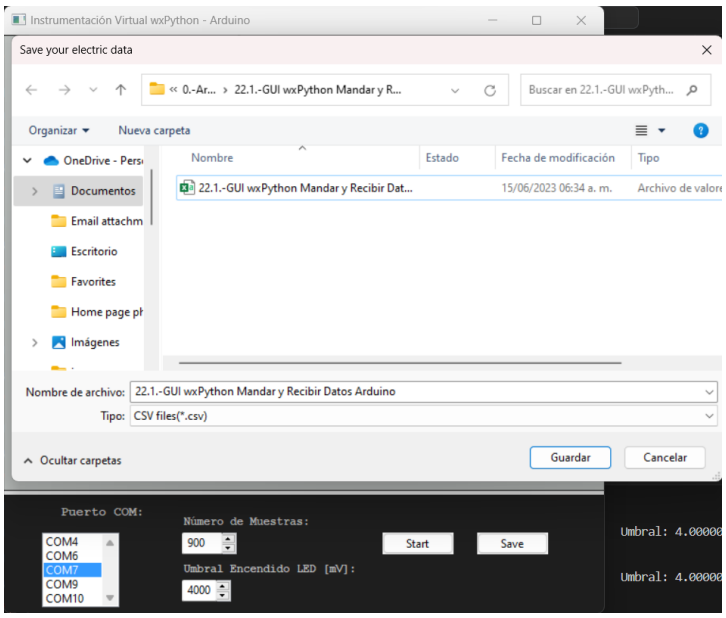
Resultado del Código Python: Interfaz Gráfica xwPython

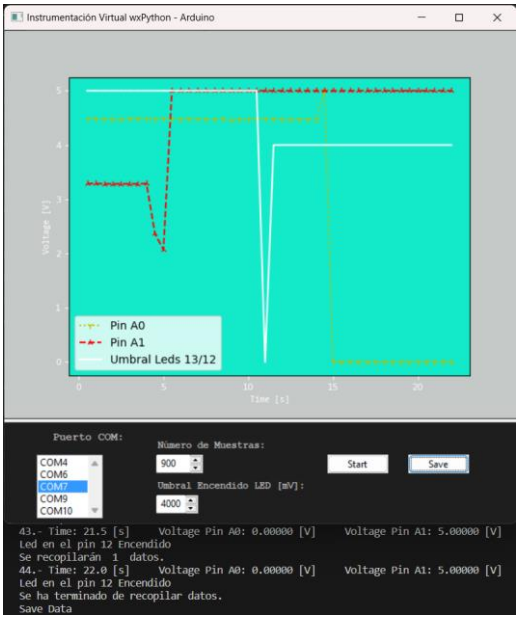


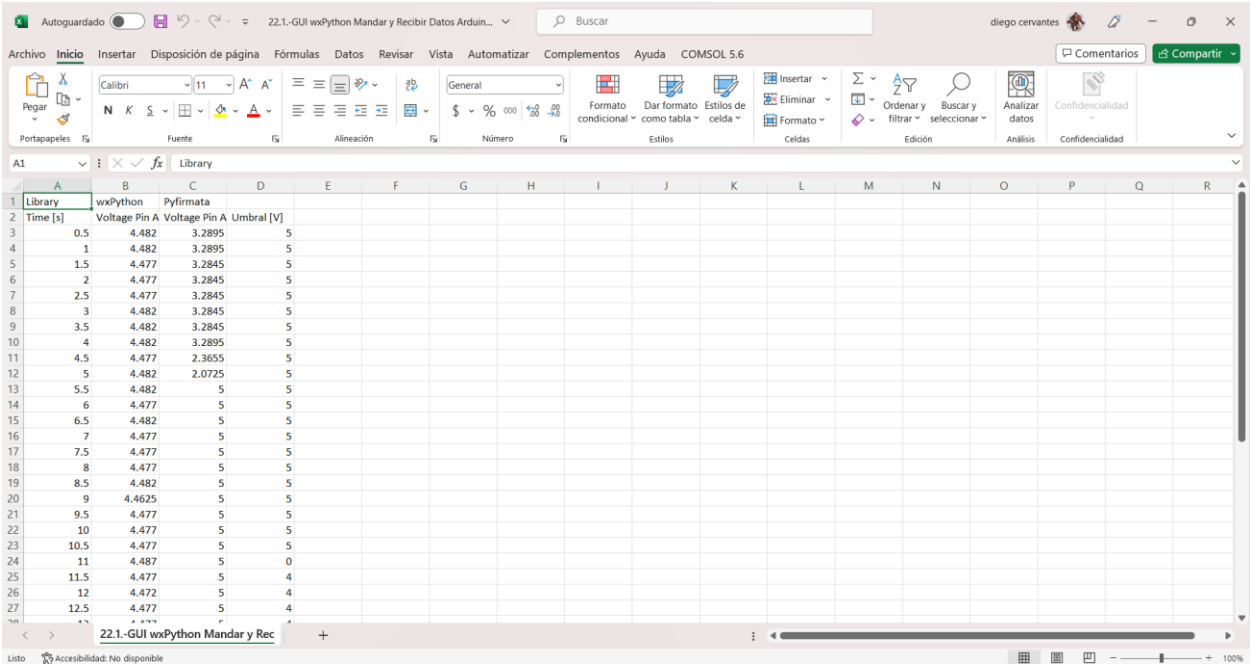


El tiempo del GUI con **wxPython** cuenta cada 0.5 segundos.

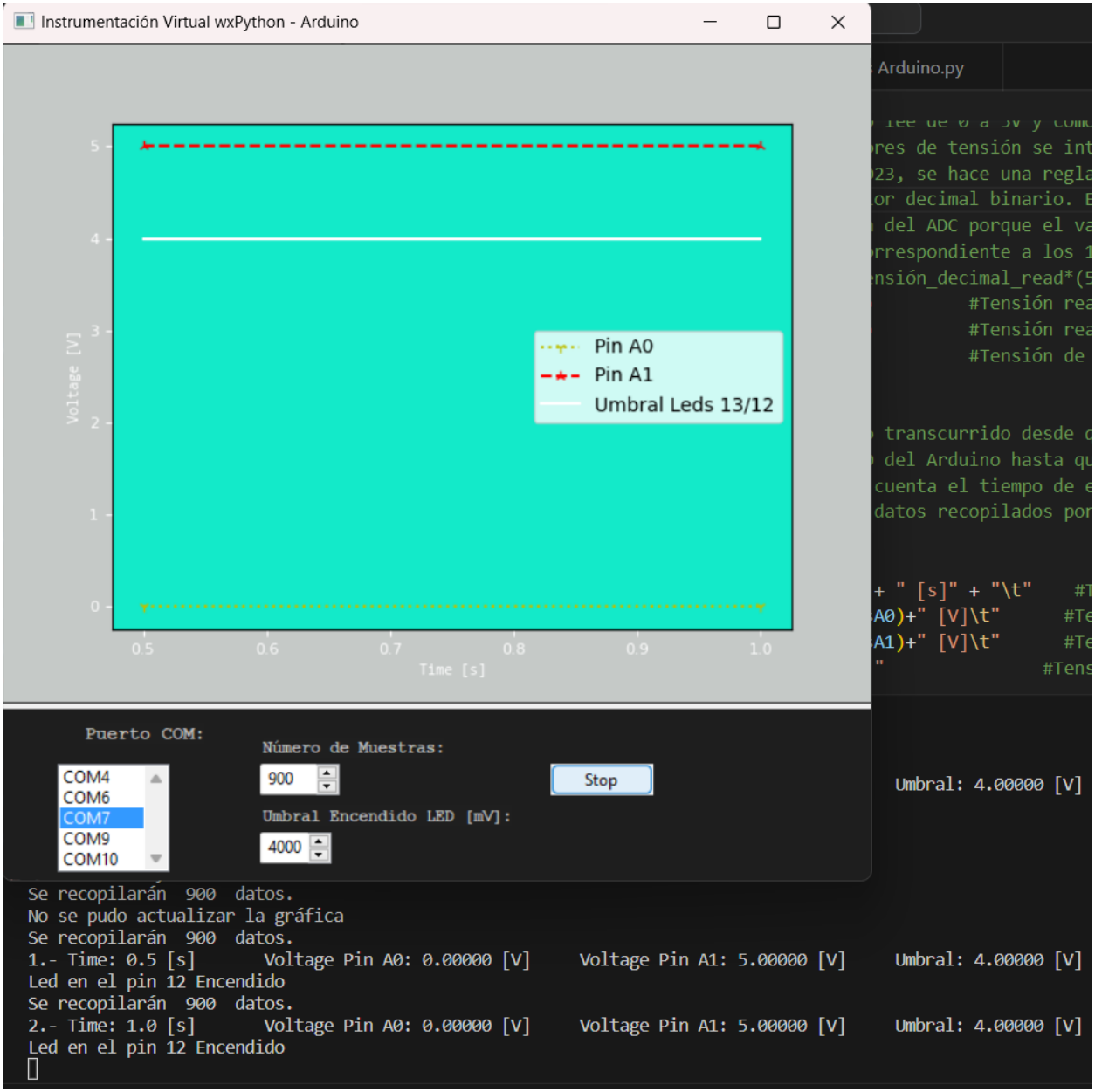








	A	B	C	D
1	Library	wxPython	Pyfirmata	
2	Time [s]	Voltage Pin A	Voltage Pin A	Umbral [V]
3	0.5	4.482	3.2895	5
4	1	4.482	3.2895	5
5	1.5	4.477	3.2845	5
6	2	4.477	3.2845	5
7	2.5	4.477	3.2845	5
8	3	4.482	3.2845	5
9	3.5	4.482	3.2845	5
10	4	4.482	3.2895	5
11	4.5	4.477	2.3655	5
12	5	4.482	2.0725	5
13	5.5	4.482	5	5
14	6	4.477	5	5
15	6.5	4.482	5	5
16	7	4.477	5	5
17	7.5	4.477	5	5
18	8	4.477	5	5
19	8.5	4.482	5	5
20	9	4.4625	5	5
21	9.5	4.477	5	5
22	10	4.477	5	5
23	10.5	4.477	5	5
24	11	4.487	5	0
25	11.5	4.477	5	4
26	12	4.472	5	4
27	12.5	4.477	5	4
28	13	4.477	5	4
29	13.5	4.477	5	4
30	14	4.477	5	4
31	14.5	5	5	4
32	15	0	5	4
33	15.5	0	5	4
34	16	0	5	4
35	16.5	0	5	4
36	17	0	5	4
37	17.5	0	5	4
38	18	0	5	4
39	18.5	0	5	4
40	19	0	5	4
41	19.5	0	5	4
42	20	0	5	4
43	20.5	0	5	4
44	21	0	5	4
45	21.5	0	5	4
46	22	0	5	4



GUI **PyQt5** – Visual Studio Code (Logo Azul):

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


#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
```

```
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#LIBRERÍAS:
#PyQt5 - QtWidgets: La clase QtWidgets proporciona todos los elementos que conforman las interfaces gráficas
#(GUI) hechas con la librería PyQt5, entre dichas herramientas se incluyen:
# - Widgets: Elementos gráficos básicos de los que se conforma una GUI como lo son botones (QPushButton), cajas
#   de texto (QLineEdit), texto estático (QLabel), listas desplegables (QComboBox), barras de progreso
#   (QProgressBar), casillas de verificación (QCheckBox), imágenes (QLabel), etc.
#   - Diálogos: Los diálogos son elementos gráficos que  facilitan la interacción con el usuario, incluyen
#     widgets como cuadros de diálogo de mensajes (QMessageBox), cuadros de diálogo para seleccionar archivos o
#     directorios (QFileDialog), cuadros de diálogo de entrada de texto (QInputDialog), etc.
# - Layouts: Es un contenedor de PyQt5 que permite almacenar y organizar varios widgets, es el equivalente al
#   Panel de la librería PyQt5.
#           - Widget: Es un contenedor que puede almacenar widgets directamente, proporcionando funcionalidades
#             para mostrar, ocultar, establecer posición, tamaño, manejar eventos, etc. de los diferentes
#             botones, checkboxes, áreas de texto, comboboxes, radiobuttons, listboxes, etc.
# - Ventana: Es la ventana principal de la GUI, que a su vez contiene todos los contenedores con los widgets que
#   conforman la GUI, se declara a través de la clase QMainWindow de la librería pyQt5 y es el equivalente al
#   Frame de PyQt5.
from PyQt5 import QtWidgets
#PyQt5 - QtWidgets: Clase que incluye métodos para trabajar con temporizadores, tamaño de elementos, fechas,
#archivos, directorios, señales, hilos, subprocesos, etc.
from PyQt5 import QtCore
#matplotlib - Figure: La clase Figure es la base para crear y organizar los elementos gráficos en Matplotlib,
#que es una librería de graficación matemática.
#PyQt5 - QtGui: Clase que incluye clases y métodos para trabajar con gráficos, fuentes, colores, imágenes,
#íconos y otros elementos visuales utilizados en una interfaz gráfica (GUI) de PyQt5.
from PyQt5 import QtGui
#matplotlib - Figure: La clase Figure es la base para crear y organizar los elementos gráficos en Matplotlib,
#que es una librería de graficación matemática.
from matplotlib.figure import Figure
#matplotlib - FigureCanvasQTAgg: La clase FigureCanvasQTAgg proporcionada por la biblioteca Matplotlib en el
#módulo matplotlib.backends.backend_qt5agg se utiliza para mostrar y manejar gráficos generados por Matplotlib
#dentro de una ventana o contenedor de pyQt5. En este caso se utiliza para mostrar una gráfica que podría
#actualizar sus datos en tiempo real o no, en este caso se mantendrá estática.
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
import numpy as np #numpy: Librería que realiza operaciones matemáticas complejas (matriciales).
import 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.
#pyfirmata: Librería que permite la comunicación bidireccional entre Python y Arduino, brindando control y
#monitoreo de sus pines digitales y analógicos, envío de señales PWM, lectura y escritura de datos y establecer
#una comunicación a través de los protocolos I2C y Serial. La comunicación entre Arduino y Python se realiza
```

```
#utilizando el protocolo Firmata, el cual permite controlar y monitorear dispositivos conectados a una placa
#Arduino desde un software de computadora, para habilitarlo, primero se debe subir el programa:
#21.-Stardard_Firmata_Mandar_Datos_Arduino.ino a la placa de desarrollo a través del IDE de Arduino.
import pyfirmata

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

#GraficaPyQt5: Esta clase propia hereda de la clase FigureCanvasQTAgg, que pertenece a la librería PyQt5 y
#permite mostrar gráficos generados por Matplotlib en un Widget de PyQt5, permitiendo manejar eventos de
#interacción del usuario, como hacer zoom, seleccionar puntos en el gráfico, etc.
class GraficaPyQt5(FigureCanvasQTAgg):
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor. Los parámetros que puede recibir el constructor de
    #la clase son los siguientes:
    # - figure: La instancia de la clase Figure que se va a asociar con el lienzo. Por defecto, se establece en
    #   None. Este parámetro es el primero y no usa la sintaxis: figure = valor, solamente se pone.
    # - parent: El widget padre al que se asocia el lienzo. Por defecto, se establece en None.
    # - width: El ancho del lienzo en píxeles o pulgadas.
    # - height: La altura del lienzo en píxeles o pulgadas.
    # - dpi: La resolución del lienzo en puntos por pulgada. Por defecto, se establece en 80 y puede afectar al
    #   tamaño de la gráfica.
    # - bgcolor: Indica el color de fondo del lienzo. Puede ser especificado a través de las siguientes formas:
    #       - Nombre color: Por medio de un string se pueden declarar valores que reconoce Matplotlib como por
    #         ejemplo, white', 'red', 'blue', etc.
    #       - Tupla RGB: Por medio de una tupla se especifican los tonos de color RGB (Rojo, Verde Azul) con
    #         valores que van de 0 a 1, como por ejemplo, (1, 1, 1) para blanco.
    def __init__(self, parent = None, width = None, height = None, dpi = 100, bgcolor = (0.077, 0.025, 0.024)):
        #Figure(): Constructor de la clase Figure perteneciente a la librería Matplotlib, usado para crear un
        #lienzo en el que se puedan dibujar gráficos, actúando como un contenedor para los subgráficos.
        #A este objeto se le pueden asignar los parámetros indicados en el constructor de la clase
        #FigureCanvasQTAgg.
        fig = Figure(figsize = (width, height), dpi = dpi)
        #matplotlib.Figure().set_facecolor(): Método que sirve para establecer el color de fondo de una gráfica.
        #A este objeto se le puede asignar el parámetros bgcolor indicado en el constructor de la clase
        #FigureCanvasQTAgg.
        fig.set_facecolor(bgcolor)
        #matplotlib.Figure().add_subplot(): Método aplicado a un objeto de la clase Figure, perteneciente a la
```

```
        #librería Matplotlib, lo que hace es agregar un subgráfico al lienzo vacío, los números de su parámetro
        #lo que indican es:
        # - Primer Número:  Indica el número de filas de las subgráficas.
        # - Segundo Número: Indica el número de columnas de las subgráficas.
        # - Tercer Número:  Indica el índice de la subgráfica que se está creando en específico.
        #El parámetro 111 crea una sola gráfica de una columna con un solo espacio para mostrar una gráfica.
        #Se declaran como self.nombreObjeto los widgets a los que se les vaya a extraer o introducir datos en el
        #transcurso del funcionamiento de la interfaz gráfica.
        self.axes = fig.add_subplot(111)
        #matplotlib.Figure().add_subplot().set_xlabel(): Método para indicar el texto que aparece en el eje
        #horizontal de la gráfica, recibe los siguientes parámetros:
        # - xlabel: Especifica el texto que se mostrará en el eje x.
        # - fontname: Indica el estilo de la fuente:
        #       - Nombres de tipos de letra estándar: "Arial", "Times New Roman", "Helvetica", "Courier",
        #         "Monospaced", "Consolas", etc. Estos nombres deben ser compatibles con los tipos de letra
        #         instalados en el sistema operativo.
        #       - Nombres de tipos de letra genéricos: "serif", "sans-serif", "monospace", etc.
        #       - Rutas de archivo: Si se tiene un archivo de tipo de letra personalizado, se puede especificar
        #         la ruta del archivo como el valor de fontname.
        # - fontsize: Indica el tamaño de la fuente.
        # - labelpad: Especifica el espaciado entre la etiqueta del eje x y el eje en sí
        # - color: Indica el color del texto colocado en el eje x, para ello es válido usar colores:
        #       - Básicos CSS: como "red", "blue", "green", etc.
        #       - Colores hexadecimales: "#FF0000", "#00FF00", etc.
        #       - Colores RGB: "rgb(255,255,255)" para el color blanco y "rgb(0,0,0)" para el negro.
        self.axes.set_xlabel(xlabel = "Time [s]", fontname = "Consolas", fontsize = 8, color = "white")
        #matplotlib.Figure.add_subplot().set_ylabel(): Método para indicar el texto que aparece en el eje y.
        self.axes.set_ylabel(ylabel = "Voltage [V]", fontname = "Consolas", fontsize = 8, color = "white")
        #matplotlib.Figure().add_subplot().tick_params(): Método para acceder a las propiedades gráficas de los
        #números que aparecen en los ejes de la gráfica creada con la librería matplotlib.
        # - axis: Indica el eje que se quiere afectar, ya sea 'x', 'y', 'both' o 'all'.
        # - colors: Indica el color de los números colocados en el eje x o y, para ello es válido usar colores:
        #       - Básicos CSS: como "red", "blue", "green", etc.
        #       - Colores hexadecimales: "#FF0000", "#00FF00", etc.
        #       - Colores RGB: "rgb(255,255,255)" para el color blanco y "rgb(0,0,0)" para el negro.
        # - labelsize: Permite especificar el tamaño de letra en pixeles.
        # - pad: espacio entre las marcas y las etiquetas en puntos.
        self.axes.tick_params(axis = "x", colors = "white", labelsize = 8)
        self.axes.tick_params(axis = "y", colors = "white", labelsize = 8)
        #matplotlib.Figure.add_subplot().set_facecolor(): Método para indicar el color de fondo de la gráfica,
        #el cual puede ser indicado por los mismos colores previamente mencionados en el método plot() o se
        #pueden usar los siguientes con el código xkcd:
        # - Colores: xkcd:aqua, xkcd:aquamarine, xkcd:azure, xkcd:beige, etc. Los colores se obtienen de:
        #https://matplotlib.org/stable/tutorials/colors/colors.html
        self.axes.set_facecolor('#180000')
```

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



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

        #Instancia de la clase GraficaPyQt5 para agregar la gráfica al Window, se le debe pasar como parámetro
        #al constructor de la clase GraficaPyQt5, los parámetros que se quiera editar para que no se ejecuten
        #los que vienen declarados por defecto, los que no se editen, se ejecutarán con el valor que fueron
        #declarados arriba en el constructor.
        self.canvas = GraficaPyQt5(self, width = 5, height = 4, dpi = 120)

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

```
        #   perteneciente a la librería QtGui para que se pueda agregar un ícono.
        # - checkable: Especifica si el botón funciona como un switch (que mantiene su estado) o como un push
        #   button (que no mantiene su estado a menos que se mantenga presionado).
        #       - checkable = True:     Botón tipo switch.
        #       - checkable = False:    Botón tipo push button.
        # - autoDefault: Indica si el botón es el predeterminado del diálogo. Si se establece en True, el botón
        #   responderá automáticamente a la tecla "Enter", sino no lo hará.
        # - flat: Con False se indica que el botón se muestre sin un marco, con True aparece el marco.
        # - menu: Especifica el menú desplegable asociado al botón.
        # - iconSize: Establece el tamaño del ícono del botón, para ello recibe un objeto QSize:
        #       - QtCore.QSize(ancho, alto): Objeto que indica el tamaño del ícono.
        #CREACIÓN DE ÍCONO PARA INCLUIR EN EL BOTÓN:
        #Variable que guarda el directorio y el nombre del archivo creado, se reemplazan los guiones \ por /
        #para poder leer una imagen o cualquier otro archivo, se usa la dirección relativa o absoluta de un
        #directorio:
        # - Dirección relativa: Es una dirección que busca un archivo desde donde se encuentra la carpeta del
        #   archivo python actualmente, esta se debe colocar entre comillas simples o dobles.
        # - Dirección absoluta: Es una dirección que coloca toda la ruta desde el disco duro C o cualquier otro
        #   que se esté usando hasta la ubicación del archivo, la cual se debe colocar entre comillas simples o
        #   dobles.
        #   ..      : Significa que nos debemos salir de la carpeta donde nos encontramos actualmente.
        #   /       : Sirve para introducirnos a alguna carpeta cuyo nombre se coloca después del slash.
        #   .ext    : Se debe colocar siempre el nombre del archivo + su extensión.
        iconPath = "Archivos_Ejercicios_Python/Img/LogoBlancoDi_cer0.png"
        #PyQt5.QtGui.QIcon(): Constructor de la clase QIcon que hereda de la clase QtGui y perteneciente a la
        #librería PyQt5, usado para crear un objeto que ícono que pueda ser añadido a cualquier widget como lo
        #puede ser un botón, un texto estático, etc. El tamaño de dicha imagen será reducido automáticamente.
        logoDicer0 = QtGui.QIcon(iconPath)
        #Se declaran como self.nombreObjeto los widgets a los que se les vaya a extraer o introducir datos en el
        #transcurso del funcionamiento de la interfaz gráfica, en este caso se aplica al botón porque este va a
        #cambiar el texto que tiene escrito cuando sea presionado.
        self.btn_start = QtWidgets.QPushButton(text = "\t\t\t\tStart", icon = logoDicer0, iconSize = QtCore.QSize(30, 30))
        self.btn_stop = QtWidgets.QPushButton(text = "\t\t\t\tStop", icon = logoDicer0, iconSize = QtCore.QSize(30, 30))
        self.btn_save = QtWidgets.QPushButton(text = "\t\t\t\tSave", icon = logoDicer0, iconSize = QtCore.QSize(30, 30))
        #widget.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos, no todos) a los
        #widgets de una interfaz gráfica de usuario (GUI).
        # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
        #   esta no es admitida por PyQt5:
        #   background: qlineargradient(x1:punto_inicial, y1:punto_inicial, x2:punto_final, y2:punto_final, stop:0
rgb(R_inicial,G_inicial,B_inicial), stop:1 rgb(R_final,G_final,B_final));
        self.btn_start.setStyleSheet("background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,230,181), stop:1
rgb(150,0,0));")
        self.btn_stop.setStyleSheet("background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,255,255), stop:1
rgb(91,150,242));")
```

```
        self.btn_save.setStyleSheet("background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,255,255), stop:1
rgb(198,132,0));")
        #widget.Hide(): Método que sirve para esconder un widget en la GUI.
        self.btn_stop.hide()            #Esconde inicialmente el botón de STOP
        self.btn_save.hide()            #Esconde inicialmente el botón de SAVE

        #CREACIÓN DE LOS WIDGETS: Texto Estático, clase QLabel
        #Instancia de la librería PyQt5 por medio del constructor de la clase QLabel que hereda de la clase
        #QtWidgets y sirve para crear un widget que muestre un texto estático o una imagen en una interfaz
        #gráfica, se le deben indicar los siguientes parámetros cuando se usa para crear texto estático:
        # - parent: Especifica el widget padre del QLabel. Si se proporciona, el texto estático se colocará
        #   dentro del widget padre.
        # - text: Permite especificar el texto que se mostrará en el widget. Puede ser una cadena de texto en
        #   formato plano o enriquecido con etiquetas HTML.
        #Para dar estilo al QLabel cuando se utiliza para mostrar texto estático es mejor utilizar etiquetas
        #HTML que contengan un style que les dé estilo por medio de instrucciones CSS, además es importante
        #mencionar que para el style se deben usar comillas simples ('') para que no tenga conlficto con las
        #comillas dobles del parámetro text = "". PyQt5 acepta algunas instrucciones CSS pero no todas.
        lbl_port = QtWidgets.QLabel("<p style='background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(150,54,41),
stop:1 rgb(64,15,34)); font-size: 15px; font-family: Courier New, monospace; color: white;'>Puerto COM:</p>")
        lbl_samples = QtWidgets.QLabel("<p style='background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(150,54,41),
stop:1 rgb(64,15,34)); font-size: 15px; font-family: Courier New, monospace; color: white;'>Número de Muestras:</p>")
        lbl_umbral = QtWidgets.QLabel("<p style='background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(150,54,41),
stop:1 rgb(64,15,34)); font-size: 15px; font-family: Courier New, monospace; color: white;'>Umbral Encendido LED [mV]:</p>")
#Label del umbral

        #CREACIÓN DE LOS WIDGETS: Combo Box, Lista Desplegable
        #Instancia de la librería PyQt5 por medio del constructor de la clase QComboBox que hereda de la clase
        #QtWidgets y sirve para crear un widget que muestre una lista desplegable de elementos seleccionables
        #en una ventana o layout.
        self.cb_port = QtWidgets.QComboBox()    #Combo box para mostrar todos los puertos detectados
        #widget.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos, no todos) a los
        #widgets de una interfaz gráfica de usuario (GUI).
        # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
        #   esta no es admitida por PyQt5:
        #   background: qlineargradient(x1:punto_inicial, y1:punto_inicial, x2:punto_final, y2:punto_final, stop:0
rgb(R_inicial,G_inicial,B_inicial), stop:1 rgb(R_final,G_final,B_final));
        self.cb_port.setStyleSheet("font-size: 15px; font-family: Courier New, monospace; color: white; background:
qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,230,181), stop:1 rgb(150,0,0));")
        #QtWidgets.QComboBox.addItems(): Método que se utiliza para agregar una lista de elementos al combo box.
        #Este puede recibir como parámetro directamente una lista, tupla o diccionario que representen los
        #elementos que se agregarán al combo box o puede recibir una función propia que cree dicha lista, para
        #luego añadir dichos elementos creados por la función al QComboBox.
        #Se ejecuta la función propia SerialPorts() declarada fuera del constructor pero dentro de la clase
        #MainWindow para obtener los puertos disponibles del ordenador actual por medio de la clase serial.
```

```
        self.cb_port.addItems(self.SerialPorts())

        #CREACIÓN DE LOS WIDGETS: Spin Box, control numérico
        #Instancia de la librería PyQt5 por medio del constructor de la clase QSpinBox que hereda de la clase
        #QtWidgets y sirve para crear un widget  que muestre un selector de números en una ventana o layout.
        spb_samples = QtWidgets.QSpinBox()      #Número de muestras.
        spb_umbral = QtWidgets.QSpinBox()       #Umbral de encendido para el led, indicado en milivolts [mV].
        #QtWidgets.QSpinBox.setMinimum(Valor_Mínimo): Método que indica el valor mínimo permitido en el control
        #numérico SpinBox.
        spb_samples.setMinimum(1)               #Mínimo número de muestras.
        spb_umbral.setMinimum(0)                #Umbral de encendido mínimo para el led.
        #QtWidgets.QSpinBox.setMaximum(Valor_Máximo): Método que indica el valor máximo permitido en el control
        #numérico SpinBox. El máximo de datos que puede recopilar un archivo de Excel sin fallar son 32,000.
        spb_samples.setMaximum(32000)           #Máximo número de muestras.
        spb_umbral.setMaximum(5000)             #Umbral de encendido máximo para el led.
        #QtWidgets.QSpinBox.setSingleStep(Intervalo): Indica de cuánto en cuanto avanza el valor numérico del
        #SpinBox.
        spb_samples.setSingleStep(1)            #Intervalo número de muestras.
        spb_umbral.setSingleStep(1)             #Intervalo umbral de encendido del LED.
        #widget.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos, no todos) a los
        #widgets de una interfaz gráfica de usuario (GUI).
        # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
        #   esta no es admitida por PyQt5:
        #   background: qlineargradient(x1:punto_inicial, y1:punto_inicial, x2:punto_final, y2:punto_final, stop:0
rgb(R_inicial,G_inicial,B_inicial), stop:1 rgb(R_final,G_final,B_final));
        spb_samples.setStyleSheet("font-size: 15px; font-family: Courier New, monospace; color: white; background:
qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,230,181), stop:1 rgb(150,0,0));")
        spb_umbral.setStyleSheet("font-weight: 900; font-size: 16px; font-family: Courier New, monospace; color: aqua;
background: qlineargradient(x1:0, y1:1, x2:0, y2:0, stop:0 rgb(255,230,181), stop:1 rgb(150,0,0));")

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

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

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

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

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

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

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

self.high_value_board = 5.0     #Valor de tensión Máxima = 5V

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

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

#QtWidgets.QSpinBox.minimum(): Método que obtiene el valor mínimo en un control numérico SpinBox.
#La variable que guarda el número de muestras se inicializa con el valor mínimo del SpinBox.
self.items = spb_samples.minimum()      #Variable que indica el número de muestras.
self.umbralLed = spb_umbral.minimum()   #Variable que indica la tensión de encendido del led [mV].
```

```
#función OnStartClick(): Método creado dentro de la clase propia llamada MainWindow que recibe como
#parámetro el evento que lo activa, para posteriormente ejecutar cierta acción.
#En este caso el evento es activado por dar un clic sobre el botón de Start y lo que hace es primero checar
#si la comunicación serial está abierta, para poderla cerrar y volverla a abrir, luego checa si se ha
#seleccionado un puerto del ComboBox y si esto es cierto, iniciliza el temporizador y establece una
#comunicación serial con el puerto seleccionado, si no ha sido elegido ningún puerto del ListBox, muestra
#una ventana emergente que indique que no se ha seleccionado ningún puerto, además si es que ha ocurrido un
#error al intentar establecer la comunicación serial con el Arduino, mostrará un mensaje de error en una
#ventana emergente que indique tal cosa.
def OnStartClick(self):
    #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
    print("\nStart")
    print("Se recopilarán", self.items, "datos.")     #Se indica el número de muestras a recopilar.
    print("Con un umbral inicial de", self.umbralLed, "[mV].")
    self.stp_acq = False            #Variable booleana que indica si se ha presionado el botón STOP.
    #MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
    # - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción
    #   durante su ejecución, el programa brinca al código del except
    # - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción
    #   cuando ocurra el error esperado.
    #Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que pueda
    #ocurrir un error durante su ejecución.
    try:
        #PYFIRMATA: CONEXIÓN SERIAL, CONTROL Y MONITOREO DE PINES DE UNA TARJETA DE DESARROLLO
        #Instancia de la clase Arduino, que pertenece a la librería pyfirmata, dicho objeto proporciona
        #métodos para comunicarse con placas Arduino utilizando el protocolo Firmata, permitiendo así el
        #control y monitoreo de sus pines digitales y analógicos, realizar el envío de señales PWM, lectura
        #y/o escritura de datos y establecer una comunicación a través de los protocolos I2C y Serial, la
        #conexión serial con los microcontroladores se realiza utilizando la clase pyfirmata.Arduino(),
        #independientemente del tipo de microcontrolador que se esté utilizando, ya que la librería
        #pyfirmata proporciona una interfaz común para comunicarse con diferentes placas de desarrollo,
        #incluyendo Arduino, Raspberry Pi, Intel Galileo, PyCARD, etc. Para ello su constructor recibe los
        #siguientes parámetros:
        # - port (obligatorio): Especifica el puerto de comunicación a través del cual se conectará la placa
        #   Arduino. Puede ser una cadena de texto que representa el nombre del puerto, como por ejemplo
        #   'COM3' en Windows o '/dev/ttyACM0' en Linux. El nombre del parámetro no se indica
        #   explícitamente.
        # - timeout (opcional): Especifica el tiempo máximo de espera (en segundos) para establecer la
        #   conexión con la placa Arduino. Si no se especifica, se utilizará un valor predeterminado.
        # - baudrate: Este parámetro establece la velocidad de comunicación en baudios para la comunicación
        #   entre la computadora y el microcontrolador. Los baudios representan la cantidad de bits que se
        #   pueden transmitir por segundo.
        #       - El valor que utiliza la librería Standard Firmata por default es de 57600, pero también se
```

```
        #         puede usar otros valores como 9600, 115200, etc. Pero esto debería ser cambiado igual en
        #         el código Arduino de la librería que se sube al Arduino.
        # - bytesize, parity y stopbits (opcionales): Estos parámetros permiten configurar la transmisión
        #   serial y se utilizan en conjunto para establecer cómo se transmiten los datos entre la
        #   computadora y la placa de desarrollo.
        self.micro_board = pyfirmata.Arduino(self.com_port, baudrate = 57600)   #Conexión serial.
        print("La conexión Pyfirmata ha sido exitosa: ", self.micro_board)
        #Instancia de la clase Iterator, que hereda de la clase util y ambas pertenecen a la librería
        #pyfirmata, dicho objeto permite al programa percibir, capturar y procesar los cambios que ocurran
        #en los pines de entrada de la placa, para ello su constructor recibe como parámetro un objeto
        #pyfirmata.Board que ya haya realizado una conexión serial entre la computadora y la tarjeta de
        #desarrollo.
        self.portListener = pyfirmata.util.Iterator(self.micro_board)           #Monitoreo de pines.
        #pyfirmata.util.Iterator.start(): Método utilizado para iniciar el proceso de lectura de datos
        #entrantes desde la placa de desarrollo previamente conectada al ordenador de forma serial con el
        #constructor pyfirmata.Arduino().
        self.portListener.start()
        #SELECCIÓN DE LOS PINES QUE SE QUIERE CONTROLAR Y/O MONITOREAR:
        #pyfirmata.Arduino.get_pin(): Método utilizado para acceder a un pin específico de la placa de
        #desarrollo con la que ya se ha realizado una conexión serial. Indicando si este es analógico o
        #digital, su número y si será utilizado como entrada o salida siguiendo la sintaxis descrita a
        #continuación:
        # - 'analogico_o_digital:   numero_pin:     entrada_o_salida'
        #       - 'analogico:       numero_pin:     entrada'            = 'a: numero_pin: i'
        #       - 'analogico:       numero_pin:     salida'             = 'a: numero_pin: o'
        #       - 'digital:         numero_pin:     entrada'            = 'd: numero_pin: i'
        #       - 'digital:         numero_pin:     salida'             = 'd: numero_pin: o'
        #El número y asignación de pin analógico o digital varía dependiendo de la tarjeta de desarrollo.
        self.analog_0 = self.micro_board.get_pin('a:0:i')                       #Pin A0: Entrada.
        self.analog_1 = self.micro_board.get_pin('a:1:i')                       #Pin A1: Entrada.
        self.digital_13 = self.micro_board.get_pin('d:13:o')                     #Pin 13: Salida.
        self.digital_12 = self.micro_board.get_pin('d:12:o')                     #Pin 12: Salida.
    except:
        #PyQt5.QtWidgets.QMessageBox(): Método de la librería PyQt5 que se utiliza para mostrar una ventana
        #emergente en la interfaz gráfica. Esta ventana de diálogo muestra un mensaje específico al usuario
        #y puede contener botones para que el usuario realice una acción, como aceptar, cancelar, etc.
        dlg_board=QtWidgets.QMessageBox()
        #PyQt5.QtWidgets.QMessageBox.setWindowTitle(): Método para colocar un título en la ventana creada
        #con la librería PyQt5.
        dlg_board.setWindowTitle("Error en Instrumentación con pyfirmata uy no!")
        #PyQt5.QtWidgets.QMessageBox.setText(): Método que se utiliza para establecer el texto principal de
        #un cuadro de diálogo QMessageBox en PyQt5. Este método recibe el siguiente parámetro:
        # - text: Indica el texto que se mostrará en el cuerpo principal del cuadro de diálogo. Puede ser
        #   una cadena de texto o admitir el formato HTML para formatear el texto. El nombre del parámetro
        #   no se menciona explícitamente.
```

```
            str_dlg_board = "<h3>No tienes ningún puerto seleccionado!</h3>"
            str_dlg_board += "<h4>O la placa no se conectó correctamente...</h4>"
            dlg_board.setText(str_dlg_board)
            #PyQt5.QtWidgets.QMessageBox.setStandardButtons(): Método usado para establecer los botones estándar
            #que se mostrarán en el cuadro de diálogo, esto se realiza a través de los siguientes parámetros:
            # - QtWidgets.QMessageBox.Ok: Botón "Aceptar".
            # - QtWidgets.QMessageBox.Open: Botón "Abrir".
            # - QtWidgets.QMessageBox.Save: Botón "Guardar".
            # - QtWidgets.QMessageBox.Cancel: Botón "Cancelar".
            # - QtWidgets.QMessageBox.Close: Botón "Cerrar".
            # - QtWidgets.QMessageBox.Yes: Botón "Sí".
            # - QtWidgets.QMessageBox.No: Botón "No".
            # - QtWidgets.QMessageBox.Abort: Botón "Abortar".
            # - QtWidgets.QMessageBox.Retry: Botón "Reintentar".
            # - QtWidgets.QMessageBox.Ignore: Botón "Ignorar".
            # - QtWidgets.QMessageBox.Reset: Botón "Restablecer".
            # - QtWidgets.QMessageBox.Help: Botón "Ayuda".
            # - QtWidgets.QMessageBox.Apply: Botón "Aplicar".
            # - QtWidgets.QMessageBox.YesToAll: Botón "Sí a todo".
            # - QtWidgets.QMessageBox.NoToAll: Botón "No a todo".
            # - QtWidgets.QMessageBox.SaveAll: Botón "Guardar todo".
            # - QtWidgets.QMessageBox.Default: Botón "Predeterminado".
            # - QtWidgets.QMessageBox.RestoreDefaults: Botón "Restaurar predeterminados".
            #Puedes combinar varios botones utilizando la compuerta lógica OR (|) para mostrar varios botones en
            #el cuadro de diálogo.
            dlg_board.setStandardButtons(QtWidgets.QMessageBox.Ok)
            #PyQt5.QtWidgets.QMessageBox.setIcon(): Método que se utiliza para establecer el ícono que se
            #muestra en un cuadro de diálogo QMessageBox. Recibe un parámetro que especifica el ícono a mostrar.
            #Los valores posibles para el parámetro son:
            # - QtWidgets.QMessageBox.NoIcon: No se muestra ningún ícono.
            # - QtWidgets.QMessageBox.Information: Muestra un ícono de información.
            # - QtWidgets.QMessageBox.Warning: Muestra un ícono de advertencia.
            # - QtWidgets.QMessageBox.Critical: Muestra un ícono de error/crítico.
            # - QtWidgets.QMessageBox.Question: Muestra un ícono de pregunta.
            dlg_board.setIcon(QtWidgets.QMessageBox.Warning)
            #PyQt5.QtWidgets.QMessageBox.exec_(): Método para que se ejecute en un loop infinito el GUI,
            #logrando así que no se ejecute una vez y luego cierre por sí solo, sino que solo se cierre
            #solamente al dar clic en el tache de la ventana emergente.
            dlg_board.exec_()
            #Reinicio de la variable que guarda los datos de tensión obtenidos del pin A0 del Arduino.
            self.micro_board = None

        #Condicional if que checa si ya se ha seleccionado algún puerto y además se ha iniciado la conexión
        #serial, empezado ya a recopilar los datos de tensión del pin A0; si ésta es diferente de None, inicia
        #el conteo con un temporizador y empieza a almacenar el tiempo transcurrido y los datos de tensión
```

```
        #recabados en una lista que los relacione entre sí.
        if(self.com_port != "" and self.micro_board != None):
            #widget.Hide(): Método que sirve para esconder un widget en la GUI.
            self.btn_start.hide()       #Esconde el botón de START.
            self.btn_save.hide()        #Esconde el botón de SAVE.
            #widget.Show(): Método que sirve para mostrar un widget en la GUI.
            self.btn_stop.show()        #Muestra el botón de STOP.

            #Condicional if que comprueba que el valor de la variable que cuenta los datos recopilados por la
            #GUI es igual a cero, para que cuando esto sea cierto, se reinicie el valor de todas las variables
            #antes de comenzar de nuevo una recopilación de datos.
            if(self.count == 0):
                self.x = np.array([])   #Reinicio del numpy array del eje horizontal (x = tiempo [s]).
                self.y0 = np.array([])  #Reinicio del eje vertical perteneciente al pin A0 (y1 = tensión [V]).
                self.y1 = np.array([])  #Reinicio del eje vertical perteneciente al pin A1 (y2 = tensión [V]).
                self.u = np.array([])   #Reinicio del eje vertical perteneciente al umbral (u = tensión [mV]).
                self.values = []        #Reinicio de la lista que guarda los valores de tiempo y tensión.

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

                #CREACIÓN DE WIDGETS: Temporizador
                #Instancia de la librería PyQt5 por medio del constructor de la clase Timer, que hereda de la
                #clase QtCore y se utiliza para crear un widget de tipo temporizador.
                self.timer = QtCore.QTimer()
                #INICIACIÓN DEL TEMPORIZADOR:
                #QtCore.QTimer.setInterval(intervalo): Método que indica el intervalo de conteo de un
                #temporizador en milisegundos.
                self.timer.setInterval(self.period)
                #Instancia_Widget.evento_señal.connect(función_que_reacciona_al_evento): Este método se utiliza
                #para enlazar un evento a un controlador de eventos, que es una función que se ejecuta cuando
                #ocurra el evento, para ello se usa el nombre del widget, seguido del evento de tipo señal que
                #detona el método, la palabra reservdada .connect() y entre paréntesis se coloca el nombre de la
                #función que ejecutará alguna acción cuando ese evento ocurra. Normalmente las funciones que
                #describen las acciones a realizar por los widgets de la GUI se encuentran dentro de esta misma
                #clase, pero fuera de su constructor.
                # - Tipos de Eventos en Python:
                #       - clicked: Señal emitida cuando se hace clic en un elemento, como un botón.
                #       - doubleClicked: Señal emitida cuando se hace doble clic en un elemento.
                #       - pressed: Señal emitida cuando se presiona un elemento, como un botón.
                #       - released: Señal emitida cuando se suelta un elemento, como un botón.
                #       - textChanged: Señal emitida cuando el texto de un elemento, como un campo de texto,
                #         cambia.
                #       - currentIndexChanged: Señal emitida cuando se cambia el índice seleccionado en un
                #         elemento,
                #         como en un menú desplegable.
```

```
            #       - activated: Señal emitida cuando se selecciona un elemento, como un elemento de un menú
            #         desplegable o una opción de una lista.
            #       - keyPressed: Señal emitida cuando se presiona una tecla en el teclado.
            #       - keyReleased: Señal emitida cuando se suelta una tecla en el teclado.
            #       - mousePressEvent: Señal emitida cuando se presiona un botón del mouse.
            #       - mouseReleaseEvent: Señal emitida cuando se suelta un botón del mouse.
            #       - mouseMoveEvent: Señal emitida cuando se mueve el mouse.
            #       - valueChanged: Señal emitida cuando se selecciona un nuevo elemento en un combo box
            #         (lista desplegable).
            #       - timeout: Señal emitida cuando transcurre cada intervalo de tiempo especificado en un
            #         temporizador.
            #Es importante mencionar que en  PyQt5, cuando se conecta una función a un evento, la función
            #conectada puede recibir argumentos adicionales proporcionados por la señal emitida. Estos
            #argumentos son transmitidos automáticamente por el sistema de señales y slots de PyQt5.
            # - Tipos de argumentos retornados al suceder un evento:
            #       - *args y **kwargs: Muchas señales en PyQt5 permiten enviar argumentos adicionales a
            #         través de *args (tupla de argumentos posicionales) y **kwargs (diccionario de
            #         argumentos de palabras clave). Estos parámetros pueden variar según la señal
            #         específica y su contexto de uso.
            #       - checked: Algunas señales, como "clicked" en un botón, pueden enviar el estado de
            #         alternancia del widget. Este parámetro indica si el widget está marcado y suele ser de
            #         tipo booleano.
            #       - text: En widgets de entrada de texto, como QLineEdit o QTextEdit, las señales pueden
            #         enviar el texto ingresado o modificado como parámetro.
            #       - index: En widgets que tienen índices o selecciones, como QComboBox o QListView, las
            #         señales pueden enviar el índice seleccionado como parámetro.
            #       - position: En widgets que trabajan con eventos de posición, como QMouseEvent, las
            #         señales pueden enviar la posición del cursor o del evento como parámetro.
            #Cada que transcurra el intervalo de tiempo indicado en el temporizador, se ejecuta una función.
            self.timer.timeout.connect(self.update_plot)    #Intervalo transcurrido  = update_plot()
            #QtCore.QTimer.start(): Método que inicializa el conteo de un temporizador, para ello,
            #previamente se tuvo que haber usado el método .setInterval() para indicar su intervalo de
            #conteo en milisegundos.
            self.timer.start()
            print("\nTime [s] \t\tVoltage Pin A0 [V] \t\tVoltage Pin A1 [V] \t\tUmbral [V]")


    #función update_plot(): Método creado dentro de la clase propia llamada MainWindow que recibe como
    #parámetro el evento que lo activa, para posteriormente ejecutar cierta acción. En este caso el método se
    #ejecuta Cada que transcurra el intervalo de tiempo indicado en el temporizador y lo que hace es actualizar
    #el estado de la gráfica, para que los datos recopilados se muestren en tiempo real.
    def update_plot(self):
        #MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
        # - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción
        #   durante su ejecución, el programa brinca al código del except
```

```
        # - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción
        #   cuando ocurra el error esperado.
        #Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que pueda
        #ocurrir un error durante su ejecución.
        try:
            #pyfirmata.Arduino.get_pin().read(): Método que se utiliza para leer el valor actual de un pin en la
            #placa. Permite obtener el estado del pin, que puede ser un valor analógico o digital, dependiendo
            #de cómo esté configurado.
            # - Pin digital de entrada: Devolverá un valor booleano (True o False), que indica si el pin está en
            #   alto (encendido) o en bajo (apagado) respectivamente.
            # - Pin analógico de entrada: Devolverá un valor numérico que representa la lectura analógica en el
            #   pin.
            #       - Placa Arduino: Este valor suele estar en un rango de 0 a 1, donde 0 coresponde al valor
            #         binario 0 y 1 al valor binario 1023, que hace referencia a la resolución de 10 bits del
            #         conversor analógico a digital (ADC) en la placa Arduino:
            #               - Resolución de 10 bits del ADC del Arduino: (2^10)-1 = 1023
            #Si se utiliza el método read() en un pin que está configurado como salida, el resultado puede ser
            #impredecible o no tener sentido.
            tensionBinariaA0 = self.analog_0.read()                                 #Lee el pin analógico A0.
            tensionBinariaA1 = self.analog_1.read()                                 #Lee el pin analógico A1.

            #CONVERSIÓN DE NUMEROS BINARIOS NUMÉRICOS DE TENSIÓN A VALORES DE TENSIÓN REALES:
            #float(): Método que convierte un tipo de dato cualquiera en númerico decimal.
            #Se realiza esta operación porque como el ADC del arduino lee de 0 a 5V y como tiene una resolución
            #de 10 bits permitiendo que en el ADC los valores de tensión se interpreten como valores numéricos
            #enteros que valen de 0 a (2^10)-1 = 1023, se hace una regla de 3 para que se imprima el valor de la
            #tensión en consola en vez del valor decimal binario. En esta operación no es necesario dividir el
            #resultado entre la resolución del ADC porque el valor que retorna el método read ya está
            #normalizado en el rango de 0 a 1 (correspondiente a los 1024 niveles).
            #Tensión = Tensión_decimal_read*(ValorMáximoTensión) = Tensión_decimal_read*(5)
            VoltsA0 = (float(tensionBinariaA0)*(self.high_value_board))             #Tensión real pin A0.
            VoltsA1 = (float(tensionBinariaA1)*(self.high_value_board))             #Tensión real pin A1.
            umbral = self.umbralLed/1000.0                                          #Tensión de umbral [V].

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

            #IMPRIMIR EN CONSOLA TIEMPO Y TENSIÓN:
            msg_console = str(self.count) + ".- Time: " + str(self.time_val) + " [s]" + "\t" #Tiempo [s].
            msg_console+= "Voltage Pin A0: " + "{0:.5f}".format(VoltsA0)+" [V]\t"          #Tensión real pin A0.
```

```
msg_console+= "Voltage Pin A1: " + "{0:.5f}".format(VoltsA1)+" [V]\t"            #Tensión real pin A1.
msg_console+= "Umbral: " + "{0:.5f}".format(umbral)+" [V]"                  #Tensión de umbral [V].
print(msg_console)

#GUARDAR LOS DATOS RECOPILADOS EN UNA MATRIZ PARA POSTERIORMENTE ALMACENARLOS EN UN ARCHIVO:
#append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
#Lista que guarda los valores de tiempo [s] y tensión [V] recopilados.
self.values.append(str(self.time_val) + ", " + "{0:05f}".format(VoltsA0) + ", " +
                                               "{0:05f}".format(VoltsA1) + ", " +
                                               "{0:05f}".format(umbral))
#matplotlib.Figure.add_subplot().cla(): Método cuyo nombre es una abreviatura de "clear axes" y
#sirve para restablecer el estado de los ejes pertenecientes a una gráfica creada con la librería
#matplotlib.
self.canvas.axes.cla()
self.x = np.append(self.x, self.time_val)           #Vector tiempo [segundos].
self.y0 = np.append(self.y0, VoltsA0)               #Vector tensión A0 [V].
self.y1 = np.append(self.y1, VoltsA1)               #Vector tensión A1 [V].
self.u = np.append(self.u, umbral)                  #Vector tensión umbral [mV].



#matplotlib.Figure().add_subplot().set_xlabel(): Método para indicar el texto que aparece en el eje
#horizontal de la gráfica, recibe los siguientes parámetros:
# - xlabel: Especifica el texto que se mostrará en el eje x.
# - fontname: Indica el estilo de la fuente:
#       - Nombres de tipos de letra estándar: "Arial", "Times New Roman", "Helvetica", "Courier",
#         "Monospaced", "Consolas", etc. Estos nombres deben ser compatibles con los tipos de letra
#         instalados en el sistema operativo.
#       - Nombres de tipos de letra genéricos: "serif", "sans-serif", "monospace", etc.
#       - Rutas de archivo: Si se tiene un archivo de tipo de letra personalizado, se puede
#         especificar la ruta del archivo como el valor de fontname.
# - fontsize: Indica el tamaño de la fuente.
# - labelpad: Especifica el espaciado entre la etiqueta del eje x y el eje en sí
# - color: Indica el color del texto colocado en el eje x, para ello es válido usar colores:
#       - Básicos CSS: como "red", "blue", "green", etc.
#       - Colores hexadecimales: "#FF0000", "#00FF00", etc.
#       - Colores RGB: "rgb(255,255,255)" para el color blanco y "rgb(0,0,0)" para el negro.
self.canvas.axes.set_xlabel(xlabel = "Time [s]", fontname = "Consolas", fontsize = 8, color = "white")
#matplotlib.Figure.add_subplot().set_ylabel(): Método para indicar el texto que aparece en el eje y.
self.canvas.axes.set_ylabel(ylabel = "Voltage [V]", fontname = "Consolas", fontsize = 8, color = "white")
#matplotlib.Figure.add_subplot().plot(): Método usado para gráficar, indicando como primer parámetro
#su eje horizontal, luego su eje vertical y finalmente el estilo de la gráfica:
# - Colores:             C1: color naranja, r: color rojo, b: color azul, g: verde, c: cyan,
#   m: morado, y: amarillo, k: negro, w: blanco.
# - Tipo de marcadores:  o: círculos, +: símbolos de más, .; puntos, v: Triángulo hacia abajo,
#   h: Hexágono, etc.
```

```
# - Tipo de Líneas:      -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto,
#   'or': Nada.
#Los marcadores se obtienen del siguiente link: https://matplotlib.org/stable/api/markers_api.html
self.canvas.axes.plot(self.x, self.y0, 'y1:') #'y:' c: color amarillo, 1: tri_down, :: línea punteada.
self.canvas.axes.plot(self.x, self.y1, 'r2--') #'r.--' c: color rojo, 2: tri_up, --: linea punteada.
self.canvas.axes.plot(self.x, self.u, 'c,-') #'w,-' c: color cyan, .: pixel, -: línea solida.
#matplotlib.Figure().add_subplot().legend(): La leyenda de un gráfico es un componente que
#proporciona información individualmente sobre los diferentes elementos o series presentes en el
#gráfico, el método permite personalizar dicha leyenda a través de los siguientes parámetros:
# - labels: Se refiere a los nombres que se les puede poner para las leyendas de los distintos
#   elementos del gráfico.
#           - Las etiquetas se generan automáticamente pero se pueden colocar manualmente de la
#             siguiente manera:
#               - labels=['Etiqueta 1', 'Etiqueta 2']
# - loc: Indica la ubicación de la leyenda en el gráfico. Se puede indicar con una cadena de texto o
#   un código numérico que represente una posición específica.
#           - "best": Coloca la leyenda en la mejor ubicación posible, evitando superponerse con
#             otros elementos.
#           - "upper right": Coloca la leyenda en la esquina superior derecha del gráfico.
#           - "upper left": Coloca la leyenda en la esquina superior izquierda del gráfico.
#           - "lower right": Coloca la leyenda en la esquina inferior derecha del gráfico.
#           - "lower left": Coloca la leyenda en la esquina inferior izquierda del gráfico.
#           - "right": Coloca la leyenda en el lado derecho del gráfico, centrada verticalmente.
#           - "center left": Coloca la leyenda en el lado izquierdo del gráfico, centrada
#             verticalmente.
#           - "center right": Coloca la leyenda en el lado derecho del gráfico, centrada
#             verticalmente.
#           - "center": Coloca la leyenda en el centro del gráfico.
self.canvas.axes.legend(labels=['Pin A0', 'Pin A1', 'Umbral Leds 13/12'], loc = "best")
#matplotlib.FigureCanvas().draw(): Método que actualiza y muestra los datos recopilados en tiempo
#real en la gráfica creada con el objeto que instancía la clase FigureCanvasQTAgg.
self.canvas.draw()

#Si la tensión que ingresa en el pin A0 es mayor al umbral, se enciende un led con el pin 13.
if(VoltsA0 > umbral):
    #pyfirmata.Arduino.get_pin().write(): Método que se utiliza para escribir un valor en algún pin
    #digital de la placa, controlando si su estado está en alto (1 o True) o bajo (0 o False).
    # - Encender LED:   write(True)  o write(1).
    # - Apagar LED:     write(False) o write(0).
    #El método write() solamente se puede usar con pines de salida digital, si se quiere controlar
    #la salida de un pin analógico se debe usa el método analog_write().
    self.digital_13.write(1)                        #Led encendido en el pin 13.
    print("Led en el pin 13 Encendido")
else:
    self.digital_13.write(0)                        #Led apagado en el pin 13.
```

```
            #Si la tensión que ingresa en el pin A1 es mayor al umbral, se enciende un led con el pin 12.
            if(VoltsA1 > umbral):
                self.digital_12.write(True)                     #Led encendido en el pin 12.
                print("Led en el pin 12 Encendido")
            else:
                self.digital_12.write(False)                    #Led apagado en el pin 12.
        except:
            print("No se pudo actualizar la gráfica")

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

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

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

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


    #función SerialPorts(): Método creado dentro de la clase propia llamada MainWindow que sirve para rellenar
    #los elementos del Combo Box que muestran todos los puertos disponibles en el ordenador a donde se podría
    #conectar la placa de desarrollo Arduino, de estos puertos se debe seleccionar el que haya sido elegido como
    #puerto de conexión dentro del IDE de Arduino.
    #def nombre_función -> tipo_de_dato: Es una sintaxis llamada anotación que se utiliza para indicar el tipo
    #de dato que devuelve una función. Es importante tener en cuenta que las anotaciones de tipo en Python son
    #opcionales y no afectan directamente el comportamiento o la ejecución de la función. Son principalmente
```

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

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

        #Bucle for para intentar abrir todos los puertos enlistados y añadirlos a la lista result:
        for port in ports:
```

```
#MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
# - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción
#   durante su ejecución, el programa brinca al código del except
# - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción
#   cuando ocurra el error.
#Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que
#pueda ocurrir un error durante su ejecución.
try:
    #Instancia de la librería serial por medio del constructor de la clase Serial para establecer
    #una comunicación serial por medio de puertos seriales o USB con dispositivos externos como
    #microcontroladores, módems, teclados, impresoras, etc. Los parámetros que puede recibir el
    #constructor de la clase Serial son:
    # - port: Especifica el nombre en formato string del puerto serial al que se desea conectar.
    #           - Por ejemplo: "COM1" para sistemas operativos Windows o "/dev/ttyUSB1" para
    #             sistemas operativos Unix/Linux o iOS.
    # - baudrate: Define la velocidad de transmisión en baudios (bit trasmitido por segundo) para la
    #   comunicación serial.
    #           - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es
    #             compatible con la mayoría de los dispositivos y programas.
    #           - Sin embargo, si se necesita una transferencia de datos más rápida y el
    #             hardware/software lo admiten, se puede optar por velocidades más altas como 115200
    #             o 57600 baudios.
    # - bytesize: Especifica el tamaño de los bytes en la comunicación serial. Puede adoptar uno de
    #   los siguientes valores:
    #           - serial.FIVEBITS: Tamaño de 5 bits en los paquetes de la transmisión serial.
    #           - serial.SIXBITS: Tamaño de 6 bits en los paquetes de la transmisión serial.
    #           - serial.SEVENBITS: Tamaño de 7 bits en los paquetes de la transmisión serial.
    #           - serial.EIGHTBITS: Tamaño de 8 bits en los paquetes de la transmisión serial.
    # - parity: Indica el tipo de paridad utilizado en la comunicación serial. La paridad es un
    #   mecanismo utilizado en las comunicaciones seriales para verificar la integridad de los datos
    #   transmitidos, se basa en la adición de un bit adicional (bit de paridad) en el bit más
    #   significativo (hasta la izquierda) de cada paquete de datos transmitido. Al seleccionar la
    #   paridad, nos debemos asegurar de que tanto el dispositivo emisor como el receptor estén
    #   configurados con la misma paridad para efecutar una comunicación adecuada:
    #           - serial.PARITY_NONE: No se utiliza ningún bit de paridad. Esto implica que no se
    #             verifica la integridad de los datos mediante la paridad.
    #           - serial.PARITY_EVEN: Se utiliza la paridad par. Para ello se cuentan el número de
    #             bits en el byte, incluido el bit de paridad:
    #               - Si el número total de bits es impar, se establece el bit de paridad en 1 para
    #                 que el número total de bits sea par.
    #               - Si el número total de bits es par, se deja el bit de paridad en 0.
    #                   - Por ejemplo, supongamos que se desea transmitir el byte 11010110. El bit
    #                     de paridad en la transmisión de la comunicación se calcularía contando el
    #                     número total de bits, que es 8, el número total de bits es par, por lo que
    #                     el bit de paridad se establece en 0, Por lo tanto, el byte transmitido
```

```
            #                   sería 011010110, donde el bit más significativo es el bit de paridad.
            #                   Luego en el extremo receptor de la comunicación, se realizará un cálculo
            #                   similar para verificar la integridad de los datos. Si el número total de
            #                   bits, incluido el bit de paridad, no coincide con la paridad esperada (en
            #                   este caso, par), se puede detectar un error en la transmisión de datos.
            #           - serial.PARITY_ODD: Se utiliza paridad impar. El bit de paridad se establece de
            #             manera que el número total de bits en el byte transmitido (incluido el bit de
            #             paridad) sea impar.
            #           - serial.PARITY_MARK: Se utiliza paridad de marca. El bit de paridad se establece en
            #             1 (marcado) para todos los bytes transmitidos.
            #           - serial.PARITY_SPACE: Se utiliza paridad de espacio. El bit de paridad se establece
            #             en 0 (espacio) para todos los bytes transmitidos.
            # - stopbits: Define el número de bits de parada en la comunicación serial. El número de bits de
            #   parada se utiliza para indicar el final de cada byte transmitido en la comunicación serial.
            #   La elección del número de bits de parada depende de la configuración del dispositivo externo
            #   con el que se está comunicando. El parámetro uno de los siguientes valores:
            #           - serial.STOPBITS_ONE: Indica que se utiliza un bit de parada.
            #           - serial.STOPBITS_ONE_POINT_FIVE: Indica que se utiliza un bit y medio de parada.
            #             Este valor puede ser utilizado en algunas configuraciones especiales.
            #           - serial.STOPBITS_TWO: Indica que se utilizan dos bits de parada.
            # - timeout: Especifica el tiempo de espera en segundos para las operaciones de lectura. Si no
            #   se recibe ningún dato dentro de este tiempo, la operación de lectura se interrumpe.
            # - xonxoff: Con True o False indica si se utiliza el control de flujo XON/XOFF para la
            #   comunicación serial.
            # - rtscts: Con True o False indica si se utiliza el control de flujo RTS/CTS para la
            #   comunicación serial.
            # - dsrdtr: Con True o False indica si se utiliza el control de flujo DSR/DTR para la
            #   comunicación serial.
            # - write_timeout: Especifica el tiempo de espera en segundos para las operaciones de escritura.
            #   Si no se puede escribir ningún dato dentro de este tiempo, la operación de escritura se
            #   interrumpe.
            # - inter_byte_timeout: Define el tiempo de espera en segundos entre la recepción de bytes
            #   consecutivos durante las operaciones de lectura.
            #str(): Método que convierte un tipo de dato cualquiera en string.
            s = serial.Serial(port)     #Inicio de comunicación serial.
            #serial.Serial.close(): Método que cierra la comunicación serial. Es muy importante mencionar
            #que si no se ejecuta este método, el puerto serial se va a quedar bloqueado y no se podrá usar.
            s.close()                   #Terminación de la comunicación serial.
            #append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
            result.append(port)
        #Para identificar el tipo de excepción que ha ocurrido y utilizarlo en la instrucción except, se
        #puede utilizar la clase Exception, que es una clase incorporada en Python utilizada para describir
        #todos los tipos de excepciones, luego de colocar el nombre de la clase Exception se usa la palabra
        #reservada "as" seguida de un nombre de variable, esto nos permitirá acceder a la instancia de la
        #excepción y utilizarla dentro del except.
```

```
            except Exception as error:
                #type(clase).__name__: Esta instrucción no es un método, sino una expresión que se utiliza para
                #obtener el nombre de la clase de un objeto en Python, donde type(error) devuelve el tipo de
                #excepción en este caso ya que error es un objeto de una clase de excepción.
                # - __name__: Es un atributo especial en Python que se utiliza para obtener el nombre de la
                #   clase del objeto.
                print("Ocurrió el siguiente tipo de error al intentar conectarse a todos los puertos disponibles: ",
type(error).__name__)
                print("Este es el mensaje del error: ", error)
                #Aunque ocurra un error al tratar de encontrar todos los tipos de puertos, esto no significa
                #que el programa no vaya a funcionar, solo significa que no se ha podido conectar con todos los
                #puertos seriales que encontró en la computadora, muy seguramente porque puede que estos estén
                #siendo ya usados en otra cosa.

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


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


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


    #Método OnStartSaving
    def umbral_changed(self, val_umbral):
        #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
        self.umbralLed = val_umbral
```

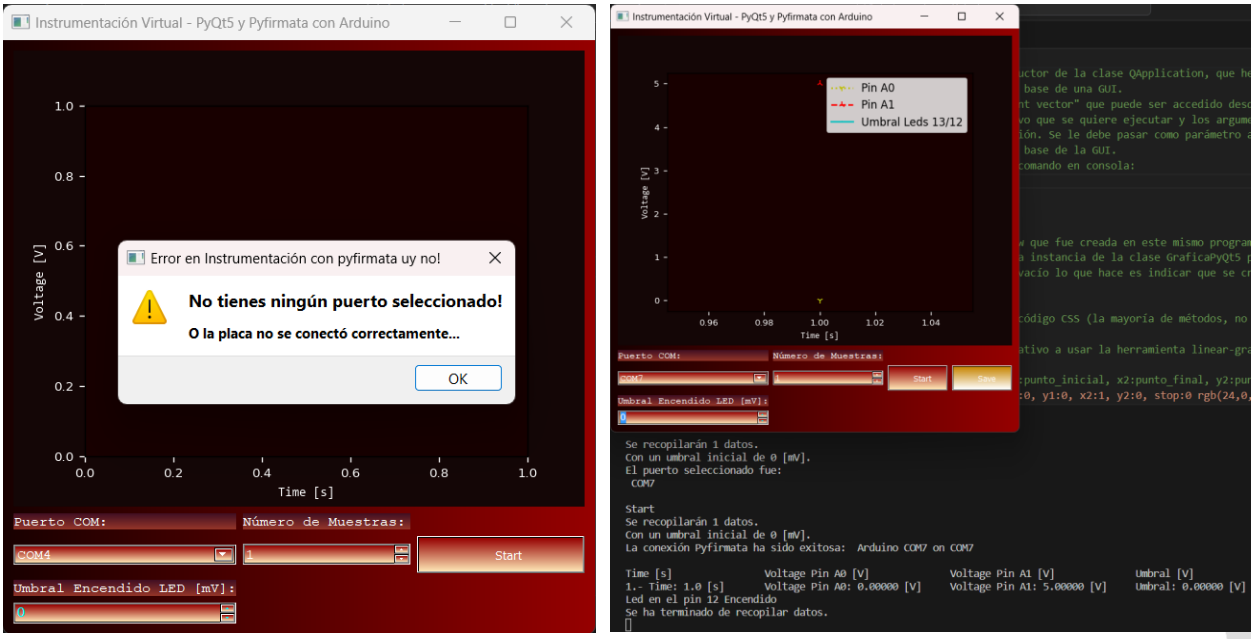
```
        print("Se debe sobrepasar el umbral de ", self.umbralLed, " [mV] = ",
              self.umbralLed/1000.0,  " [V] para que encienda el led.")


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


    #función OnStartSaving(): Método creado dentro de la clase propia llamada MainWindow que recibe como
    #parámetro el evento que lo activa, para posteriormente ejecutar cierta acción.
    #En este caso el evento es activado por dar un clic sobre el botón de Save y lo que hace es abrir el
    #explorador de archivos para nombrar el archivo Excel que guardará los datos recabados de tensión y tiempo,
    #estos datos los tomará del vector self.values, creado en la función update_plot().
    def OnStartSaving(self):
        print("Save Data")
        #Instancia de la librería PyQt5 por medio del constructor de la clase Options, que hereda de las clases
        #QFileDialog y QtWidgets para ejecutar un método que cree un cuadro de diálogo, el objeto Options se
        #puede modificar para especificar las opciones de comportamiento del diálogo de archivo, pero para ello
        #primero se debe crear un objeto y luego usar la compuerta lógica OR (|), ya que el constructor de la
        #clase Options() no recibe parámetros.
        options = QtWidgets.QFileDialog.Options()
        #Las opciones de configuración disponibles para el objeto Options() son:
        # - DontUseNativeDialog: Indica que no se debe utilizar el explorador de archivos nativo del sistema
        #   operativo y se debe utilizar el diálogo proporcionado por PyQt, esto significa que tendrá el mismo
        #   estilo que se indique a la ventana.
        # - ReadOnly: Abre el diálogo en modo de solo lectura, lo que impide al usuario guardar o modificar
        #   archivos existentes.
        # - HideNameFilterDetails: Oculta los detalles del filtro de nombre en el diálogo.
        # - DontResolveSymlinks: No resuelve los enlaces simbólicos al mostrar el diálogo de archivo.
        # - DontConfirmOverwrite: No muestra un mensaje de confirmación al guardar o sobrescribir un archivo
        #   existente.
        # - DontUseSheet: No utiliza una hoja de diálogo (específico de macOS).
        # - DontUseCustomDirectoryIcons: No utiliza iconos personalizados para los directorios en el diálogo.
        # - DontUseNativeFileSizeDisplay: No utiliza la representación nativa del tamaño de archivo en el
        #   diálogo.
        # - DontUseCustomDirectoryIcons: No utiliza iconos personalizados para los directorios en el diálogo.
        options |= QtWidgets.QFileDialog.DontUseNativeDialog
```

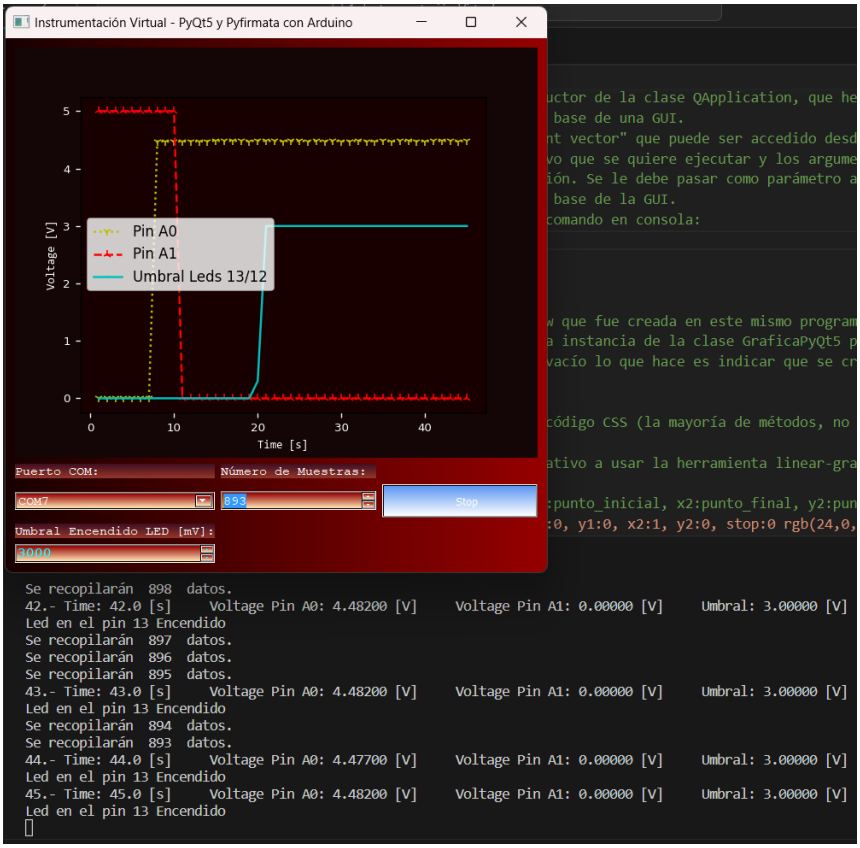
```
#PyQt5.QtWidgets.QFileDialog.getSaveFileName(): El objeto QtWidgets.QFileDialog proporciona métodos para
#mostrar una ventana de selección de archivos. Uno de estos métodos es getSaveFileName(), que se utiliza
#para mostrar un diálogo de archivo para guardar un archivo y recibe los siguientes parámetros:
# - parent: Es el objeto que se utiliza como referencia para mostrar el diálogo de archivo. Puede ser
#   una ventana o un widget. Si se proporciona, la ventana se mostrará en la parte superior del objeto.
# - caption: Es el título que se muestra en la parte superior de la ventana del explorador de archivos.
# - directory: Es el directorio inicial que se muestra cuando se abre el diálogo de archivo. Se puede
#   especificar una ubicación específica  o si no se proporciona solo se mostrará el directorio actual.
# - filter: Son las opciones de filtro para los tipos de archivos que se mostrarán en la ventana. Se
#   pueden especificar diferentes tipos de archivos separados por puntos y coma (;). Por ejemplo, se
#   puede indicar un filtro para mostrar solo archivos CSV, PDF o todos los archivos.
# - initialFilter: Es el filtro que se seleccionará inicialmente cuando se abra el explorador de
#   archivos. Si se tiene varios filtros y solo uno se desea que sea seleccionado por defecto, se puede
#   especificar en este parámetro.
# - options: Son opciones adicionales para personalizar el comportamiento del explorador de archivos. Se
#   puede usar para ocultar ciertos elementos, permitir la selección de múltiples archivos o mostrar
#   miniaturas, modificar el aspecto estético de la ventaba, etc. Para ello se debe asignar el valor
#   options al parámetro options y en el método main, usar métodos que afecten a todas las ventanas.
#   Además se pueden combinar diferentes opciones utilizando operadores especiales.
#El método .getSaveFileName() devuelve una tupla de dos valores:
# - filename: Es un string que representa el nombre del archivo seleccionado por el usuario. Si el
#   usuario no selecciona ningún archivo o cancela el diálogo, este valor será una cadena vacía.
# - filter: Es un string que representa el filtro seleccionado por el usuario en el diálogo de archivo.
#   El filtro corresponde al tipo de archivo seleccionado en el diálogo. Por ejemplo, si el usuario
#   selecciona el filtro "csv Files (.csv)", este valor será la cadena "csv Files (.csv)". Si el usuario
#   cancela el diálogo, este valor también será una cadena vacía.
#En Python, si queremos almacenar ambos valores en dos variables distintas se usa la siguiente sintaxis:
#       variable1, variable2 = QtWidgets.QFileDialog.getSaveFileName()
#Pero si alguno de estos valores no nos interesa que se almacene en una variable, simplemente se coloca
#un guión bajo para indicar eso, en este ejemplo el segundo valor de la tupla no se quiere usar:
#       variable1, _ = QtWidgets.QFileDialog.getSaveFileName()
nombreArchivo, _= QtWidgets.QFileDialog.getSaveFileName(parent = self,
                                        caption = "Almacena los datos recopilados del Arduino",
                                        directory = "",
                                        filter = "csv Files(*.csv);;All Files (*)",
                                        options = options)
#En Python si solo se coloca un if con el nombre de una variable, la condición que se está evaluando
#dentro de su paréntesis es que esa variable sea distinta de Null, si es así, se ejecuta la acción que
#está dentro del condicional.
#En este caso se está evaluando que ya se haya seleccionado un nombre de archivo para que se almacenen
#los datos recabados del Arduino en él.
if(nombreArchivo):
    #open(): Método que sirve para abrir un archivo cualquiera, para ello es necesario indicar dos
    #parámetros, el primero se refiere a la ruta relativa o absoluta del archivo previamente creado y la
    #segunda indica qué es lo que se va a realizar con él, el contenido del archivo se asigna a una
```

```
            #variable.
            # - w: Sirve para escribir en un archivo, pero borrará la información que previamente contenía el
            #   archivo.
            # - a: Sirve para escribir en un archivo sin que se borre la info anterior del archivo, se llama
            #   append.
            file = open(nombreArchivo, 'w')
            #myFile.write(): Método para colocar un string en un archivo previamente abrierto con el método
            #open(), en este caso se utiliza para colocar el valor de las dos columnas en el Excel, donde se
            #acomodan verticalmente los valores de tiempo y tensión recopilados.
            file.write("Library, PyQt5, Pyfirmata" + "\n")
            file.write("Time [s], Voltage Pin A0 [V], Voltage Pin A1 [V], Umbral [V]" + "\n")
            #Del vector values se obtienen los valores de tiempo y tensión recabados y agrupados.
            for i in range(len(self.values)):
                file.write(self.values[i] + "\n")
            #file.close(): Método para cerrar un archivo previamente abrierto con el método open(), es peligroso
            #olvidar colocar este método, ya que la computadora lo considerará como si nunca hubiera sido
            #cerrado, por lo cual no podré volver a abrirlo al dar clic sobre él.
            file.close()



#__name__ == __main__: Método main, esta función es super importante ya que sirve para instanciar las clases del
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.
if (__name__ == "__main__"):
    #Instancia de la librería PyQt5 por medio del constructor de la clase QApplication, que hereda de la clase
    #QtWidgets para crear un objeto que funcione como la base de una GUI.
    # - sys.argv: Se refiere a un vector llamado "argument vector" que puede ser accedido desde la librería sys,
    #   este incluye en su contenido el nombre del archivo que se quiere ejecutar y los argumentos necesarios
    #   que se le deben pasar para que efectúe su ejecución. Se le debe pasar como parámetro al constructor de
    #   la clase QApplication para que se pueda crear la base de la GUI.
    #       - Por ejemplo, si se ejecutara el siguiente comando en consola:
    #           python mi_script.py arg1 arg2 arg3
    #         El contenido de sys.argv sería:
    #           ['mi_script.py', 'arg1', 'arg2', 'arg3']
    app = QtWidgets.QApplication(sys.argv)
    #Instancia de nuestra clase propia llamada MainWindow que fue creada en este mismo programa (window se
    #refiere a la ventana del GUI en PyQt5) e incluye una instancia de la clase GraficaPyQt5 para agregar un
    #widget que crea una gráfica dentro, el constructor vacío lo que hace es indicar que se cree y muestre la
    #ventana.
    window = MainWindow()
    #widget.setStyleSheet(): Método que permite aplicar código CSS (la mayoría de métodos, no todos) a los
    #widgets de una interfaz gráfica de usuario (GUI).
    # - La siguiente línea de código es un método alternativo a usar la herramienta linear-gradient, ya que
    #   esta no es admitida por PyQt5:
```

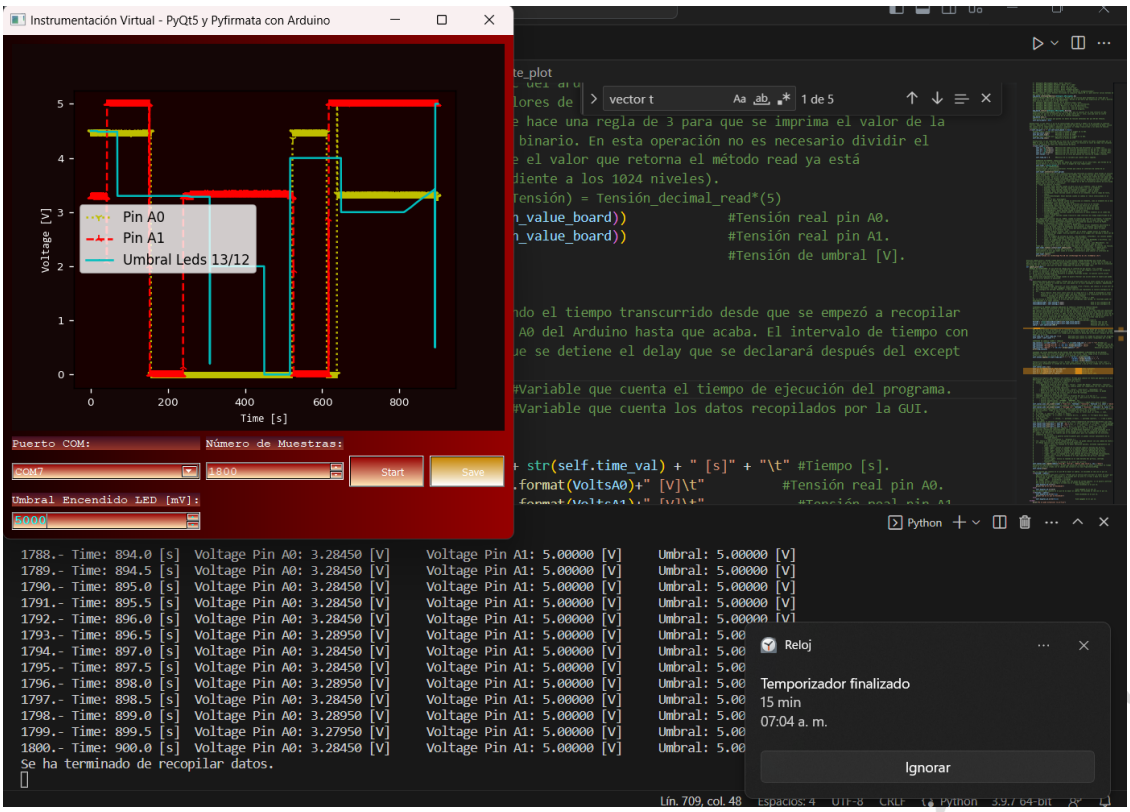
```
    #   background: qlineargradient(x1:punto_inicial, y1:punto_inicial, x2:punto_final, y2:punto_final, stop:0
rgb(R_inicial,G_inicial,B_inicial), stop:1 rgb(R_final,G_final,B_final));
    window.setStyleSheet("background: qlineargradient(x1:0, y1:0, x2:1, y2:0, stop:0 rgb(24,0,0), stop:1 rgb(150,0,0)); color:
white;")
    #PyQt5.QtWidgets.QMainWindow.move() = window.move(): Método utilizado para indicar la posición inicial de
    #la ventana dentro de la pantalla del ordenaror, este método recibe como parámetro una tupla que indica la
    #posición:
    # - (x, y): Con este atributo se indica la posición inicial del Frame en pixeles, siendo la posición 0,0 la
    #   esquina superior izquierda, donde las "y" positivas indican que se mueva el botón hacia abajo y las "x"
    #   positivas hacia la derecha.
    window.move(100, 100)
    #PyQt5.QtWidgets.QMainWindow.show() = window.show(): Método aplicado al objeto de la clase QMainWindow,
    #del que hereda esta clase propia para mostrar la ventana del GUI.
    window.show()
    #PyQt5.QtWidgets.QApplication.exec_(): Método para que se ejecute en un loop infinito el GUI, logrando así
    #que no se ejecute una vez y luego cierre por sí solo, sino que solo se cierre solamente al dar clic en el
    #tache del window.
    app.exec_()
```

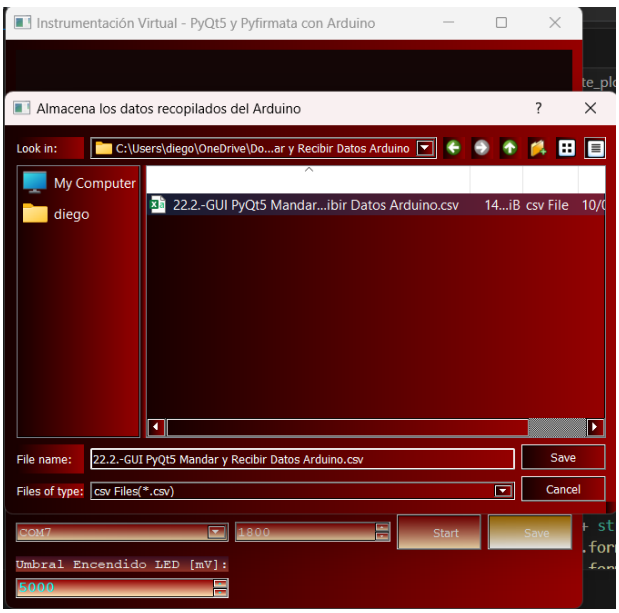
Resultado del Código Python: Interfaz Gráfica PyQt5

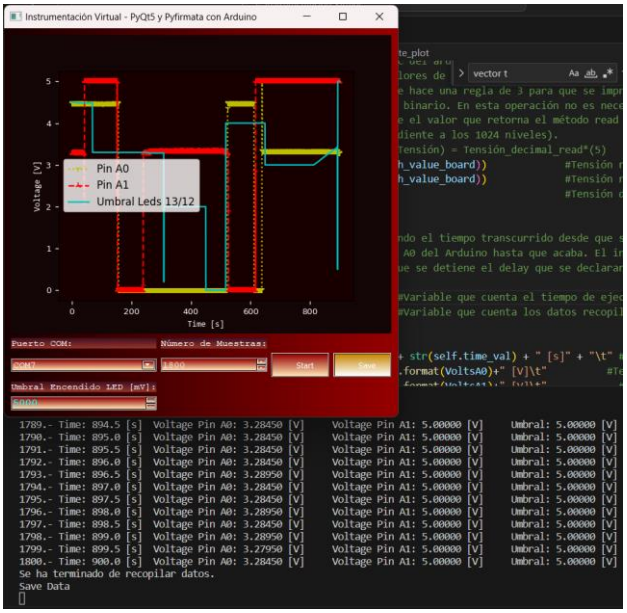


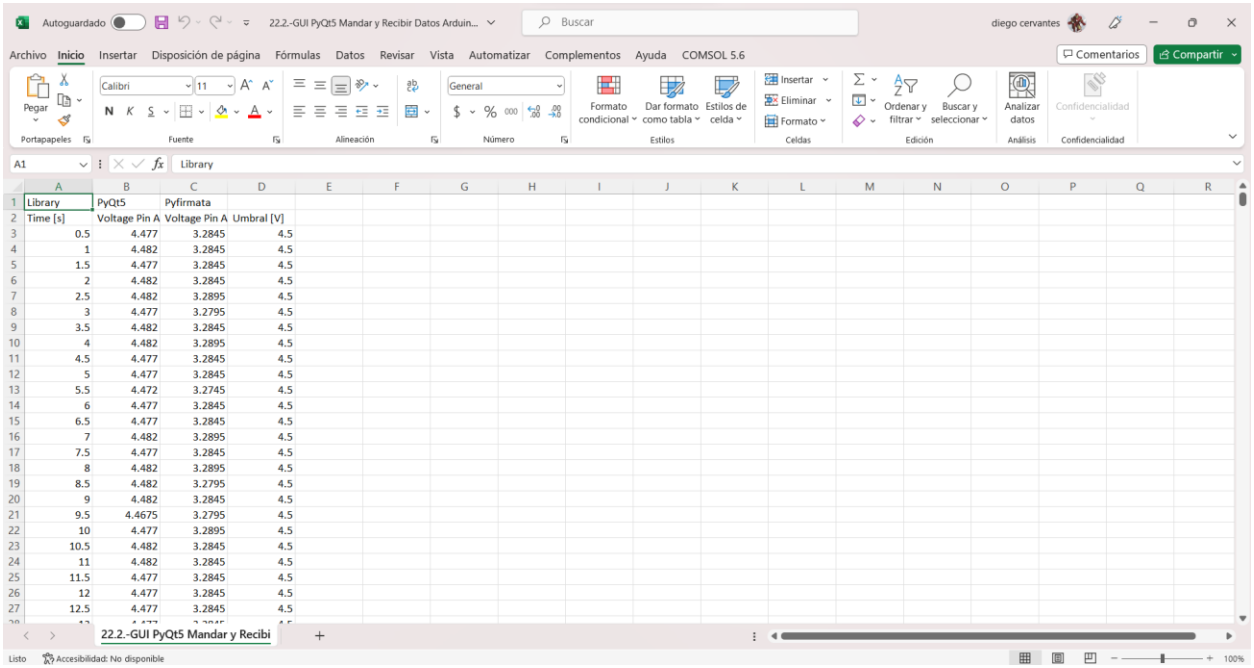


El tiempo del GUI con **PyQt5** cuenta cada 0.5 segundos también.

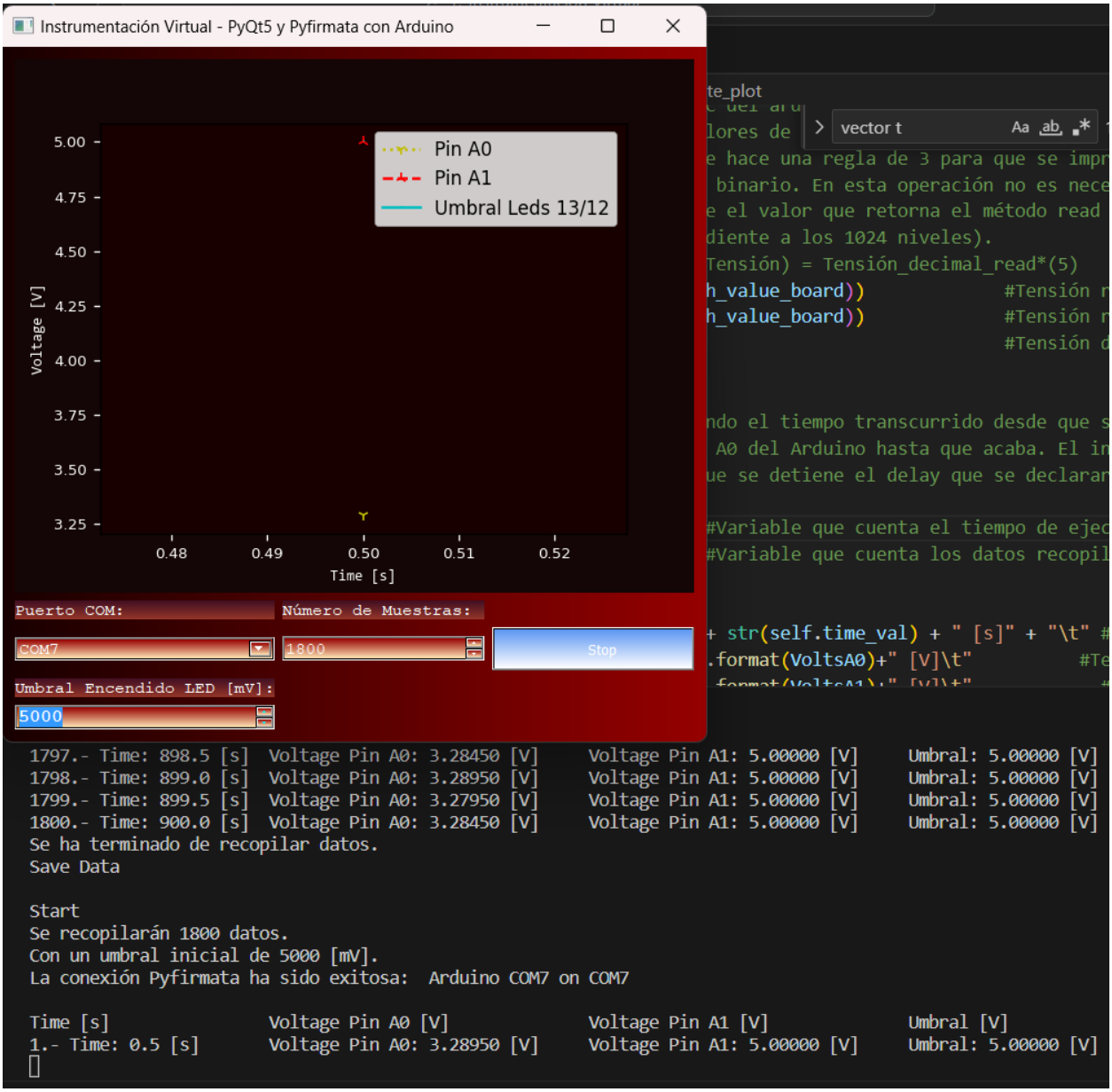



Instrumentación Virtual - PyQt5 y Pyfirmata con Arduino
Almacena los datos recopilados del Arduino
Look in:
C:\Users\diego\OneDrive\Do...ar y Recibir Datos Arduino
My Computer
diego
22.2.-GUI PyQt5 Mandar...ibir Datos Arduino.csv
File name:
22.2.-GUI PyQt5 Mandar y Recibir Datos Arduino.csv
Save
Files of type:
csv Files(*.csv)
Cancel
COM7
1800
Start
Save
Umbral Encendido LED [mV]:
6000


Instrumentación Virtual - PyQt5 y Pyfirmata con Arduino
Pin A0
Pin A1
Umbral Leds 13/12
Voltage [V]
Time [s]
Puerto COM:
Número de Muestras:
COM7
1800
Start
Save
Umbral Encendido LED [mV]:
6000
1789.- Time: 894.5 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1790.- Time: 895.0 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1791.- Time: 895.5 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1792.- Time: 896.0 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1793.- Time: 896.5 [s] Voltage Pin A0: 3.28950 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1794.- Time: 897.0 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1795.- Time: 897.5 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1796.- Time: 898.0 [s] Voltage Pin A0: 3.28950 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1797.- Time: 898.5 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1798.- Time: 899.0 [s] Voltage Pin A0: 3.28950 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1799.- Time: 899.5 [s] Voltage Pin A0: 3.27950 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1800.- Time: 900.0 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
Se ha terminado de recopilar datos.
Save Data


Autoguardado
22.2.-GUI PyQt5 Mandar y Recibir Datos Arduin...
Buscar
diego cervantes
Archivo Inicio Insertar Disposición de página Fórmulas Datos Revisar Vista Automatizar Complementos Ayuda COMSOL 5.6
Comentarios
Compartir
Pegar
Calibri
11
General
Formato condicional
Dar formato como tabla
Estilos de celda
Insertar
Eliminar
Formato
Ordenar y filtrar
Buscar y seleccionar
Analizar datos
Confidencialidad
Portapapeles
Fuente
Alineación
Número
Estilos
Celdas
Edición
Análisis
Confidencialidad
A1
Library
Library PyQt5 Pyfirmata
Time [s] Voltage Pin A Voltage Pin A Umbral [V]
0.5 4.477 3.2845 4.5
1 4.482 3.2845 4.5
1.5 4.477 3.2845 4.5
2 4.482 3.2845 4.5
2.5 4.482 3.2895 4.5
3 4.477 3.2795 4.5
3.5 4.482 3.2845 4.5
4 4.482 3.2895 4.5
4.5 4.477 3.2845 4.5
5 4.477 3.2845 4.5
5.5 4.472 3.2745 4.5
6 4.477 3.2845 4.5
6.5 4.477 3.2845 4.5
7 4.482 3.2895 4.5
7.5 4.477 3.2845 4.5
8 4.482 3.2895 4.5
8.5 4.482 3.2795 4.5
9 4.482 3.2845 4.5
9.5 4.4675 3.2795 4.5
10 4.477 3.2895 4.5
10.5 4.482 3.2845 4.5
11 4.482 3.2845 4.5
11.5 4.477 3.2845 4.5
12 4.477 3.2845 4.5
12.5 4.477 3.2845 4.5
22.2.-GUI PyQt5 Mandar y Recibi
Listo
Accesibilidad: No disponible
100%


1780 889 3.3335 5 3.409
1781 889.5 3.3335 5 3.412
1782 890 3.3285 5 3.414
1783 890.5 3.3285 5 3.417
1784 891 3.3335 5 3.42
1785 891.5 3.3285 5 3.422
1786 892 3.3335 5 3.422
1787 892.5 3.3285 5 0.5
1788 893 3.3285 5 5
1789 893.5 3.2845 5 5
1790 894 3.2845 5 5
1791 894.5 3.2845 5 5
1792 895 3.2845 5 5
1793 895.5 3.2845 5 5
1794 896 3.2845 5 5
1795 896.5 3.2895 5 5
1796 897 3.2845 5 5
1797 897.5 3.2845 5 5
1798 898 3.2895 5 5
1799 898.5 3.2845 5 5
1800 899 3.2895 5 5
1801 899.5 3.2795 5 5
1802 900 3.2845 5 5
22.2.-GUI PyQt5 Mandar y Recibi
Listo
Accesibilidad: No disponible
Promedio: 114.7076238
Recuento: 7204
Suma: 825894.8915
100%


Instrumentación Virtual - PyQt5 y Pyfirmata con Arduino
Pin A0
Pin A1
Umbral Leds 13/12
Voltage [V]
Time [s]
Puerto COM:
COM7
Número de Muestras:
1800
Stop
Umbral Encendido LED [mV]:
5000
te_plot
vector t
1797.- Time: 898.5 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1798.- Time: 899.0 [s] Voltage Pin A0: 3.28950 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1799.- Time: 899.5 [s] Voltage Pin A0: 3.27950 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
1800.- Time: 900.0 [s] Voltage Pin A0: 3.28450 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]
Se ha terminado de recopilar datos.
Save Data
Start
Se recopilarán 1800 datos.
Con un umbral inicial de 5000 [mV].
La conexión Pyfirmata ha sido exitosa: Arduino COM7 on COM7
Time [s] Voltage Pin A0 [V] Voltage Pin A1 [V] Umbral [V]
1.- Time: 0.5 [s] Voltage Pin A0: 3.28950 [V] Voltage Pin A1: 5.00000 [V] Umbral: 5.00000 [V]