UNIVERSIDAD AUTONOMA DE TAMAULIPAS

FACULTAD DE INGENIERÍA

“ARTURO NARRO SILLER”

INGENIERÍA EN SISTEMAS COMPUTACIONALES

PROGRAMACION DE INTERFACES Y PUERTOS 6 I

Portafolio de Evidencias Equipo:

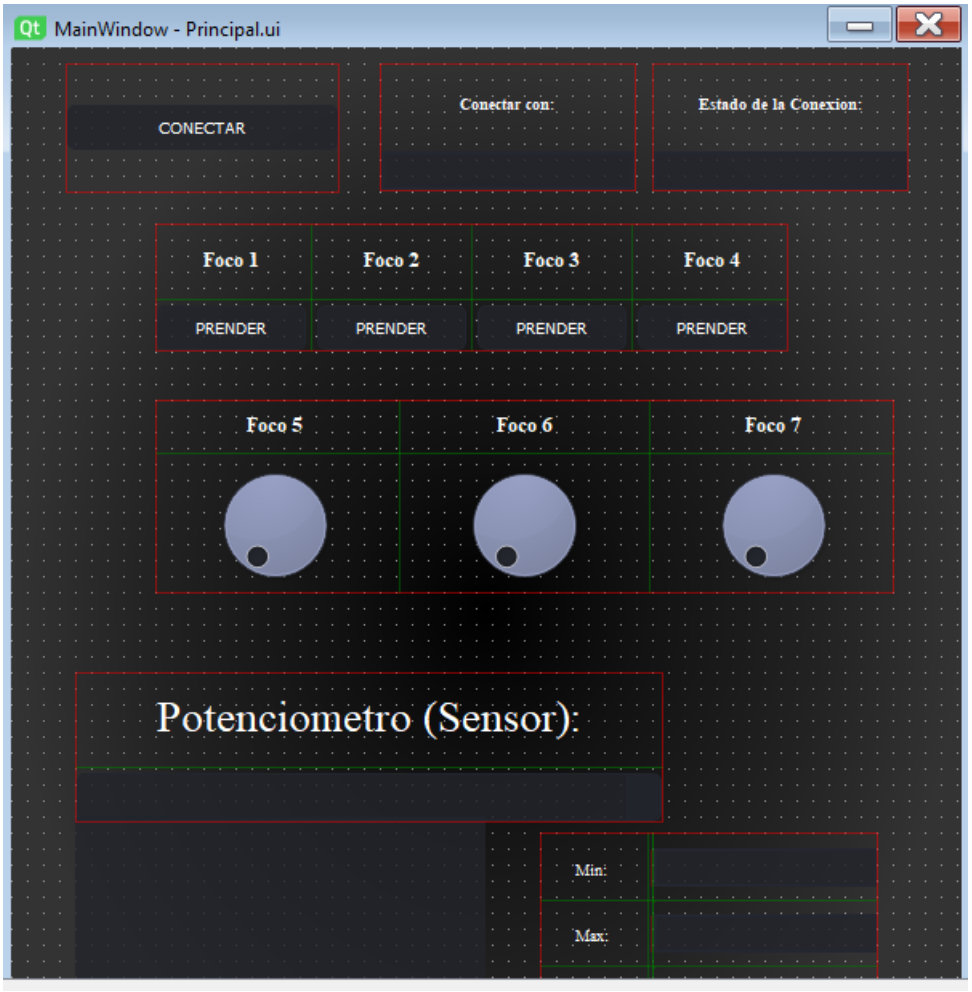
Integrantes:

- Gonzalez Santiago Jose Alonso
- Moreno Wilches Fernando
- Verástegui Cruz Carlos

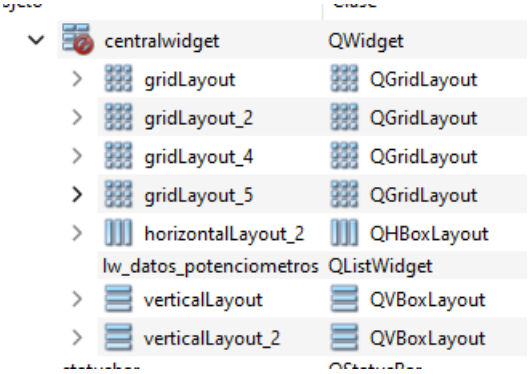
Fecha de entrega: 23 de Mayo del 2023

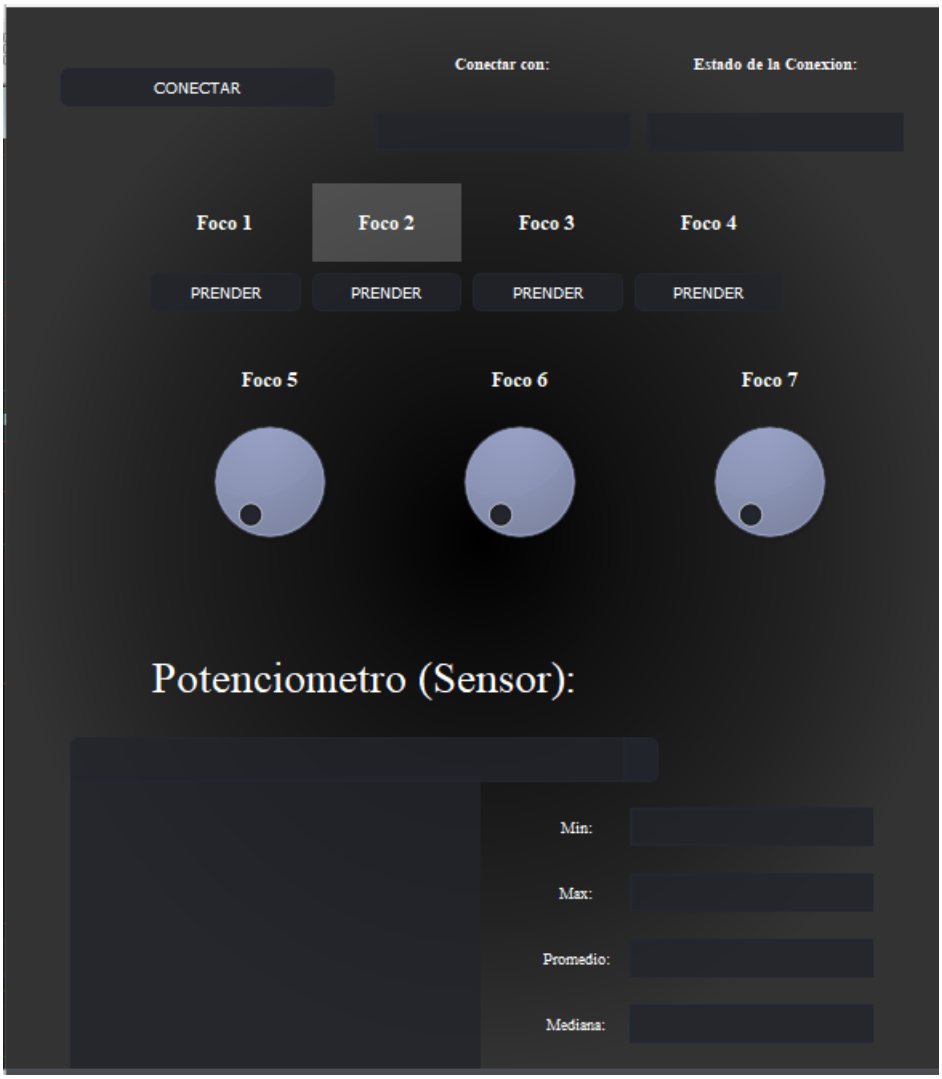
Documentación

Diseño de nuestro Equipo agregado al código del maestro



Utilizando diferentes Layouts para mejorar el diseño y organización





En ejecución

En términos de código, empezamos definiendo la lógica del código

```
    # Área de los Signals
    self.btn_accion.clicked.connect(self.accion)
```

Esta línea establece una conexión entre la señal clicked del botón btn_accion y la función accion. Básicamente, le dice al programa que cuando se haga clic en el botón btn_accion, se debe llamar a la función accion. Esto se utiliza para asociar una acción específica a un botón, por ejemplo, para realizar alguna operación cuando se presiona el botón.

```
self.btn_foco1.clicked.connect(self.control_foco1)
self.btn_foco2.clicked.connect(self.control_foco2)
self.btn_foco3.clicked.connect(self.control_foco3)
self.btn_foco4.clicked.connect(self.control_foco4)
```

Aquí, se establece una conexión similar entre la señal clicked del botón btn_foco1 y la función control_foco1. Esto significa que cuando se haga clic en el botón btn_foco1, se llamará a la función control_foco1.

Las líneas siguientes tienen una estructura similar y establecen conexiones entre las señales clicked de otros botones (btn_foco2, btn_foco3 y btn_foco4) y sus respectivas funciones de control (control_foco2, control_foco3 y control_foco4).

```
self.dial_foco5.setMinimum(0)
self.dial_foco5.setMaximum(255)
self.dial_foco5.setSingleStep(1)
self.dial_foco5.setValue(0)
self.dial_foco5.valueChanged.connect(self.control_foco5)

self.dial_foco6.setMinimum(0)
self.dial_foco6.setMaximum(255)
self.dial_foco6.setSingleStep(1)
self.dial_foco6.setValue(0)
self.dial_foco6.valueChanged.connect(self.control_foco6)

self.dial_foco7.setMinimum(0)
self.dial_foco7.setMaximum(255)
self.dial_foco7.setSingleStep(1)
self.dial_foco7.setValue(0)
self.dial_foco7.valueChanged.connect(self.control_foco7)
```

Estas líneas configuran el dial dial_foco5 estableciendo su valor mínimo como 0, valor máximo como 255, incremento de 1 y valor inicial como 0. Luego, se establece una conexión entre la señal valueChanged del dial y la función control_foco5. Esto significa que cuando el valor del dial cambie, se llamará a la función control_foco5 y se le pasará el nuevo valor del dial como argumento.

Las líneas similares siguientes configuran los diales dial_foco6 y dial_foco7 de manera análoga, estableciendo sus valores mínimo, máximo, incremento y valor inicial, así como las conexiones con las funciones de control correspondientes (control_foco6 y control_foco7).

```
self.foco1 = 0
self.foco2 = 0
self.foco3 = 0
self.foco4 = 0
self.foco5 = 0
self.foco6 = 0
self.foco7 = 0
```

Estas líneas inicializan las variables foco1, foco2, foco3, foco4, foco5, foco6 y foco7 con el valor 0. Estas variables probablemente se utilizan para almacenar y rastrear los valores de los focos en la aplicación.

```
    self.arduino = None

    # ACTUALIZACION .... VIERNES 19 DE MAYO

    self.txt_estado.setEnabled(False)
    self.txt_com.setText("/dev/cu.usbmodem101")
```

Estas líneas configuran el estado de dos elementos de la interfaz de usuario. La primera línea deshabilita el campo de texto txt_estado, lo que significa que el usuario no puede modificar su contenido. La segunda línea establece el texto "/dev/cu.usbmodem101" en el campo de texto txt_com.

```
    # self.btn_accion.setFocus()
    # or ...
    self.txt_com.selectAll()
    self.txt_com.setFocus()

    self.segundoPlano = QtCore.QTimer()
```

Estas líneas de código se encargan de configurar el campo de texto txt_com. La línea self.txt_com.selectAll() selecciona todo el texto en el campo de texto, lo que facilita al usuario reemplazarlo o borrarlo de manera rápida si así lo desea.

La línea self.txt_com.setFocus() establece el enfoque en el campo de texto, lo que significa que el cursor estará allí y el usuario podrá comenzar a escribir directamente sin tener que hacer clic en el campo de texto primero.

```
self.segundoPlano = QtCore.QTimer()
self.segundoPlano.timeout.connect(self.lectura_datos_arduino)
```

Aquí se crea un nuevo objeto QTimer llamado segundoPlano utilizando la clase QtCore.QTimer. QTimer es una clase que permite ejecutar tareas en segundo plano con un intervalo de tiempo específico. Luego, se establece una conexión entre la señal timeout del temporizador y la función lectura_datos_arduino. Esto significa que cuando se alcance el intervalo de tiempo especificado en el temporizador, se llamará a la función lectura_datos_arduino. Esta conexión permite realizar una acción periódica o repetitiva en respuesta a la señal de tiempo transcurrido.

```
#agrega datos al combobox
self.cb_potenciometros.addItem("Sensor 1", 10)
self.cb_potenciometros.addItem("Sensor 2", 12)
self.cb_potenciometros.addItem("Sensor 3", 16)
self.cb_potenciometros.currentIndexChanged.connect(self.cambiaComboBoxSeleleccion)
```

Estas líneas agregan elementos al ComboBox cb_potenciometros. Cada elemento tiene un texto de visualización y un valor asociado. En este caso, se agregan tres elementos al ComboBox:

El primer elemento tiene el texto "Sensor 1" y el valor asociado es 10.

El segundo elemento tiene el texto "Sensor 2" y el valor asociado es 12.

El tercer elemento tiene el texto "Sensor 3" y el valor asociado es 16.

Estos elementos se utilizan para mostrar opciones al usuario y representar diferentes sensores.

Esta línea establece una conexión entre la señal currentIndexChanged del ComboBox cb_potenciometros y la función cambiaComboBoxSeleleccion. La señal currentIndexChanged se emite cuando el índice seleccionado en el ComboBox cambia. Al establecer esta conexión, cuando el usuario seleccione una opción diferente en el ComboBox, se llamará a la función cambiaComboBoxSeleleccion para manejar el evento.

```
# datos leidos de los sensores ---z igual se podria agregar a : min, max, promedio, mediana
self.datos_sensores = {10: [],  # sensor 1
                       12: [],  # sensor 2
                       16: []}  # sensor 3
```

Estas líneas crean un diccionario llamado datos_sensores. Este diccionario se utiliza para almacenar los datos leídos de los sensores. Cada sensor está asociado con una clave que corresponde al valor asociado a cada opción del ComboBox anteriormente mencionado. Inicialmente, las listas asociadas a cada sensor están vacías, pero posteriormente se pueden agregar datos a estas listas según sea necesario.

```
com = self.txt_com.text()
```

Esta línea obtiene el texto ingresado en el campo de texto txt_com y lo asigna a la variable com. Supongo que este texto representa el puerto COM al que está conectado el Arduino.

```
if self.arduino == None:
    self.btn_accion.setText("DESCONECTAR")
    self.txt_estado.setText("CONECTADO")
    ##establece la conexion con arduino la primera vez  ## COM5
    self.arduino = controlador.Serial(com, baudrate=9600, timeout=1)
    self.segundoPlano.start(100)
```

Aquí se verifica si la variable arduino es None, lo que significa que no hay conexión establecida previamente con Arduino. En este caso, se cambia el texto del botón btn_accion a "DESCONECTAR" y el texto del campo txt_estado a "CONECTADO". Luego, se establece la conexión con Arduino utilizando el puerto COM obtenido anteriormente y se asigna a la variable arduino. La función controlador.Serial se utiliza para crear una instancia de un objeto de conexión serie con Arduino. Luego, se inicia el temporizador segundoPlano con un intervalo de 100 ms.

```
elif self.arduino.isOpen():  # Si das clic y esta abierta la conexion, entonces se pasa a desconectar
    self.btn_accion.setText("RECONECTAR")
    self.txt_estado.setText("DESCONECTADO")
    self.segundoPlano.stop()
    self.arduino.close()
```

En este bloque elif, se verifica si la conexión con Arduino está abierta utilizando el método isOpen() del objeto arduino. Si la conexión está abierta, significa que se está desconectado. En ese caso, se cambia el texto del botón a "RECONECTAR", el texto del campo txt_estado a "DESCONECTADO" y se detiene el temporizador segundoPlano. Luego, se cierra la conexión con Arduino utilizando el método close().

```
else:  # reconectar
    self.btn_accion.setText("DESCONECTAR")
    self.txt_estado.setText("RECONECTADO")
    self.arduino.open()
    self.segundoPlano.start(100)
```

En el bloque else, se asume que la conexión con Arduino está cerrada y se procede a reconectar. Se cambia el texto del botón a "DESCONECTAR", el texto del campo txt_estado a "RECONECTADO". Luego, se abre la conexión con Arduino utilizando el método open(). Por último, se inicia el temporizador segundoPlano con un intervalo de 100 ms.

```
def generaCadenaControlFocos(self):
    # 1-1-0-1-100-240-50
    cadena = str(self.foco1) + "-" + str(self.foco2) + "-" + \
             str(self.foco3) + "-" + str(self.foco4) + "-" + \
             str(self.foco5) + "-" + str(self.foco6) + "-" + str(self.foco7)
    print(cadena)
    # Si se ha establecido la conexion con arduno y esta se encuentra abierta...
    if not self.arduino is None and self.arduino.isOpen():
        self.arduino.write(cadena.encode())
```

Esta función llamada generaCadenaControlFocos se encarga de generar una cadena de control para los focos y enviarla a Arduino a través de la conexión serial

```
# 1-1-0-1-100-240-50
cadena = str(self.foco1) + "-" + str(self.foco2) + "-" + \
         str(self.foco3) + "-" + str(self.foco4) + "-" + \
         str(self.foco5) + "-" + str(self.foco6) + "-" + str(self.foco7)
print(cadena)
```

En esta parte, se crea una cadena llamada cadena concatenando los valores de los focos separados por guiones ("-"). Los valores de los focos se obtienen de las variables foco1, foco2, foco3, foco4, foco5, foco6 y foco7. Estas variables presumiblemente representan el estado o control de los focos.

```
if not self.arduino is None and self.arduino.isOpen():
    self.arduino.write(cadena.encode())
```

En esta sección, se verifica si la conexión con Arduino está establecida y abierta. Primero, se comprueba si self.arduino no es None, lo que significa que la conexión ha sido establecida previamente. Luego, se verifica si la conexión está abierta utilizando el método isOpen() del objeto arduino. Si ambas condiciones se cumplen, se envía la cadena cadena a Arduino utilizando el método write(). Antes de enviarla, se codifica la cadena en formato de bytes utilizando el método encode().

```
def control_foco1(self):
    # Obtiene el texto del botón "btn_foco1"
    txtBoton = self.btn_foco1.text()
    if txtBoton == "PRENDER":
        # Si el texto es "PRENDER", cambia el texto a "APAGAR"
        self.btn_foco1.setText("APAGAR")
        # Asigna el valor 1 al foco1 para indicar que está encendido
        self.foco1 = 1
    else:
        # Si el texto es "APAGAR", cambia el texto a "PRENDER"
        self.btn_foco1.setText("PRENDER")
        # Asigna el valor 0 al foco1 para indicar que está apagado
        self.foco1 = 0
    # Genera la cadena de control de los focos
    self.generaCadenaControlFocos()
```

La primera linea obtiene el texto actual del botón btn_foco1 y lo asigna a la variable txtBoton. El propósito de esto es verificar el estado actual del foco representado por el botón.

Después se realiza una verificación del texto del botón. Si el texto es "PRENDER", significa que el foco está apagado. En ese caso, se cambia el texto del botón a "APAGAR" y se asigna el valor 1 a la variable foco1 para indicar que el foco está encendido. Si el texto es "APAGAR", significa que el foco está encendido. En ese caso, se cambia el texto del botón a "PRENDER" y se asigna el valor 0 a la variable foco1 para indicar que el foco está apagado.

Al finalizar se debe actualizar el estado del foco, se llama al método generaCadenaControlFocos para generar la cadena de control de los focos y enviarla a Arduino a través de la conexión serial. Este método se encarga de concatenar los estados de los focos y enviarlos a través de la conexión serial, como se explicó anteriormente.

Se repite el mismo procedimiento co los focos restantes

```
def control_foco5(self):
    # Obtiene el valor actual del dial_foco5
    self.foco5 = self.dial_foco5.value()
    # Genera la cadena de control de los focos
    self.generaCadenaControlFocos()

def control_foco6(self):
    # Obtiene el valor actual del dial_foco6
    self.foco6 = self.dial_foco6.value()
    # Genera la cadena de control de los focos
    self.generaCadenaControlFocos()

def control_foco7(self):
    # Obtiene el valor actual del dial_foco7
    self.foco7 = self.dial_foco7.value()
    # Genera la cadena de control de los focos
    self.generaCadenaControlFocos()
```

Los métodos control_foco5, control_foco6 y control_foco7 también siguen una estructura similar. En lugar de basarse en el texto de un botón, estos métodos obtienen el valor actual de un dial (dial_foco5, dial_foco6, dial_foco7). Se asigna ese valor a la variable de foco correspondiente (foco5, foco6, foco7). Luego, se llama al método generaCadenaControlFocos para generar la cadena de control de los focos actualizada.

```
def cambiaComboBoxSeleleccion(self):
    clave = self.cb_potenciometros.currentData()
    print(clave)
    self.lw_datos_potenciometros.clear() ##limpia los registros anteriores para cambiar de sensor...
    self.datos_sensores[clave] = [] #limpia el registro  --- podria no limpiarse y entonces sería un historico...
```

Este método se ejecuta cuando se produce un cambio en la selección del combo box cb_potenciometros. Veamos qué hace cada línea:

clave = self.cb_potenciometros.currentData(): Se obtiene el valor asociado al ítem seleccionado actualmente en el combo box. Este valor es obtenido utilizando el método currentData(). El propósito de esta línea es obtener la clave del sensor seleccionado.

print(clave): Imprime el valor de clave en la consola. Esto es útil para verificar qué clave se ha seleccionado.

self.lw_datos_potenciometros.clear(): Limpia los registros anteriores en el widget lw_datos_potenciometros. Este widget podría ser una lista o una vista de lista donde se muestran los datos relacionados con los potenciómetros.

self.datos_sensores[clave] = []: Inicializa o limpia el registro de datos asociado a la clave del sensor seleccionado en datos_sensores. datos_sensores parece ser un diccionario donde se almacenan los datos leídos de diferentes sensores. En esta línea, se vacía el registro correspondiente al sensor seleccionado, lo cual puede interpretarse como limpiar el historial de lecturas del sensor o prepararlo para almacenar nuevos datos.

```
min_val = min(self.datos_sensores[clave])
max_val = max(self.datos_sensores[clave])

avg_val = sum(self.datos_sensores[clave]) / len(self.datos_sensores[clave])
avg_val = round(avg_val, 2)

mediana_val = median(self.datos_sensores[clave])

self.txt_min.setText(str(min_val))  # Actualiza el campo de texto con el valor mínimo
self.txt_max.setText(str(max_val))  # Actualiza el campo de texto con el valor máximo
self.txt_promedio.setText(str(avg_val))  # Actualiza el campo de texto con el valor promedio
self.txt_mediana.setText(str(mediana_val))  # Actualiza el campo de texto con el valor de la mediana
```

Estas líneas calculan el valor mínimo (min_val), máximo (max_val), promedio (avg_val) y mediana (mediana_val) de los datos del sensor seleccionado. Utilizando las funciones min(), max() y sum() de Python, se obtienen estos valores a partir de la lista de datos correspondiente al sensor seleccionado en el diccionario self.datos_sensores.

Después de calcular estos valores, se actualizan los campos de texto en la interfaz gráfica con los resultados. Los métodos setText() se utilizan para asignar los valores convertidos a cadena (str()) a los campos de texto self.txt_min, self.txt_max, self.txt_promedio y self.txt_mediana.

```
except KeyboardInterrupt:
    self.close()  # Maneja la interrupción de teclado (Ctrl+C) y cierra la aplicación
except Exception as E:
    print(E)  # Muestra cualquier otra excepción en la consola de salida
```

Se manejan u bloque de excepciones

```
def closeEvent(self, event):
    # reply = QtWidgets.QMessageBox.question(self, "Mensaje", "Esta seguro de salir?", QtWidgets.QMessageBox.Yes,
    #                                        QtWidgets.QMessageBox.No)

    # if reply == QtWidgets.QMessageBox.Yes:
    self.segundoPlano.stop()
    event.accept()
    # else:
    #     event.ignore()
```

Para finalizar, este método closevent se llama cuando se produce un evento de cierra, cuando se cierra la ventana, por lo que en este caso se llama al método stop del temporizador para detenerlo.

Documentacion Arduino

Aquí se declaran las variables para los LED que se controlarán mediante señales de encendido y apagado (ON/OFF). Los pines asociados a estos LED son los pines digitales 10, 11, 12 y 13 respectivamente.

```
//ON / OFF
int led1 = 10;
int led2 = 11;
int led3 = 12;
int led4 = 13;
```

```
//PWM
int led5 = 3;
int led6 = 5;
int led7 = 6;
```

Aquí se declaran las variables para los LED que se controlaran mediante señales PWM.

```
//Potenciometros (sensores)
int pot1 = A0;
int pot2 = A1;
int pot3 = A2;
```

Aquí se declaran las variables para los potenciómetros utilizados como sensores, los pines aalogicos A0, A1 y A2 estan conectados a estos potenciómetros

```
void setup() {
  // put your setup code here, to run once:

  pinMode(led1, OUTPUT); //SOLO LOS PINDES DIGITALES QUE SERAN USADOS COMO ON/OFF, REQUIEREN SE ESTABLEZCA
  pinMode(led2, OUTPUT); //EL MODO DE TRABAJO DEL PIN... EN ESTE CASO , COMO OUTPUT, DEBIDO A QUE SE
  pinMode(led3, OUTPUT); // PRENDERAN O APAGARAN LOS LEDS
  pinMode(led4, OUTPUT);
```

En esta parte del código, se escalece el modo de trabajo de los pines que controlan los LED. Usando la función pinMode se configura los pines led1-led4 como pines de salida. Esto significa que se utilizan para acceder y apagar los LED conectados a ellos

```
  Serial.begin(9600);
  Serial.setTimeout(100);
}
```

En estas líneas s configuran la comunicación serial, la cual usando la función Serial.begi(9600) inicalizaliza la comunicación serial que velocidad de retransmitió de 9600 baudios. Lo cual permite comunicación entre arudino y toros dispositivos conectados a través del puerto serial.

La función setTimeOut e4stabelce el tiempo de espera máximo para recibir datos atravesó de la comunicación serial en este caso se estable un tiempo de espera de 100 milisegundos

```
char *c;
char *token;

int vPot1, vPot2, vPot3;
String cadenaSensores;

String respuesta;
void loop() {
  // put your main code here, to run repeatedly:

  //OBTIENE EL VALOR DE LOS POTENCIOMETROS
  vPot1 = analogRead(pot1) * 1;  //0 - 1023
  vPot2 = analogRead(pot2) * 2;  //* 1  ... * 2  y * 3 --- solo solo valores para hacer prueba, en la practica real no deben ir
  vPot3 = analogRead(pot3) * 3;

  //CONCATENA LOS VALORES DE LOS POTENCIOMETROS EN UNA UNICA CADENA PARA FACILITAR EL ENVIO DE LA INFORMACION
  cadenaSensores = String(vPot1) + "-" + String(vPot2) + "-" + String(vPot3);

  //ENVIA LA INFORMACION DE LOS POTENCIOMETROS A PYTHON A TRAVES DE COMUNICACION SERIAL
  Serial.println(cadenaSensores);
```

En esta parte del código, se realiza la lectura de los valores de los potenciómetros analógicos y se envían a través de la comunicación serial. Aquí tienes una explicación detallada:

Las variables vPot1, vPot2 y vPot3 se utilizan para almacenar los valores de los potenciómetros después de la lectura analógica. Cada valor se multiplica por un factor (1, 2 y 3 respectivamente) para realizar una escala o ajuste. En la práctica real, estos factores se pueden ajustar según las necesidades específicas.

El código analogRead(pot1) lee el valor analógico del pin pot1 y lo multiplica por 1. El resultado se almacena en la variable vPot1. Se realiza un proceso similar para vPot2 y vPot3, pero con diferentes pines y factores de escala.

Después de obtener los valores de los potenciómetros, se concatenan en una única cadena llamada cadenaSensores. Esto se realiza utilizando la función String(), que convierte los valores enteros en cadenas de texto. Los valores se separan con guiones ("-") para facilitar el posterior procesamiento en Python u otro lenguaje de programación.

Finalmente, la cadena cadenaSensores se envía a través de la comunicación serial utilizando la función Serial.println(). Esto permite que la información de los potenciómetros se transmita a un dispositivo conectado (como Python) para su posterior procesamiento.

```
if(Serial.available()>0){ //si hay informacion en el buffer para ser leida... entonces se leera

  respuesta = Serial.readString(); //leera la cadena recibida por el usuario (comunicacion serial...)
  respuesta.replace("\n","");
  respuesta.replace("\r","");
  c = respuesta.c_str();

  token = strtok(c,"-"); //Tokeniza la cadena
```

Aquí s e Compruebe asi hay información disponible en el buffer de comunicación serial utilizando Serial.avialble. Si hay información, se ejecuta el bloque del código if+

Luego se utilizar Serial.readTstrig para leer la cadena completa enviada a través de la comunicación serial y se almacena en la variable “respuesta”., Después se utilizan las funciones replace para eliminar los caracteres de nueva linea y retorno de carro de la cadea de respuesta.

Luego, se utiliza strtok(c, "-") para tokenizar la cadena c utilizando el delimitador -. La función strtok() devuelve el primer token encontrado en la cadena y establece el puntero interno para buscar el siguiente token en las llamadas posteriores. En este caso, se utiliza - como delimitador para separar los diferentes valores en la cadena.

```
String temp;
int contador  = 0;
while(token != NULL){
  temp = String(token);
//Serial.println("Token: " + temp + " Contador: " + String(contador));
```

Después de obtener el primer token, se inicia un bucle while que se ejecutará mientras el token no sea NULL. En cada iteración del bucle, se realiza lo siguiente:

La variable temp se inicializa con el valor del token actual, pero convertido en un objeto de tipo String. Esto es útil para poder utilizar las funciones de manipulación de cadenas proporcionadas por la clase String en Arduino.

Se utiliza un comentario para imprimir en la consola de depuración (comentado con //) el valor del token y el contador. Esto puede ser útil para verificar los valores durante la ejecución del programa.

```
switch(contador){
  case 0:
    digitalWrite(led1, temp.toInt());
  break;
  case 1:
    digitalWrite(led2, temp.toInt());
  break;
  case 2:
    digitalWrite(led3, temp.toInt());
  break;
  case 3:
    digitalWrite(led4, temp.toInt());
  break;
  case 4:
    analogWrite(led5, temp.toInt());
  break;
  case 5:
    analogWrite(led6, temp.toInt());
  break;
  case 6:
    analogWrite(led7, temp.toInt());
  break;
}
```

Dentro del bucle while, se encuentra un bloque switch que se encarga de realizar diferentes acciones dependiendo del valor del contador. En este caso, se utiliza el valor del contador para determinar qué pin debe ser controlado y cómo debe ser controlado.

Para los casos del 0 al 3, se utilizan las funciones digitalWrite() para controlar los pines led1, led2, led3 y led4, respectivamente. El valor pasado como segundo argumento a digitalWrite() es temp.toInt(), que convierte el valor del token en un entero.

Para los casos del 4 al 6, se utilizan las funciones analogWrite() para controlar los pines led5, led6 y led7, respectivamente. Nuevamente, se utiliza temp.toInt() para convertir el valor del token en un entero.

```
contador++;
token = strtok(NULL,"-");
```

Después de cada iteración del bucle while, se incrementa el contador en 1 para avanzar al siguiente caso en el bloque switch. Luego, se obtiene el siguiente token utilizando strtok(NULL, "-"), que busca el siguiente token en la cadena utilizando - como delimitador.

```
  delay(100); //milisegundos - por lo que se utilizo e python QTimer, el temporizador

}
```

Al final del bucle loop(), se agrega un pequeño retardo de 100 milisegundos utilizando delay(100). Esto proporciona un breve tiempo de espera antes de que se ejecute el siguiente ciclo del bucle loop(). El retardo puede ser útil para evitar una ejecución excesivamente rápida del código y permitir que otros procesos o acciones tengan lugar entre las iteraciones del bucle.