

**PROYECTO FINAL: INTERCONEXIÓN DEL CÓDIGO  
DE ARDUINO Y EL CÓDIGO DE PYTHON**

**PROFESOR: Rafael García**

<b>OBJETIVO .....</b>	<b>2</b>
<b>RECURSOS NECESARIOS .....</b>	<b>2</b>
<b>DESARROLLO PRÁCTICO .....</b>	<b>2</b>
<b>Tarea 1: Configuración de la placa Arduino .....</b>	<b>2</b>
<b>Tarea 2: Conectar Arduino con Raspberry Pi .....</b>	<b>4</b>
<b>Tarea 3: Uso de Python para leer y escribir datos en serie a Arduino .....</b>	<b>4</b>
<b>FOTOS DEL MONTAJE .....</b>	<b>8</b>
<b>BIBLIOGRAFÍA .....</b>	<b>10</b>

## **OBJETIVO:**

El objetivo de este trabajo es el de usar la interfaz de serie de la placa de Arduino para intercambiar datos entre un Arduino y una Raspberry Pi, así como controlar la placa de Arduino usando Python desde la Raspberry Pi.

## **RECURSOS NECESARIOS:**

- PC con acceso a Internet.
- Raspberry Pi con cable de alimentación y conexión de red por cable Ethernet o inalámbrica.
- Placa de Arduino UNO.
- Cable USB.
- Resistencia de 10 K $\Omega$  y una LDR.

## **DESARROLLO PRÁCTICO:**

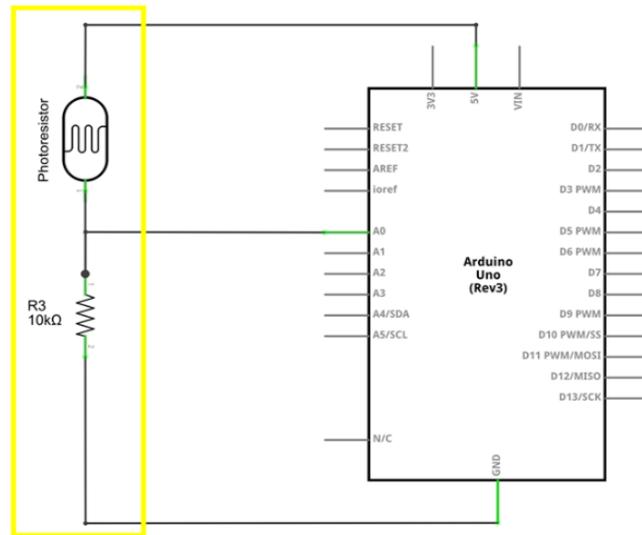
### **Tarea 1: Configuración de la placa Arduino**

#### **Montaje de un divisor de tensión para conectar un sensor lumínico (LDR) al Arduino**

Realizar el montaje del esquema. El punto de unión entre la fotorresistencia y el resistor se conecta a la entrada A0 analógica de la placa de Arduino. El pin de conexión a tierra de la placa se conecta al otro lado de la resistencia, mientras que el pin de +5 V de la placa se conecta al otro lado de la fotorresistencia. Mientras más luz recibe el cabezal de la fotorresistencia, menor es la resistividad. Esto significa que con más luz, el pin de entrada A0 analógica medirá un voltaje mayor que en condiciones de oscuridad.

Según las condiciones de luz sobre la fotorresistencia, el pin A0 medirá valores de tensión de entrada de 0 a +5 V y los transformará en valores digitales de 0 a 1023.

## Esquema de Arduino:



## Programación de Arduino

Para algunos sensores y aplicaciones en tiempo real, es necesario ejecutar el código de aplicación de procesamiento directamente en la placa de microcontrolador de Arduino. En este montaje, el código de Arduino se usa para leer valores de los sensores analógicos y para enviar los valores por un canal de comunicación serie para su procesamiento con Python en Raspberry Pi. El código mismo también proporciona un mecanismo para cambiar el pin analógico predeterminado que se usa para leer los datos almacenados. Si se envían datos por la interfaz serie a la placa Arduino, se actualizará el número de pin de entrada analógica en función del valor leído en la interfaz de serie.

El código a grabar en la placa Arduino para poder enviar continuamente los valores medidos como una cadena a la interfaz serie. es el siguiente:

```
// default analog pin to read from:
int analogInputPin = 0;

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // if any data on the serial port are available
  // read it and try to update the analogInputPin
  // based on the number that was read on the serial
  if (Serial.available() > 0) {
    analogInputPin = Serial.parseInt();
  }

  // read the analog value:
  int analogInputPinValue = analogRead(analogInputPin);
```

```

// print the results to the serial port:
// the output should have the following form: INPUTPIN:VALUE
// followed by a newline character
Serial.print(analogInputPin);
Serial.print(':');
Serial.print(analogInputPinValue);
Serial.println('');

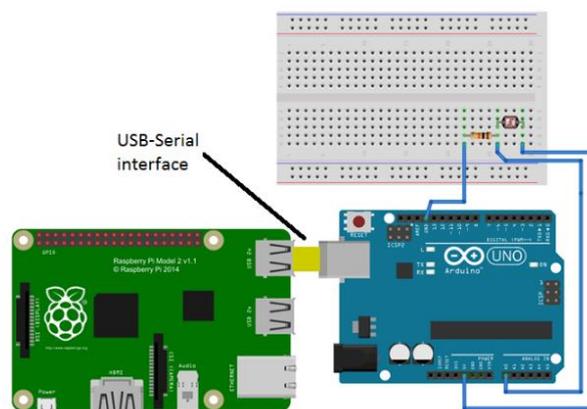
// wait 50 milliseconds before the next loop
delay(50);
}

```

## Tarea 2: Conectar Arduino con Raspberry Pi

### Conectar Arduino a Raspberry Pi

Con un cable USB, conectar Arduino a Raspberry Pi. El cable USB entre Arduino y Raspberry Pi no solo se usa para alimentar a Arduino, sino que además proporciona un canal de comunicación serie para controlar y monitorear a Arduino desde las aplicaciones que se ejecutan en Raspberry Pi.



## Tarea 3: Uso de Python para leer y escribir datos en serie a Arduino

### Configuración

El canal de transporte serie entre Arduino y Raspberry Pi en Linux está generalmente identificado como dispositivo en /directorio de dispositivos con el nombre ttyACM0 o ttyUSB0 (/dev/ttyACM0 o /dev/ttyUSB0).

Hay que identificar el nombre del dispositivo de comunicación serie de Arduino en el sistema operativo Linux de PL-App ejecutando el código de la celda siguiente. La celda a continuación contiene un script de prueba que identifica el nombre del dispositivo. El nombre del dispositivo de comunicación serie será necesario más adelante:

```
%%bash
# ^^^ The commands below are to be executed as Linux Bash commands.
# You can get the same output by opening a terminal connection to the device and
executing these commands manually.
dmesg | grep -v disconnect | grep -Eo "tty(ACM|USB)." | tail -1
```

Una vez identificado el nombre del dispositivo serie de Arduino, podemos seguir implementando una comunicación de serie personalizada de Arduino.

### Apertura de la interfaz de serie en Python

La biblioteca de Python pyserial se usa para leer y escribir datos entrantes y salientes de una interfaz de serie. Para usar la biblioteca pyserial, hay que importar el módulo serial mediante la orden:

```
import serial
```

Seguidamente, se debe crear un objeto, “**ser**” que represente al dispositivo de interfaz de serie local (**/dev/ttyUSB0**), especificado como el primer parámetro del comando `ser = serial.Serial('/dev/ttyUSB0', 9600)`. El segundo parámetro (9600) define la velocidad de la interfaz de serie. Este número debe ser el mismo que se especificó en el código de Arduino `Serial.begin(9600)`; en la función `setup()`:

```
# make sure to use the correct serial communication device name
# e.g. /dev/ttyUSB0, /dev/ttyUSB1, /dev/ttyACM0, ...
# as you discovered in the steps above
ser = serial.Serial('/dev/ttyUSB0', 9600)
```

Al establecerse la conexión serie, se pueden usar las diversas funciones del objeto “**ser**” para leer y escribir datos entrantes y salientes de la interfaz de serie:

- `flushInput()`: para purgar todo lo que podría haberse almacenado en búfer de entrada antes de leer
- `readline()`: para leer una sola línea de Bytes de la interfaz de serie
- `write()`: para escribir una cadena de Bytes a la interfaz de serie
- `close()`: para cerrar la interfaz de serie (solo una aplicación por vez puede abrir la interfaz de serie)

El código de Arduino envía datos en este formato: `INPUTPIN:VALUE`, donde `INPUTPIN` es el número de pin donde se tomó la medición, y `VALUE` es el valor real medido.

Seguidamente pasamos a ejecutar el código siguiente para leer los datos de la interfaz de serie línea por línea e imprimirlos en la salida:

```
# loop until manually stopped
# first flush possibly existing data in the input buffer:
ser.flushInput()
while True:
    try:
        # read a single line from the serial interface represented by the ser object
        lineBytes = ser.readline()
        # convert Bytes returned by the ser.readline() function to String
        line = lineBytes.decode('utf-8')
        # print the read line to the output
        print(line)

    except KeyboardInterrupt:
        break # stop the while loop
```

También se puede usar la función `write()` del objeto `ser` para escribir datos en la interfaz serie. Arduino podrá procesar estos datos localmente. En este caso, Arduino acepta como entrada los datos en serie que representan el pin de entrada analógica del que se debe leer. Al ejecutar el código siguiente, podemos cambiar el pin de entrada del Arduino del pin 0 predeterminado para entradas analógicas al pin 5. Conectando cualquier sensor o solo un simple cable de acoplamiento al pin analógico 5, podremos medir los valores de tensión en el pin analógico 5:

```
# write output data to the serial interface
# these data are read by Arduino's serial:
inputPin = 5
# the write() function expects a bytes parameter
# therefore the integer inputPin is first turned into a string with the str() function
# and then the string is turned into Bytes by the encode() function
inputPinStr = str(inputPin)
inputPinStrBytes = inputPinStr.encode()
ser.write(inputPinStrBytes)

# the same code as above, this time it should read data from a new pin
# loop until manually stopped
# first flush possibly existing data in the input buffer:
ser.flushInput()
while True:
    try:
        # read a single line from the serial interface represented by the ser object
```

```

lineBytes = ser.readline()
# convert Bytes returned by the ser.readline() function to String
line = lineBytes.decode('utf-8')
# print the read line to the output
print(line)

```

```

except KeyboardInterrupt:
    break # stop the while loop

```

El código final siguiente no escribirá simplemente la línea de lectura a la salida, sino que la analizará. La línea debe tener el formato INPUTPIN:VALUE y puede analizarse con una variedad de valores separados con el separador : y la función split.

El código también iterará entre los pines de entrada analógica A0 y A5 de la placa Arduino escribiendo un nuevo mensaje en la interfaz de serie que contiene el número de pin solicitado. Una vez que este mensaje se reciba en la placa Arduino, se procesará localmente y se cambiará el pin de entrada que se leerá:

```

inputPin = 0
# the same code as above, this time it should read data from a new pin
# loop until manually stopped
# first flush possibly existing data in the input buffer:
ser.flushInput()
while True:
    try:
        # read a single line from the serial interface represented by the ser object
        lineBytes = ser.readline()
        # convert Bytes returned by the ser.readline() function to String
        line = lineBytes.decode('utf-8')
        # split the string that should have the following form: INPUTPIN:VALUE
        # into an array
        data = line.split(":")
        # if the array has two elements, the line was correctly formatted as INPUTPIN:VALUE
        if(len(data) == 2):
            inputpin = data[0]
            value = data[1]
            print("Input pin {} has value {}".format(inputpin, value))

        # iterate the input pin of the Arduino to be read from
        inputPin = inputPin + 1
        if(inputPin > 5):
            inputPin = 0
        # send a message to the Arduino to update the pin to read from

```

```
ser.write(str(inputPin).encode())
```

```
except KeyboardInterrupt:  
    break # stop the while loop
```

Debemos cerrar y liberar la interfaz de serie una vez que ya no se la necesite. Esto permite que otras aplicaciones usen la interfaz.

```
ser.close()
```

## FOTOS DEL MONTAJE:

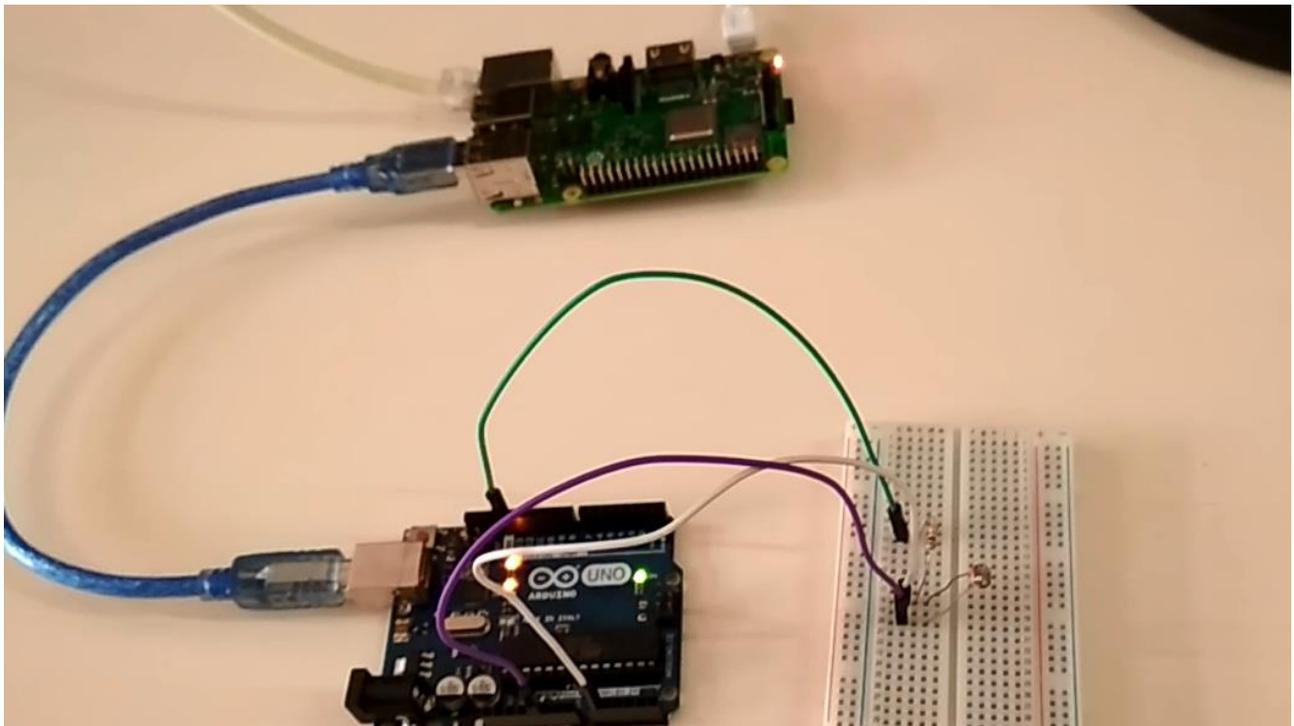


Foto de la conexión entre la Raspberry y la placa de Arduino con el sensor.

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 O
+ -> Run Code Blockly

In [24]: 1 import serial
2
3 ser = serial.Serial('/dev/ttyACM0', 9600)
4
5 inputPin = 0
6
7 ser.flushInput()
8 while True:
9     try:
10        lineBytes = ser.readline()
11        line = lineBytes.decode('utf-8')
12        data = line.split(",")
13        if(len(data) == 2):
14            inputpin = data[0]
15            value = data[1]
16            print("Entrada pín analógica {} tiene el valor {}".format(inputpin, value))
17
18            inputPin = inputPin + 1
19            if(inputPin > 5):
20                inputPin = 0
21                ser.write(str(inputPin).encode())
22
23        except KeyboardInterrupt:
24            break
Entrada pín analógica 2 tiene el valor 291

In [ ]: 1
```

Foto del código de la Rasperry en Python

```
PL-App 3.2.6.4 Interconexión con el código de Ardu... Last Checkpoint: hace 10 minutos (saved changes) Python 3
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
+ -> Run Code Blockly

In [7]: 1 import serial
2
3 ser = serial.Serial('/dev/ttyACM0', 9600)
4
5 inputPin = 0
6
7 ser.flushInput()
8 while True:
9     try:
10        lineBytes = ser.readline()
11        line = lineBytes.decode('utf-8')
12        data = line.split(",")
13        if(len(data) == 2):
14            inputpin = data[0]
15            value = data[1]
16            print("Entrada pín analógica {} tiene el valor {}".format(inputpin, value))
17
18            inputPin = inputPin + 1
19            if(inputPin > 5):
20                inputPin = 0
21                ser.write(str(inputPin).encode())
22
23        except KeyboardInterrupt:
24            break
Entrada pín analógica 1 tiene el valor 330
Entrada pín analógica 1 tiene el valor 463
Entrada pín analógica 1 tiene el valor 329
Entrada pín analógica 1 tiene el valor 479
Entrada pín analógica 1 tiene el valor 327
Entrada pín analógica 1 tiene el valor 410
```

Foto de la ejecución del programa transfiriendo datos.

## **BIBLIOGRAFÍA:**

<https://www.arduino.cc/>

<https://www.raspberrypi.org/>

<https://www.python.org/>

Curso IoT Connecting Things de Netacad.