MÓDULO 5: "Cómo comunicar Arduino con Python"

CURSO PROGRAMACIÓN DE PLACAS ROBÓTICAS

Introducción

- En este proyecto veremos **cómo conectar Arduino con Python y la librería PySerial**, para emplearlo en nuestros proyectos de electrónica, robótica e IoT.
- Lo primero que necesitamos es tener instalado Python en nuestro dispositivo. Si aún no te has iniciado con Python puedes consultar la entrada Nuestro primer programa en Python donde vimos cómo instalar Python en Windows y Linux, y unos ejemplos básicos para introducir su uso.
- Una vez que tengamos Python instalado para poder comunicarnos con Arduino necesitamos la librería PySerial, que nos permite emplear de forma sencilla el puerto serie. La librería PySerial está disponible en este enlace https://github.com/pyserial/pyserial
- Podemos instalar la librería PySerial directamente desde Python, escribiendo el siguiente comando desde una consola.

```
python -m pip install PySerial
```

Ejemplos de código

- **Recibir información desde Arduino**
  - En este primer ejemplo vamos a leer información enviada por Arduino, y capturada y mostrada en pantalla por Python.
  - Para ello empezamos cargamos en Arduino el siguiente sketch, que simplemente envía continuamente una vez por segundo el texto “Hola Mundo”.

```
1  void setup() {
2  Serial.begin(9600);
3  }
4
5  void loop() {
6  Serial.println("Hola mundo");
7  delay(1000);
8  }
```

Ejemplos de código

- Dejamos el sketch funcionando en Arduino, y vamos a realizar el script en Python. Creamos un nuevo archivo de texto vacio, que guardamos con el nombre “read.py”. En su interior copiamos el siguiente código.

```
1 import serial, time
2 arduino = serial.Serial('COM4', 9600)
3 time.sleep(2)
4 rawString = arduino.readline()
5 print(rawString)
6 arduino.close()
```

- Lo que hacemos es **importar la librería Serial (PySerial) e instanciar un objeto PySerial**, que hemos llamado “arduino”. En el constructor del objeto Serial pasamos los parámetros del puerto serie que estemos empleado.

Ejemplos de código

- A continuación, **empleamos la orden "readline()"** del objeto Serial para leer una línea enviada por Arduino. Mostramos la línea en pantalla mediante la orden "Print()"
- Finalmente **mediante la orden "close()" cerramos el puerto serie**.
- Como vemos, emplear el puerto serie con PySerial es realmente sencillo. Lo único que puede parecer extraño es por qué hemos tenido que importar la librería "time".
- El motivo es que desde que creamos el objeto Serial hasta realmente está disponible para ser usado, se necesita un cierto tiempo para abrir el puerto serie. Por tanto, tenemos que introducir una espera mediante la función "Sleep", que pertenece a la librería "time".

Enviar información a Arduino

- En este segundo ejemplo vamos a enviar datos a Arduino desde Python. Para ello vamos a usar el siguiente sketch que vimos en la entrada Comunicación de Arduino con puerto serie.
- Este sketch recibe un número desde 1 a 9 y hace parpadear el LED integrado, conectado al PIN13, el número de veces recibido. Cargamos el sketch en Arduino, e igual que antes lo dejamos funcionado.
- En primer lugar importamos la librería PySerial y creamos un nuevo objeto de tipo Serial, indicando los valores del puerto serie que estemos empleando.
- En esta ocasión, **escribimos el valor mediante la función "write"**. La función "Write" envía bytes, por lo que **es necesario convertir el valor a bytes** antecediendo una b al valor enviado.
- Por último, cerramos el puerto serie con la función "close()".

Enviar información a Arduino

- En este segundo ejemplo vamos a enviar datos a Arduino desde Python. Para ello vamos a usar el siguiente sketch que vimos en la entrada Comunicación de Arduino con puerto serie.
- Este sketch recibe un número desde 1 a 9 y hace parpadear el LED integrado, conectado al PIN13, el número de veces recibido. Cargamos el sketch en Arduino, e igual que antes lo dejamos funcionado.
- En primer lugar importamos la librería PySerial y creamos un nuevo objeto de tipo Serial, indicando los valores del puerto serie que estemos empleando.
- En esta ocasión, **escribimos el valor mediante la función "write"**. La función "Write" envía bytes, por lo que **es necesario convertir el valor a bytes** antecediendo una b al valor enviado.
- Por último, cerramos el puerto serie con la función "close()".

Pyfirmata

1. Descargar e instalar el software de Arduino, según el sistema operativo de la computadora anfitriona. Al momento de configurar el IDE, hay que anotar el **puerto serie** (*serial port*) al que se conecta el Arduino, ya que más adelante haremos referencia a éste. El puerto serie puede ser algo así como COM3
2. Descargar Firmata en el Arduino. En el IDE de Arduino seleccionar File del menú principal. De ahí seleccionar: Examples/Firmata/StandardFirmata.
3. Descargar e instalar la biblioteca pyFirmata en la computadora anfitriona. Desde una terminal (posiblemente con privilegios de administrador), teclear:

```
pip install pyfirmata
```

Pyfirmata

- Para comenzar a usar pyFirmata debemos importar el paquete correspondiente al inicio de nuestro programa:

```
import pyfirmata
```

- Podemos determinar qué versión de pyFirmata estamos usando:

```
print('pyFirmata version:', pyfirmata.__version__)
```

- A continuación creamos un objeto que representa nuestra placa (*board*) de Arduino. Para ello necesitamos indicar como cadena de caracteres el puerto serie que anotamos durante la configuración del IDE ('/dev/ttyACM0' en mi caso, ya que estoy usando Linux):

```
placa = pyfirmata.Arduino('COM3')
```

Pyfirmata

- Con el objeto referido por la variable placa podemos determinar la versión de Firmata que está usando nuestro Arduino. El método get_firmata_version() devuelve una tupla con dos valores: la versión mayor y la versión menor. Podemos utilizar una *asignación paralela* para extraer los valores de la tupla en dos variables:

```
v_mayor, v_menor = placa.get_firmata_version()
print('Firmata version mayor:', v_mayor)
print('Firmata version menor:', v_menor)
```

Pyfirmata

- El método **get_pin()** sirve para activar un pin en el Arduino, configurándolo según la cadena que se le envía como argumento. Dicha cadena está compuesta de a o d (para indicar que el pin es analógico o digital), el número de pin, y el modo (i para entrada, o para salida, p para PWM).

- Estos tres elementos se separan entre sí usando un carácter de dos punto (:). Por ejemplo d:12:o indica que el pin 12 es una salida digital. Este es un ejemplo de cómo se utiliza el método:

```
salida = placa.get_pin('d:12:o')
```

Pyfirmata

- Después del código de arriba, es necesario invocar el método enable_reporting() sobre el objeto que representa el pin de entrada para poder leer las señales recibidas:

```
entrada.enable_reporting()
```

- Si se utilizan uno o más pines de entrada es necesario crear un *thread* iterador, de lo contrario la placa puede seguir enviando datos por la conexión serial hasta producir un desbordamiento. Para crear dicho *thread* usamos la siguiente instrucción:

```
pyfirmata.util.Iterator(placa).start()
```

Pyfirmata

- Para escribir a un pin de salida digital, usamos el método write() enviando como argumento False (0V) o True (5V), según la cantidad de voltaje que deseemos producir como salida. Ejemplo:

```
salida.write(True)
```

- En lugar de False y True se puede usar 0 y 1, respectivamente.

- Usamos el método read() para leer de un pin de entrada digital. Este método devuelve 0 si el voltaje de entrada es menor a 2.5V, o 1 si es mayor a 2.5V. Por ejemplo:

```
x = entrada.read()
```

Pyfirmata

- Posteriormente, podemos usar la variable x en una instrucción condicional (if) para determinar qué valor efectivamente se leyó.

- Para pausar nuestro programa por una cierta cantidad de tiempo se recomienda usar el método pass_time() sobre el objeto que representa nuestra placa de Arduino. Se le envía como argumento el número de segundos que deseamos que dure la pausa. La siguiente porción de código detiene el programa en curso durante medio segundo:

```
placa.pass_time(0.5)
```

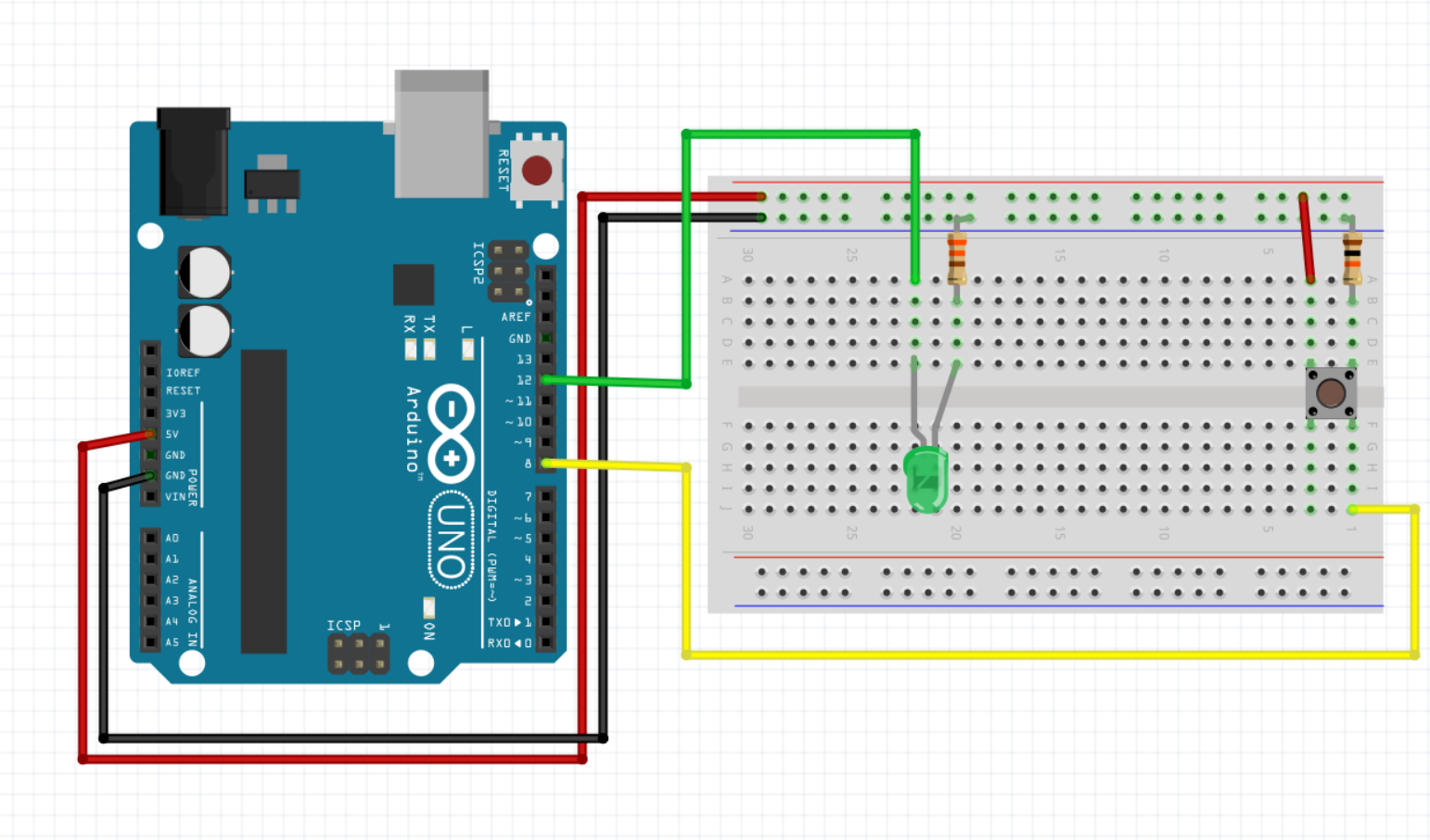
- Por último, para terminar nuestro programa de manera limpia, se debe invocar el método exit() así:

```
placa.exit()
```

Un ejemplo completo

- Vamos a usar un botón para controlar el prendido y apagado de un LED
- Además del Arduino Uno conectado vía USB a la computadora anfitriona, vamos a requerir el siguiente hardware:
  - Un protoboard.
  - Un botón o pulsador (*push button*).
  - Un LED (diodo emisor de luz).
  - Una resistencia de 330Ω (bandas naranja, naranja, café).
  - Una resistencia de 10KΩ (bandas café, negro, naranja).
  - Cinco cables conectores.

Un ejemplo completo



Un ejemplo completo

- Vamos a usar un botón para controlar el prendido y apagado de un LED
- Además del Arduino Uno conectado vía USB a la computadora anfitriona, vamos a requerir el siguiente hardware:
  - Un protoboard.
  - Un botón o pulsador (*push button*).
  - Un LED (diodo emisor de luz).
  - Una resistencia de 330Ω (bandas naranja, naranja, café).
  - Una resistencia de 10KΩ (bandas café, negro, naranja).
  - Cinco cables conectores.