mediaLab Documentation

Publicación 1

Johnny Cubides, Carolina Pulido

ene. 06, 2017

Índice general

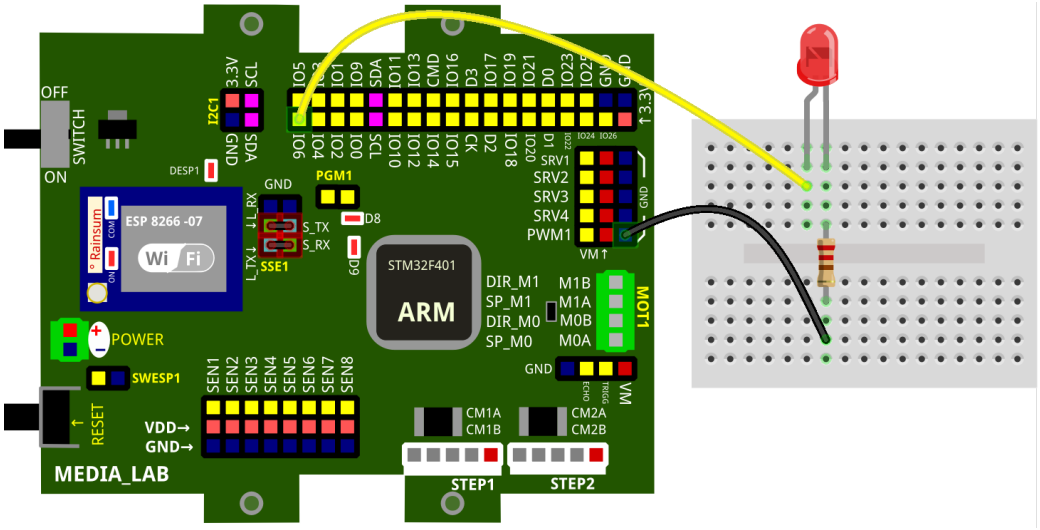
Contents:

Referencia Rápida

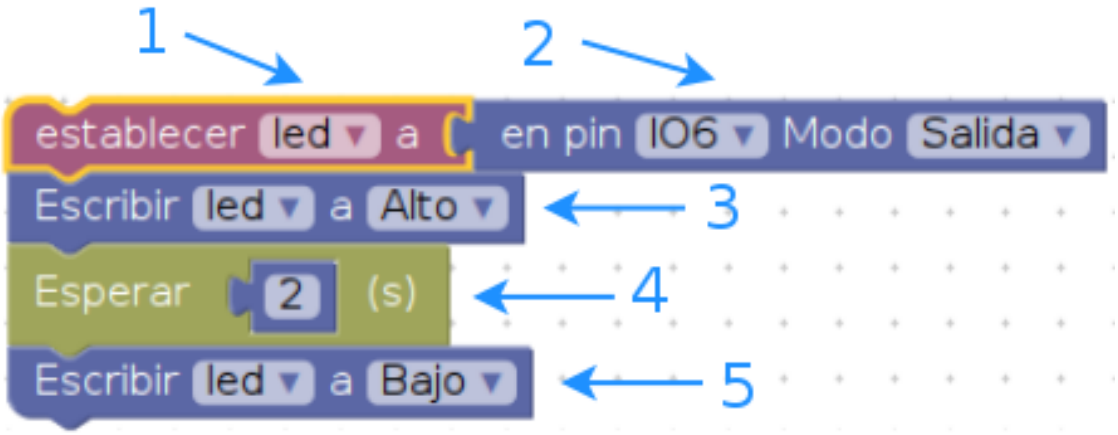
1.1 SALIDAS DIGITALES

En éste ejemplo un led será encendido durante un par de segundos para luego ser apagado.

1.1.1 Conexión física



1.1.2 Programa en blockly



Explicación de programa

En la imagen se numeran los bloques para luego ser explicados como sigue:

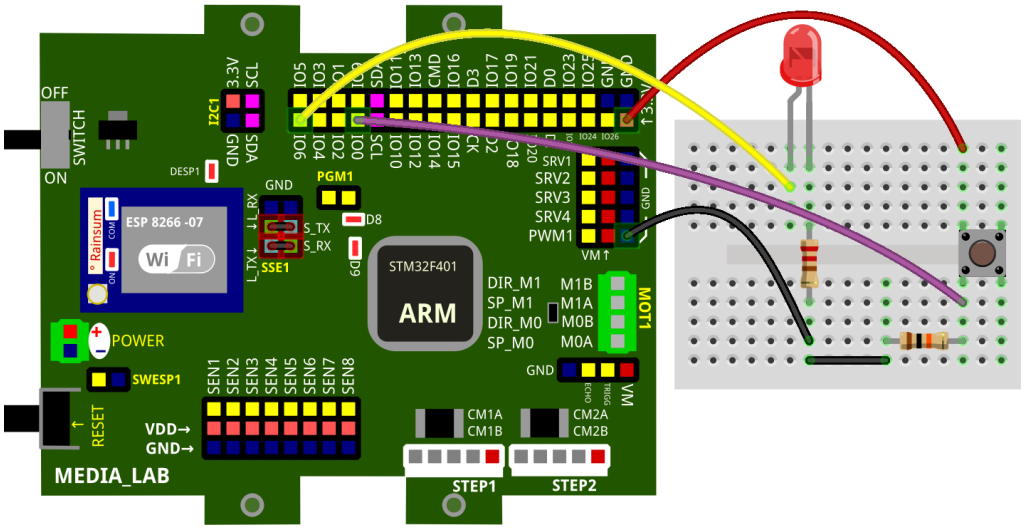
1. Los pines digitales son manejados como objetos, así que debe ser iniciado como una variable con un identificador (nombre), se sugiere un nombre explícito como en éste caso (led).
2. Aquí se ubica el nombre del pin correspondiente a la conexión física en la placa, además, permite definir el pin como una entrada o como una salida digital. (Compare la imagen de la conexión física con la del código blockly, notará que ambas se refieren al pin IO6).
3. Ya definido el pin (led) es hora de darle un estado, en éste caso es alto. Cuando se refiere a ALTO quiere decir que envía un 1 lógico a el pin (led), en lógica positiva indica que la presencia de energía estimula a el led a encenderse.
4. La permanencia en un estado depende de las transiciones, en éste caso pasar de un estado a otro estado depende del tiempo, éste bloque permite que se permanezca en un estado por determinados segundos.
5. Siendo éste el mismo bloque que está en **3** se ha indicado que pondrá el led en bajo, es decir, el led será apagado, terminando así con el programa.

1.2 ENTRADAS DIGITALES

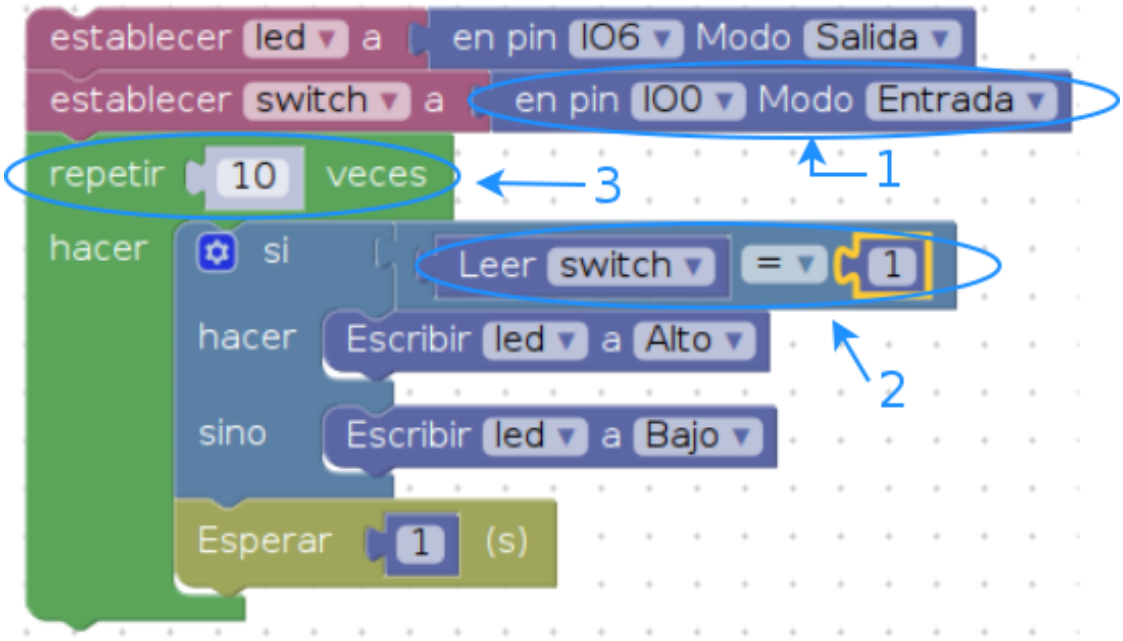
En éste ejemplo un switch (como entrada digital) controlará el estado de un led. Cuando el switch es oprimido el led es encendido, en caso contrario el led permanece apagado. Éste algoritmo será ejecutado únicamente diez veces, para luego terminar con el programa.

Nota: Se recomienda hacer ésta prueba con un bloque de secuencia finita, ya que con uno infinito, µpython no podrá ejecutar otra tarea hasta no recibir un reset.

1.2.1 Conexión física



1.2.2 Programa en blockly



Explicación de programa

En la imagen se numeran los bloques relevantes, los demás fueron explicados en la sección SALIDAS DIGITALES; el bloque **si... hacer... sino** (en azul) será explicado de forma detallada en otra sección.

1. Este bloque denominado modo del pin permite construir el objeto **switch** en el pin **IO0** (ver también imagen *conexión física*) con características de entrada. Por tratarse de un pin digital solo tendrá dos posibles estados de lectura, **1** o **0**.

2. Este par de bloques permiten leer, retornar el estado del pin (switch) y comparar su estado. El valor retornado será 1 si se detecta presencia de tensión con un umbral de nivel positivo, de lo contrario, el valor será 0. Con la ayuda del bloque **si... hacer... sino**, se determina el estado que tendrá el led. El estado que tenga el led durará el valor asignado en el bloque de espera.
3. El bloque repetir, asigna un límite finito de repeticiones del algoritmo antes de terminar con el programa.

1.3 ENTRADAS ANÁLOGAS

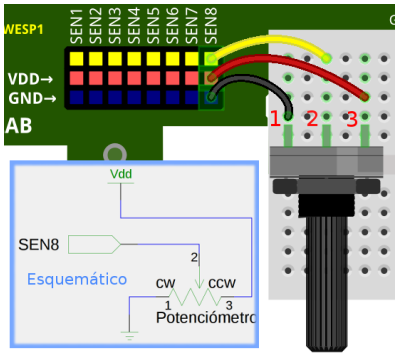
La información análoga tiene un comportamiento continuo, cuando se discretiza la información (digitalizar) la resolución con la que se logra digitalización dependerá de la capacidad de observación del dispositivo.

media_lab tiene la capacidad de digitalizar el nivel de tensión por 8 canales diferentes (ver conexión física). Se reitera que la conversión lograda es de tensión (voltaje). La resolución que se pude lograr con *media_lab* es de 12 bits. Los umbrales de tensión que pude observar *media_lab* son: 0 a 3.3 Voltios. Si se desea medir niveles de tensión diferentes se debe usar un sistema de adaptación.

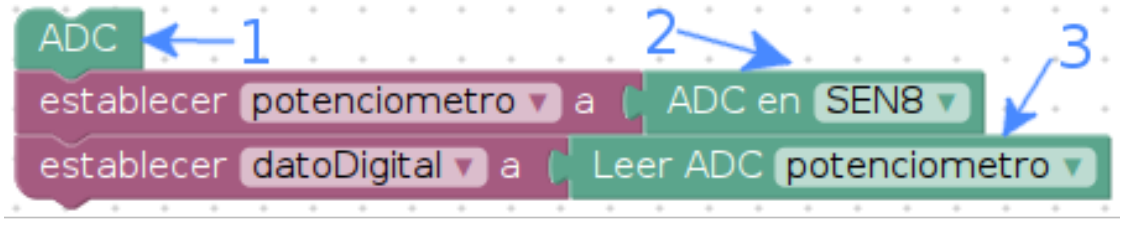
Ejemplo: Si al conectar a una de las entradas un cable que contiene una tensión variable de un potenciómetro como en la imagen de conexión física y la lectura obtenida fue 2800, la interpretación se hace como sigue:

> Resolución = 12 bits -> 2^12 = 4096 posibilidades -> puede medir desde **0** a **4095**
>
> Tensión = 2800*3.3 Voltios / 4095 = 2.256 Voltios.

1.3.1 Conexión física



1.3.2 Programa en blockly



Explicación de programa

En la imagen se numeran los bloques para luego ser explicados como sigue:

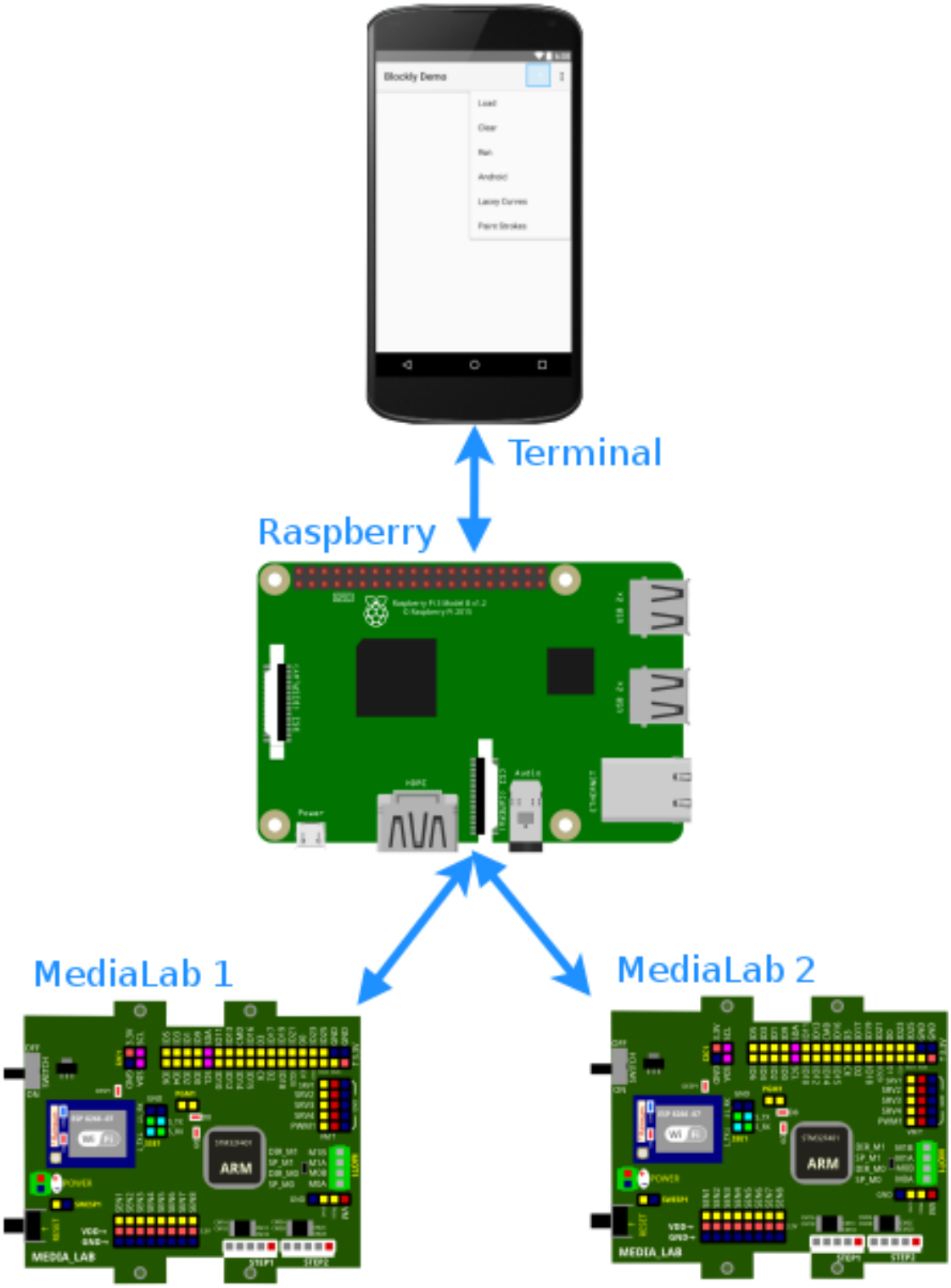
1. El bloque **ADC** activa la librería que permite crear el conversor análogo digital.
2. Como en la construcción de objetos digitales, en los objetos análogos se requiere hacer uso de variables para así hacer uso de sus propiedades. Éste bloque construye objetos ADC con capacidad de leer en el pin con el nombre correspondiente (en éste caso *SEN8*, ver conexión física).
3. La lectura análoga puede ser almacenada en otra variable para su respectivo uso, éste bloque extrae la lectura análoga, no olvidar que es de tipo entero, con una resolución ya definida.

1.4 MQTT

1.4.1 Concepto

MQTT en sí es un protocolo de comunicación **publicación/suscripción**, se trata de mensajes clasificados según **asuntos** (*topics*) los cuales deben ser gestionados por un servidor llamado **Broker**, los elementos conectados a éste servicio podrán publicar mensajes con el respectivo asunto, otros dispositivos conectados al servicio podrán suscribirse a mensajes con asuntos publicados; así, mensajes publicados serán enviados a los suscriptores por parte del Broker que hará de corredor "cartero".

Conexión física



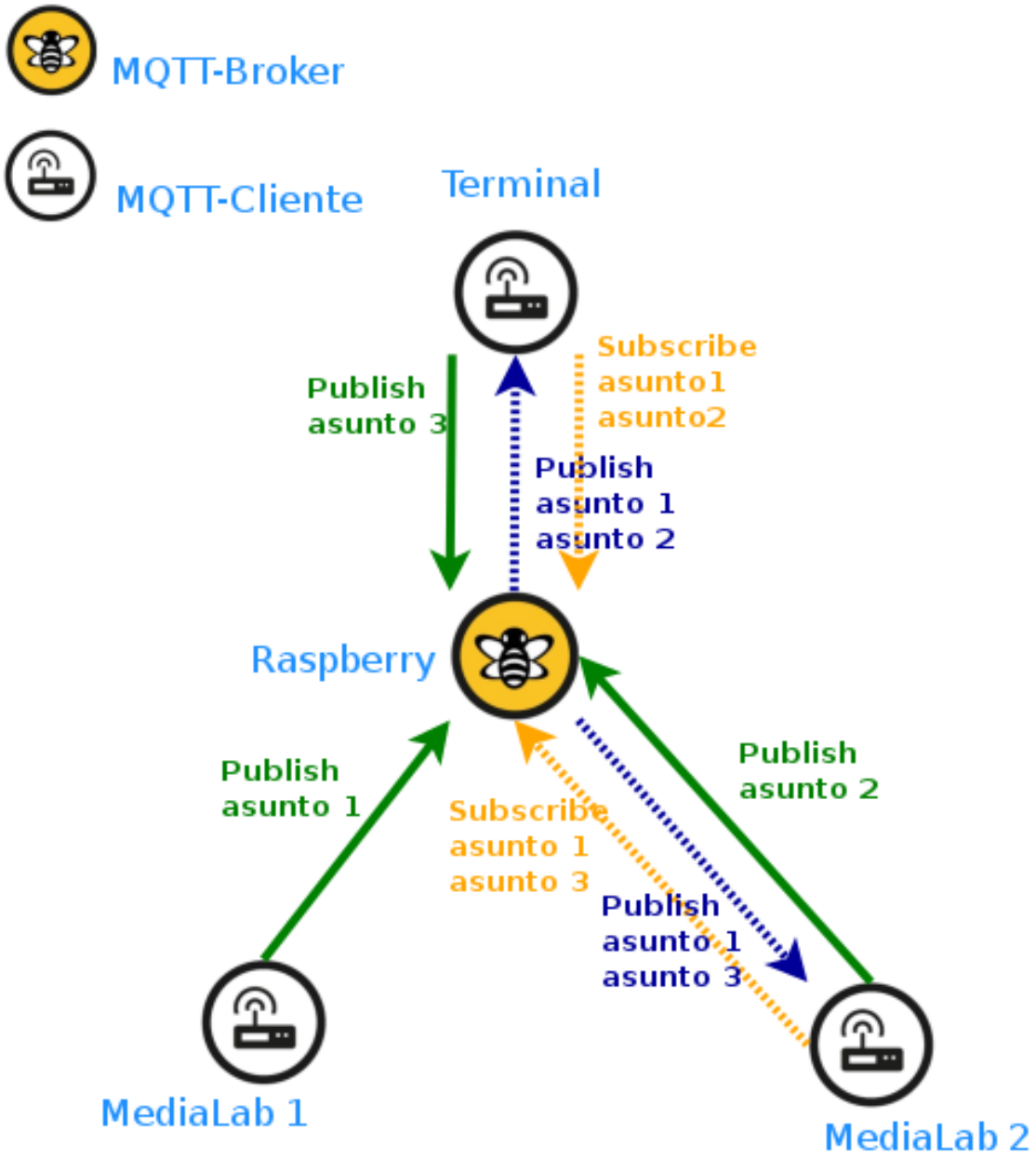
MQTT (Message Queue Telemetry Transport) funciona sobre TCP/IP, así que se requiere un canal con esas características. En la imagen se pueden observar un ejemplo de infraestructura, en él hay cuatro elementos conectados entre sí: dos media_lab, un celular que hace de interfaz de usuario (terminal) y una raspberry pi; la conexión es lograda a partir

de una comunicación wifi.

La raspberry pi es la responsable de crear el punto de acceso wifi y además de tener acceso a Internet por medio de su puerto de Ethernet. Los demás dispositivos se conectarán como estaciones a esta pequeña red con el fin de compartir en el canal información como lo es estados y ordenes a partir de mensajes.

Comportamiento

De la imagen de *conexió física* se puede describir la infraestructura necesaria para hacer uso del protocolo MQTT. Tanto en el celular (terminal) como en las dos tarjetas media_lab se debe contener software **MQTT-cliente** el cual permita hacer la solicitudes de suscripción o publicación de mensajes; en las tarjetas *media_lab* es logrado a partir de las librerías pre-instaladas, en el celular el cliente debe ser instalado, si el celular es Android como en el ejemplo, basta con instalar un cliente MQTT, como es el caso de **MyMQTT** desde *Play Store*; la raspberry tiene la responsabilidad de alojar el **MQTT-broker**, para el ejemplo, el broker es el popular software denominado **mosquitto**, el cual ya es pre-configurado para raspbian-lite.



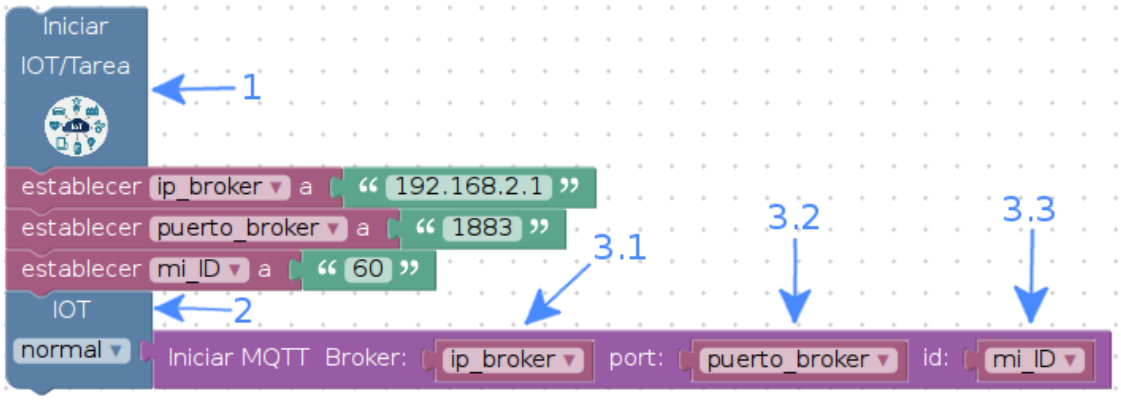
Ahora se explicará como los elementos interactúan a partir del flujo de mensajes; la explicación será realizada a partir de un ejemplo:

1. El servicio MQTT-broker debe ser iniciado en la raspberry; **mosquitto** se inicia en la raspberry, escuchará por el puerto 1883 y estará listo para gestionar la mensajería según publicaciones, suscripciones y asuntos (*topic*).
2. Los demás dispositivos como tarjetas *media_lab* y celulares deben conectarse a la dirección del servicio (dirección IP de la raspberry) especificando el puerto por el cual **mosquitto** escucha (por defecto es 1883).
3. Los dispositivos que desean **suscribirse** a la mensajería, deben hacerlo a el **MQTT-broker** especificando los asuntos que sean de interés; en la imagen, el celular solicita al broker suscribirse al *asunto 1* y al *asunto 2*, *MediaLab 2* solicita suscribirse al *asunto 1* y al *asunto 3*
4. Cualquier dispositivo conectado al servicio puede **publicar** mensajes con el asunto que desee, el broker se encargará de gestionar el flujo y hacer llegar a los dispositivos suscritos el mensaje correspondiente. Así entonces, el celular recibe información de ambas tarjetas ya que ambas publican asuntos a los cuales el celular está suscrito,

del mismo modo, MediaLab 2 está suscrito a publicaciones de MediaLab 1 con el asusto 1, Además, MediaLab 2 está suscrita a mensajes provenientes del celular.

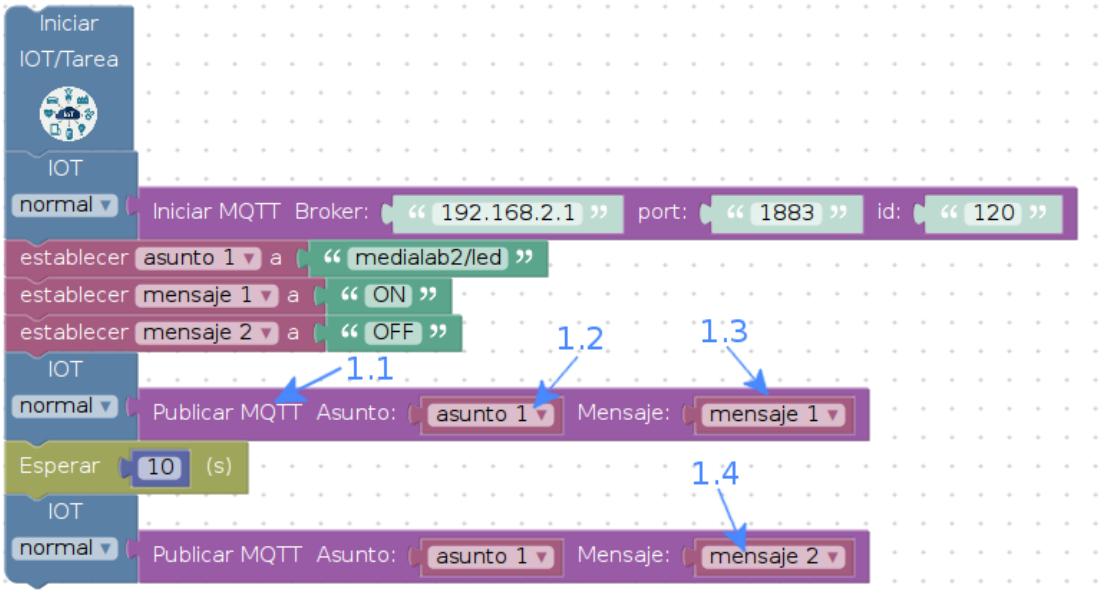
Se puede pensar entonces, que los dispositivos tendrán entre ellos un **"chat"** el cual permite que **"las cosas"** hablen entre ellas. Si pensamos un poco más afuera de lo que se ha mencionado, se puede pensar que una máquina le puede dar una orden a otra para que cambie su estado y entonces los mensajes no solo serán informativos, sino tomas de decisiones en sistemas máquina-máquina donde poco o nada habrá intervención del hombre.

1.4.2 Iniciar clientes MQTT



1. Todas las tareas que requieran de IOT necesitan de éste bloque, éste bloque debe ser iniciado una única vez por programa.
2. El bloque IOT es responsable de ejecutar tareas de Internet, además, de otras configuraciones. Adicional, éste bloque tiene una pestaña de control de velocidad la cual controla y sincroniza la comunicación entre el esp y el stm, las razones de cambiar la velocidad serán tenidas en cuenta por el programador según él note el rendimiento.
3. El bloque **Iniciar MQTT** es responsable de hacer el enlace con el Broker, se requiere conocer del Broker la dirección IP **3.1**, el puerto por donde escucha **3.2**, un identificador del dispositivo que solicita la conexión **3.3**. Nótese en la imagen que el bloque luego de estar diligenciado es usado por IOT **2** para pasarle los parámetros al esp responsable de la comunicación Wifi.

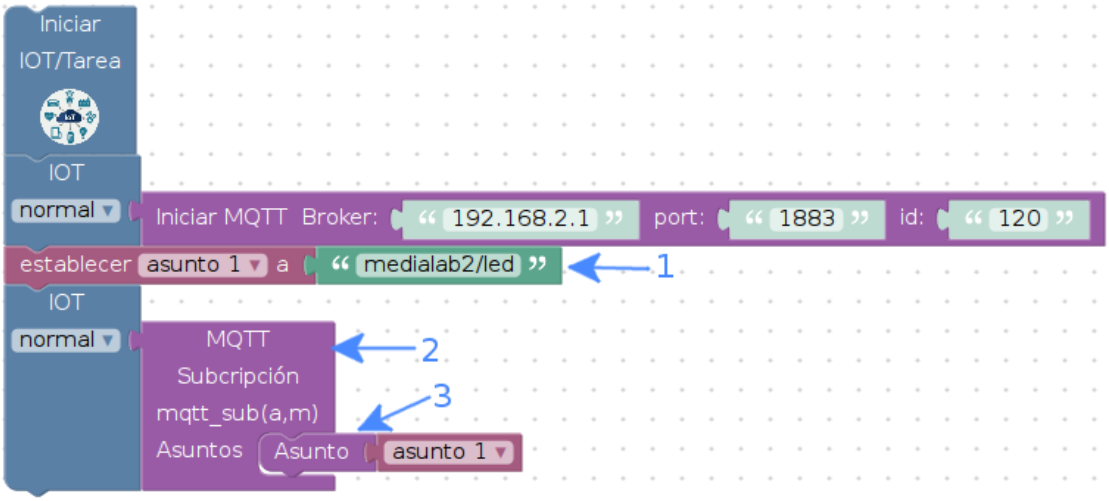
1.4.3 MQTT Publicar



1. Las publicaciones no requieren tantos bloques como lo requiere una suscripción. En éste ejemplo se publicará mensajes con el asusto *medialab2/led* instintivamente el asunto indica que medialab2 tiene un led, al notar en el bloque **Publicar MQTT 1.1** el asunto **1.2** será el mismo para las dos publicaciones, el cambio estará en el mensaje **1.3** que inicialmente pide que el led de medialab2 esté en su estado encendido **ON**, en la segunda publicación **1.4** se solicita apagar el led **OFF**.

1.4.4 MQTT Suscribir

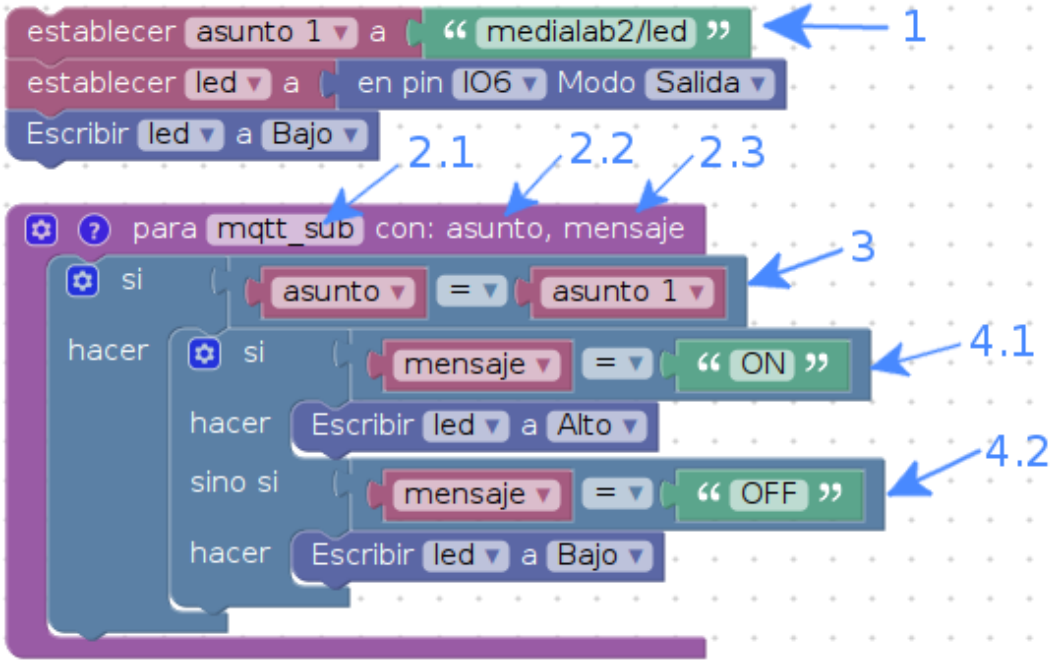
Una suscripción requiere más bloques que una publicación pero no por eso es más compleja.



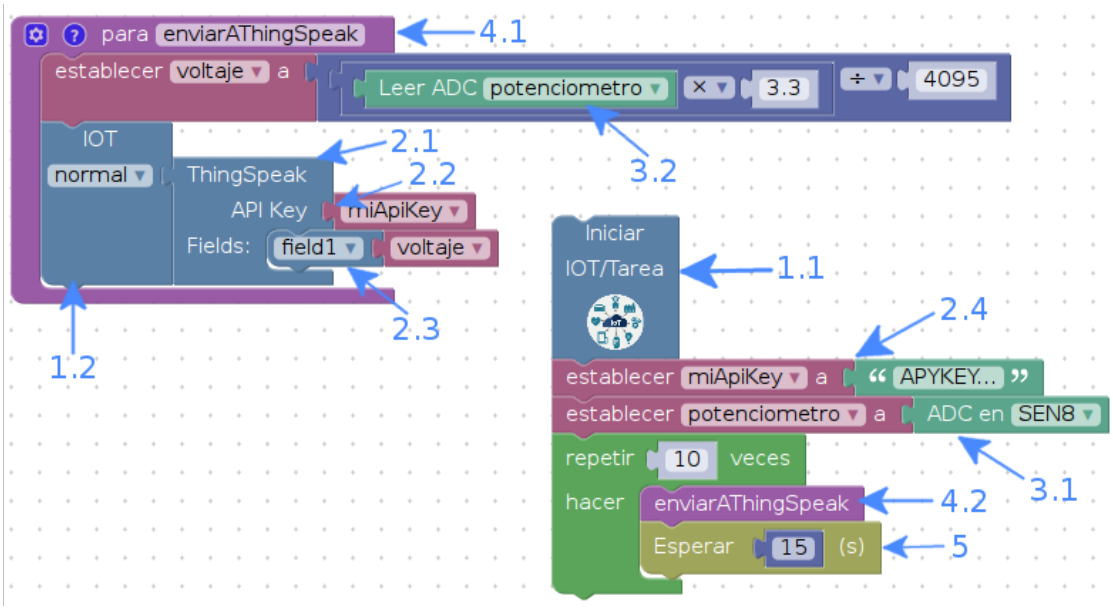
1. Como en la publicación, la suscripción requiere el asunto, para completar el ejemplo, se suscribirá a **medialab2** para atender las ordenes que puedan venir de otros dispositivos como es el caso de **medialab1**.
2. Éste bloque permite que medialab2 se suscriba a diferentes asuntos los cuales serán usados por el bloque **IOT** capaz de comunicar al esp la tarea a ejecutar. Nótese también que éste bloque contiene escrito en su cuerpo

mqtt_sub(a,m) lo cual en el lenguaje de *Python* es una función. Se explicará a continuación el porqué: *medialab2* está suscrito a mensajes y los recibirá cuando sean publicados, pero para que no solo sea información, sino que se convierta en algo útil para tomar decisiones, **esp** llamará la función **mqtt_sub(asunto,mensaje)** del stm y adicionalmente le entregará la publicación recibida para que tome decisiones a partir del asunto y del mensaje.

3. Éste bloque permite que el dispositivo se suscriba a un asunto, se puden llamar varios asuntos para suscribirse, por ahora se puede suscribir a tres diferentes asuntos.



1.5 ThingSpeak



Guia de Uso

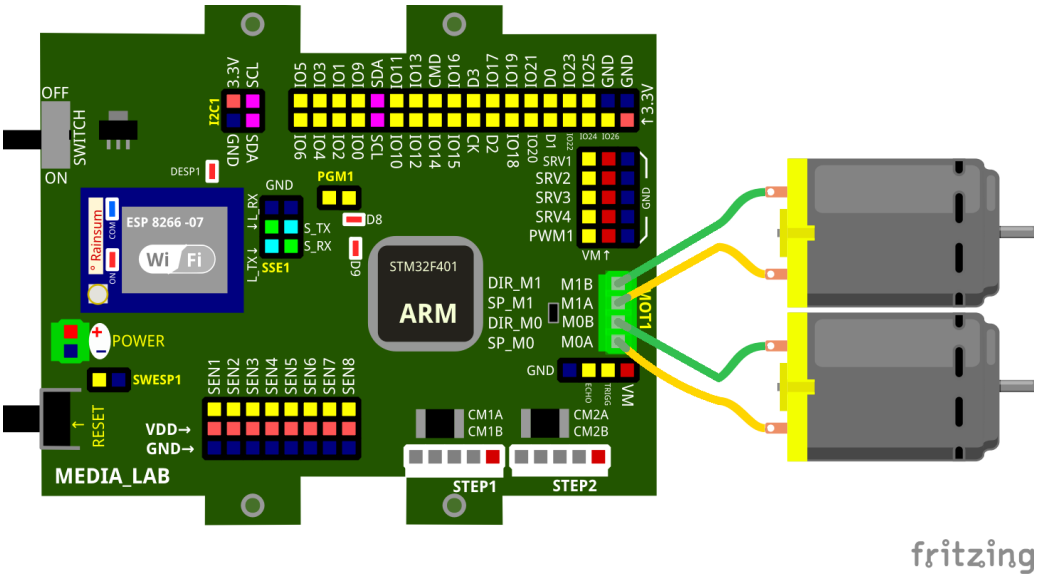
2.1 Hola Mundo

```
import pyb

def blink():
    LED = pyb.LED(1)
    LED.toggle()
    return

blink()
```

2.2 Manejo de Motores

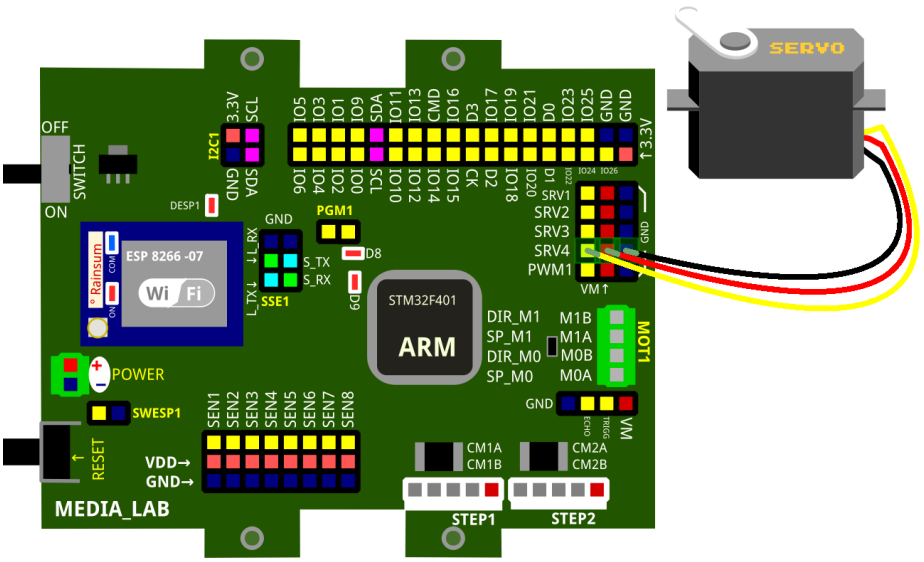



```
from pyb import Timer, Pin

class mobile:
    pl = 100
    pr = 100
```

```
    def __init__(self):
        timerMotors = Timer(1, freq=1000)
        self.lPWM = timerMotors.channel(2, Timer.PWM, pin=Pin('SP_M0'))
        self.rPWM = timerMotors.channel(4, Timer.PWM, pin=Pin('SP_M1'))
        self.ld = Pin('DIR_M0', mode=Pin.OUT)
        self.rd = Pin('DIR_M1', mode=Pin.OUT)
    def calibration(self, l, r):
        self.pl = l
        self.pd = r
    def run(self, l, r):
        self.lPWM.pulse_width_percent(int(abs(l)*self.pl/100))
        self.rPWM.pulse_width_percent(int(abs(r)*self.pr/100))
        if l < 0:
            self.ld.low()
        else:
            self.ld.high()
        if r < 0:
            self.rd.low()
        else:
            self.rd.high()
```

2.3 Manejo de ServoMotores




```
import pyb
class servomotor:
def __init__(self, pin):
    timer = pyb.Timer(4, freq=200)
    self.ch = timer.channel(5-pin, pyb.Timer.PWM, pin=pyb.Pin('SRV'+str(pin)))
def angulo(self,x):
    if 0 <= x and x <= 180:
        self.ch.pulse_width(4800+int(x*15600/180))
```

CAPÍTULO 3

Indices and tables

- genindex
- modindex
- search