Tomas De Angelis - 2020

TALLER DE TECNOLOGIAS CHEAT SHEET

COMANDOS BASICOS IDE ARDUINO:

Pines:

- **pinMode(nroPin , configuración);** //Configuración→ INPUT o OUTPUT
- **digitalWrite(nroPin , estadoDeseado);** //Estado Deseado→ HIGH (5V), LOW (0V)
- **digitalRead(nroPin);** → Devuelve High (>2,5V) o Low (<2,5V)
- **analogWrite(nroPin, estadoDeseado);** //Estado Deseado→ Rango entre 0 y 255
- **analogRead(nroPin);** →Devuelve valor entre 0 y 1023

Puerto Serial:

- **Serial.begin(baudRate);** → Inicializa el puerto serial. El baud rate es la velocidad de transmisión, un valor muy utilizados es 9600 Baudios por segundo. Este comando se utiliza una vez dentro del setup del programa.
- **Serial.println(mensaje);** → Para enviar un mensaje por serial y hacer un salto de línea:

INFORMACION TEORICA:

Microcontrolador: Un microcontrolador es un encapsulado que ya contiene en su interior todo el sistema necesario para interactuar con el mundo exterior. Por lo general cumplen sólo una tarea, es decir que ejecutan un único programa y el microprocesador que posee en su interior es de mucho menor poder que el de una computadora y a su vez son mucho más económicos.

Microprocesador: Integrado digital que entiende y ejecuta una secuencia de instrucciones (programa). Contiene circuitos que llevan a cabo operaciones aritméticas , lógicas y de control. Los microprocesadores no funcionan sólos, son una parte de un sistema que posee otros componentes. El microprocesador es la parte inteligente del sistema.

Señales analógicas: Las señales analógicas son aquellas que son representables por una función matemática continua en el tiempo, su amplitud varía en el tiempo y puede adoptar un número infinito de valores. En particular, las señales analógicas de Arduino van de 0 a 1023 y son enviadas o recibidas por los pines analógicos (A0 a A5).

Señales digitales: Las señales digitales son aquellas que solo pueden adoptar un número finito de valores. En el caso de la electrónica que utilizaremos, emplearemos principalmente señales digitales binarias, las cuáles solo pueden tomar dos valores: 1 (HIGH o Alto) y 0 (LOW o bajo). Las señales digitales en Arduino son enviadas o recibidas por los pines digitales (0 a 13).

PINES EN LOS ARDUINOS

Pines Digitales: Los pines digitales en una Arduino son aquellos que se utilizan solamente para recibir o emitir una señal digital eléctrica dependiendo si se lo configura como entrada o salida respectivamente.

En la mayoría de las placas Arduino(como en la Arduino Uno que utilizaremos), los pines digitales configurados como salida sólo emiten una señal digital eléctrica que puede ser de 0V o 5 V. Cuando son configurados como entrada, la mayoría pueden recibir señales que como LOW utilizan valores de 0 V y como HIGH de 5 V o 3,3 V.

Los pines digitales en Arduino están identificados por la palabra DIGITAL y van del 0 al 13.

Pines Analógicos: La función principal de los pines analógicos es recibir señales analógicas entre 0 V y 5 V para convertirlas y devolverlas como valores numéricos. Cada conversión devuelve un número entre 0 y 1023 siguiendo la siguiente formula:

$$V_{SEÑAL} = Resultado\ Conversión \times \frac{5\,V}{1023}$$

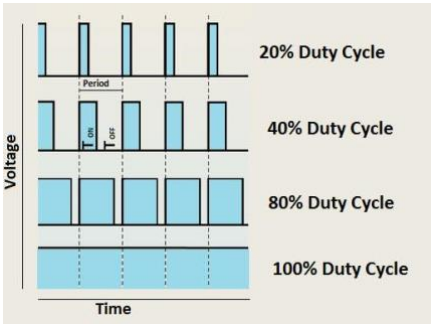
También pueden usarse pines analógicos para enviar señales analógicas con un rango de 0 a 255. El numero enviado se mapea a los voltios que mandará el pin mediante la siguiente regla de 3: Ej.: analogWrite(led, 144);

$$255\_\_\_\_\_5(V)$$
$$144\_\_\_\_\_x(V)$$

Los pines analógicos en Arduino están identificados por la palabra ANALOG (del A0 al A5).

Pines con PWM: Recordar que también es posible leer y escribir señales analógicas con ciertos pines digitales de Arduino que admiten PWM (Pulse Width Modulation). Los pines que presentan PWM son claramente indicados mediante el símbolo de "~" en la placa.

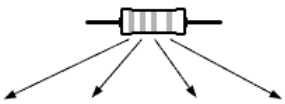
El concepto de PWM se basa en la variación del tiempo de trabajo para generar la ilusión de que por determinado pin digital se están escribiendo valores de voltaje diferentes a 0 o 5V.



$$Ciclo\ de\ Trabajo\ = \frac{T_{ON}}{T_{ON} + T_{OFF}} \times 100\%$$

RESISTENCIAS:

Las resistencias son componentes que se utilizan con el fin de limitar la corriente que pasa por ellas y por lo tanto por el resto del circuito. No poseen polaridad. El valor de la resistencia lo vemos en los colores que tienen impresos y su orden.



Código de colores

Colores	1ª Cifra	2ª Cifra	Multiplicador	Tolerancia
Negro		0	0	
Marrón	1	1	$\times 10$	± 1%
Rojo	2	2	$\times 10^2$	± 2%
Naranja	3	3	$\times 10^3$	
Amarillo	4	4	$\times 10^4$	
Verde	5	5	$\times 10^5$	± 0.5%
Azul	6	6	$\times 10^6$	
Violeta	7	7	$\times 10^7$	
Gris	8	8	$\times 10^8$	
Blanco	9	9	$\times 10^9$	
Oro			$\times 10^{-1}$	± 5%
Plata			$\times 10^{-2}$	± 10%
Sin color				± 20%

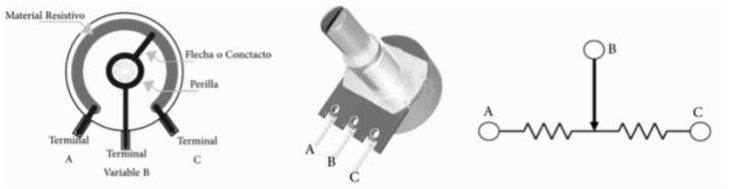
LEDS: Light Emitting Diode.

Semiconductor de 2 terminales (Ánodo y Cátodo) que convierte en luz la corriente que pasa por él. Para que funcione el voltaje positivo tiene que estar aplicado al ánodo y el cátodo estar conectado a tierra cerrando así el circuito.

Por lo gral utilizamos resistencias de aprox. 200 Ω, pero pueden calcularse particularmente para cada led siguiendo la ley de ohm y teniendo en cuenta la intensided máxima que maneja y el voltaje que circula por el circuito.

Ley de OHM: V= R.I

POTENCIOMETRO:



Un potenciómetro es básicamente una resistencia variable. Consta de tres terminales. Entre las terminales A y C tenemos una resistencia siempre constante, el valor mas común para esa resistencia es de 10 KΩ.

Entre las terminales A y B tenemos una resistencia variable (R_{AB}) y entre las terminales B y C hay otra (R_{BC}). $R_{AB} + R_{BC} = R_{\text{AC}}$ (RTotal).

Al mover la perilla en sentido horario, el contacto se acerca hacia el terminal C, haciendo que R_{AB} aumente y R_{BC} disminuya. Hasta que la perilla hace tope cuando el contacto llega al terminal C, en ese momento.

R_{AB} =RTotal y R_{BC} =0.

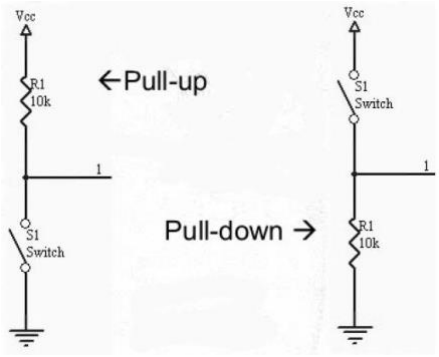
Se puede seguir un razonamiento análogo al mover la perilla hacia el otro lado.

Si conectamos 5 V entre las terminales A y C y medimos el voltaje en la terminal B, este variará al mover la perilla, tomando valores de 0 V y 5 V en los extremos mientras que en los otros puntos mediremos valores intermedios entre 0 y 5.

RESISTENCIAS PULLUP Y PULLDOWN

Estas resistencias no son resistencias especiales, simplemente son resistencias que se colocan en lugares particulares de un circuito para generar determinado comportamiento.

Son utilizadas cuando en cierto circuito algún pin declarado como entrada está al aire (no conectado a nada). Cuando esto ocurre, como los pines de entrada tienen una muy alta resistencia de entrada, se detecta ruido al medir la entrada y no es posible determinar un estado concreto del pin.



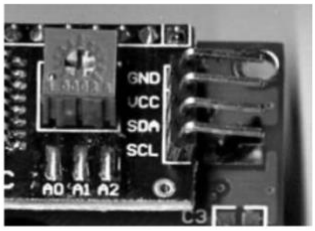
Como la imagen muestra, un extremo de la resistencia se conecta al pin que está al aire y el otro a 5 V si es PULL-UP o a 0 V (GND) si es PULL-DOWN.

Generalmente se utilizan resistencias de 10 KΩ.

PANTALLA LCD CON ADAPTADOR i^2C

I^2C: Es un módulo adaptador que utilizaremos principalmente para bajar la cantidad de cables que habría que conectar en la pantalla LCD. Nos permite conectar simplemente 4 cables en lugar de 12. Deberemos conectar: VCC (5V o 3,3V), GND, y los pines de comunicación del módulo I^2C que utilizará para comunicarse: uno para reloj (SCL) y otro para el data (SDA).

A0	A1	A2	Adress
1	1	1	0x27
0	1	1	0x26
1	0	1	0x25
0	0	1	0x24
1	1	0	0x23
0	1	0	0x22
1	0	0	0x21
0	0	0	0x20



CAMBIAR DIRECCION I^C: Se pueden conectar varios displays lcd con módulos I^2C al mismo Arduino. Para esto, debemos definir su dirección I^2C y que así el Arduino sepa a cuál comunicar cada información. Para cambiar la dirección deberemos soldar uno o más de los pines marcados como A0, A1, A2, y así formar un código diferente en cada módulo. En la imagen se muestran los 3 pines soldados (código 0,0,0).

COMANDOS BASICOS: Acordarse de incluír la librería: **#include <LiquidCrystal_I2C.h>** y después definir el contructor: `LiquidCrystal_I2C lcd = LiquidCrystal_I2C(0x27, 16, 2);`

Metodos	Función
LiquidCrystal_I2C()	Constructor, establece la configuración de la pantalla.
init()	Prepara el LCD para su uso.
clear()	Borra la pantalla.
setCursor(col,row)	Permite mover el cursor a la posición indicada.
print()	Imprime por pantalla.
backlight()	Enciende la iluminación de fondo.
createChar(num, data)	

Tomas De Angelis - 2020

SENSOR DE DISTANCIA POR ULTRASONIDO: Sensor HC-SR04

El HC SR04 Tiene 4 pines: GND y VCC, Echo y Trigg, siendo trigg el que emite la onda y echo quien la recibe.

Se calcula: Distancia = {(Tiempo entre Trig y el Echo) * (V.Sonido 340 m/s)}/2.

Como "340 m/s" es constante y "/2" es constante, se calcula y luego se pasa en unidades a microsegundos/cm, dando como resultado 0,01715 µs/cm. Nos queda:

Distancia = tiempo entre Trig y Echo * 0,01715 µs/cm. Si ahora hayamos el tiempo entre trigg y echo en µs, nos dará la distancia en cm.

Método para medir la distancia en el IDE de Arduino:

1. Bajamos el trig y esperamos 5 microsegundos para que se estabilice todo:
    a. digitalWrite(pinDelTrig, LOW);
    b. delayMicroSeconds(5);
2. Activamos el trig y esperamos el tiempo en que demora en enviar la onda:
    a. digitalWrite(pinDelTrig , HIGH);
    b. delayMicroSeconds(10);
3. Apagar el pinDelTrig para que no mande mas ondas. digitalWrite(trig, LOW);
4. Guardamos el tiempo que demora en ir y volver: int t = pulseIn(Echo, HIGH);
5. t * 0,01715 será la distancia en cm a la que se encuentra el objeto.

Podemos incluír esos 5 pasos en un método y simplemente cuando se quiera hallar determinada distancia llamar al método dameLaDistancia();. el método dameLaDistancia() retornará t.

```
long dameLaDistancia()
{
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(5);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  return pulseIn(echoPin, HIGH);
}
```

SERVO

Es un tipo especial de motor que pretende girar 180 grados. 3 pintes de conexión: GND VCC y data.

COMANDOS BASICOS: Acordarse de importar la librería Servo.h: **#include <Servo.h>**

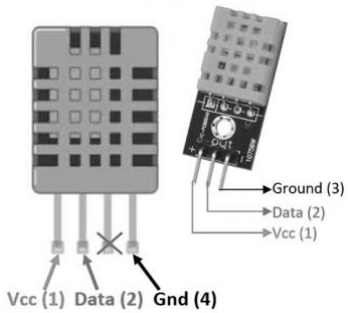
Metodos	Función
Servo.attach(pin)	Asigna la variable Servo a un pin.
Servo.detach()	Libera la variable Servo a de un pin.
Servo.write(angulo)	Escribe un valor en el servo, controlando el eje en consecuencia. Recibe un valor entre 0 y 180.
Servo.attached()	Comprueba si la variable Servo está conectada a un pin. Retorna TRUE o FALSE
Servo.read()	Retorna el ángulo actual del servo.

FUNCIONAMIENTO:

Funciona mediante las señales de PWM. Las ondas de pulso son requeridas para el circuito de control electrónico son similares para la mayoría de los modelos de servo. Esta señal tiene la forma de una onda cuadrada. Dependiendo del ancho del pulso, el motor adoptará una posición fija.

TEMPERATURA Y HUMEDAD: Sensor DHT11

Tiene 3 pines: VCC, GND y data. Si algún sensor trae una cuarta pata no se usa.



Para utilizar:

1. #include <DHT.h>

2. #define DHTPIN n // Siendo N el pin donde se transfiere la data.
3. #define DHTTYPE DHT11
4. DHT dht (DHTPIN, DHTTYPE)
5. En void loop: dht.begin();

Para leer humedad el commando es: dht.readHumidity() mientras que para leer temperatura el comando es dht.readTemperature().

COMUNICACION SERIAL:

La comunicación serial es una forma de enviar información entre dos dispositivos distintos.

La información se transmite en una señal de voltaje que varía en el tiempo. La onda alterna entre estados altos (1) y bajos (0). Con cada estado se transmite 1 bit y cada bit tiene una duración de tiempo definida, que debe ser la misma para los 2 dispositivos que se están comunicando. Este tiempo está definido por la velocidad de transmisión, generalmente llamada Baud Rate. Si tengo un Baud Rate de 9600 significa que transmito 9600 bits/segundo.

Los bits se agrupan en paquetes de información, de manera que cada paquete transmite 1 byte de información (8 bits); por lo tanto 1 byte es la mínima información que se puede transmitir.

La placa Arduino Uno cuenta con dos pines que admiten comunicación serial con Hardware dedicado, esto quiere decir que cuenta con circuitería especializada para la comunicación. Estos pines son el 0 (RX) y 1 (TX). Cuando se utilizan para comunicación serial, no pueden utilizarse para otra cosa.

Además estos pines están conectados a un conversor para poder comunicarse con el puerto USB de la computadora. Por dicha razón generalmente son utilizados para enviar mensajes a la computadora con el fin de identificar errores en los programas.

Aunque otros pines no tengan circuitería dedicada para comunicación serial, existe la librería SoftwareSerial, que utiliza software para utilizar dos pines digitales como RX y TX.

COMUNICAR DOS ARDUINOS:

La comunicación serial consta de 2 cables TX y RX. Cada dispositivo transmite por su TX y recibe por su RX. Por lo tanto el TX de un dispositivo se conecta al RX del otro y viceversa. También es necesario conectar las tierras de los dispositivos para estar seguros que la referencia de ambos sea la misma.

COMANDOS BASICOS

- **Serial.begin(BaudRate);**
  Este comando inicializa la comunicación serial y define la velocidad de transmisión.
- **Serial.available();**
  Este comando devuelve la cantidad de bytes que hay disponibles para leer.
- **Serial.readString();**
  Lee todos los bytes que hayan disponibles y los devuelve para guardarlos en una variable del tipo String.

- **Serial.println(texto**); Envía por serial lo que tenga la variable texto con un salto de línea al final

También se pueden sumilar los pines de comunicación serial mediante software serial.

I. Primero hay que incluir la librería: #include <SoftwareSerial.h>.
II. Luego, definir miSerial → SoftwareSerial miSerial(4, 5) donde 4 es RX y 5 TX.
III. Los comandos son los mismos, pero con miSerial en vez de Serial.