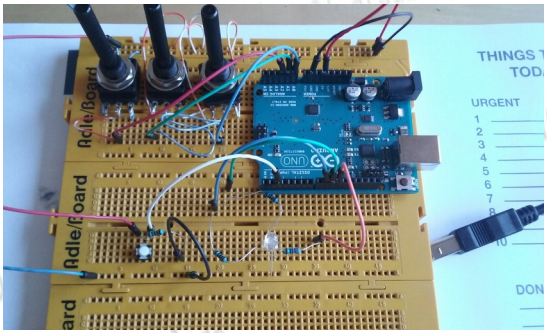
Arduino



Fco. Javier Rodríguez Navarro

Cuadernos técnicos
www.pinguytaz.net

Indice

1. HISTÓRICO

Versión	Fecha	Autor	Observaciones
1.0	Abril 2018	FJRN	Creación
1.1	Noviembre 2019	FJRN	Se añade PullUP para botones

2. INTRODUCCIÓN

Este cuaderno cubre Tipos de Arduinos con sus características, ventajas y desventajas.

También se comenta la programación básica y común del Arduino, los tipos de entradas y salidas de este a la que conectaremos nuestros sensores y actuadores. Con esta información podremos conocer el potencial de arduino y empezar a programar.

Los detalles específicos de los sensores o dispositivos a conectar se realizan en los cuadernos técnicos específicos para estos.

3. ARDUINO

Arduino es hardware con un microcontrolador, normalmente AVR, puertos digitales y analógicos de entrada / salida, el número de estos dependerá del modelo de Arduino, que nos permitira interactuar con el mundo exterior. Podemos que arduino es el cerebro (lo programamos nosotros) y los sensores y actuadores son por ejemplo los ojos y manos.

A las placas de Arduino se pueden conectar otras placas de expansión que llamaremos "*shields*", que amplían los funcionamientos de la placa Arduino.

¿Que nos ofrece arduino frente a un microcontrolador?. La diferencia es que arduino es una placa que contiene este microcontrolador y a su vez todo un conjunto de circuitería a su alrededor y entorno de desarrollo (lenguaje de alto nivel adaptado y no ensamblador) así como un bootloader que nos facilita su programación y diseño de nuestros proyectos. El diseño de esta placa es hardware libre lo que ha permitido la creación de muchas otras placas compatibles con sus pros y contras que hablaremos así como una gran

comunidad de usuarios que aportan conocimiento y librerías que podremos usar en nuestros proyectos.

El micro tiene un cargador de arranque (bootloader) que nos permite que realicemos una programación de este con un lenguaje de alto nivel (basado en processing y Wiring) y un entorno común para todas las placas mediante el puerto de USB de la placa.

3.1. Modelos de Arduino

Ya hemos comentado que el diseño hardware de Arduino esta publicado como licencia libre, por lo que cualquiera puede crear su tarjeta Arduino, disponemos de muchas pero aquí recomendamos ir a la pagina de Arduino para elegir el modelo que nos interesa utilizar o buscar otros modelos. Lo que vamos a definir son las características más habituales a mirar para nuestros proyectos.

- **Microprocesado y frecuencia de reloj**: normalmente no nos importara ya que no bajaremos tanto de nivel para usar características tan propias de la placa.

- **Entradas / salidas digitales:** Son las entradas / salidas que solo permiten los valores *ALTO* y *BAJO*, por regla general operan a 5V y 20mA máximo. Este parametro es importante pues si necesitamos más podemos poner un SHIELD que las amplie o usar varios Arduinos que se comuniquen entre ellos y una ser el maestro.

  Algunas de estas también se utilizan como PWM (Segun se configuren) o como puerto serie, SPI, I2C.

- **Salidas PWM**: Simulan una salida analógica mediante modulación de pulsos y son algunas de las digitales comentadas anteriormente.

  Algunos Arduinos disponen de salidas digitales propiamente dichas y nos simulaciones como las PWM.

- **Entradas analógicas:** Poca descripción necesitan, suelen tener una resolución de 10 bits y los valores de entrada van desde 0V a +5V(Este puede ser configurables mediante AREF, en algunos modelos, y programación).

- **UART** o puertos serie, podemos pasar de uno del Arduino-UNO a 4 que tiene el

Arduino-DUE.

- **Tamaño EPROM:** Memoria permanente que nos permite almacenar datos cuando este esta apagado, es ideal para guardar datos de configuración de nuestro proyecto.

  Por desgracia muchas veces no la usamos guardamos estos datos a capón en el código. Recordemos que un simple K, son 1.024 bytes (0-255) a poder utilizar.

- **SRAM:** Memoria donde se guardan la variables globales y locales.

- **Memoria FLASH**: Es donde almacenamos nuestro Bootloader y programa, por eso siempre definiremos el tamaño de esta y el tamaño del Bootloader y sabremos de la que disponemos para el programa.

  Para ahorrar memoria SRAM podemos guardar las cadenas fijas en esta memoria mediante el macro F().

  Siempre podremos cambiar el bootloader pero debemos ser consciente de lo que hacemos.

- **Tensión / Corriente funcionamiento:** Es la tensión/corriente de los pines de E/S, normalmente 5V pero el DUE por ejemplo es de 3.3V

- **Tensión de alimentación** es el rango en el que el Arduino se alimenta, normalmente 7-12V y este ya rectifica a 5V.
- **Comunicación:** SPI, I2C, etc.

Modelo	E/S Dg. (PWM)	E. Analog.	UART	EPROM	SRAM	Flash (Boot)	T/C	Alimentación
UNO	14(6)	6	1	1K	2K	32(0,5K)	5V 20mA	7-12V
NANO ATMega328P ATMega168	14(6)	8	1	1K 512B	2K 1K	32(2K) 16(2K)	5V 20mA	7-9V
ZERO	22 (10)	6 12b 1S 10b	2	0	32	256K	3,3V 7mA	3,3V

Podemos ampliar el número de placas y características en la pagina oficial de arduino, que se describe en el anexo de enlaces, así como otras placas de otros fabricantes. OJO debemos tener claro las placas que son de otro fabricante, las originales de arduino y las que llamaremos pirata que figuran como arduinos y en realidad son copias de mala calidad.

Dentro de placas de otros fabricantes tendremos:

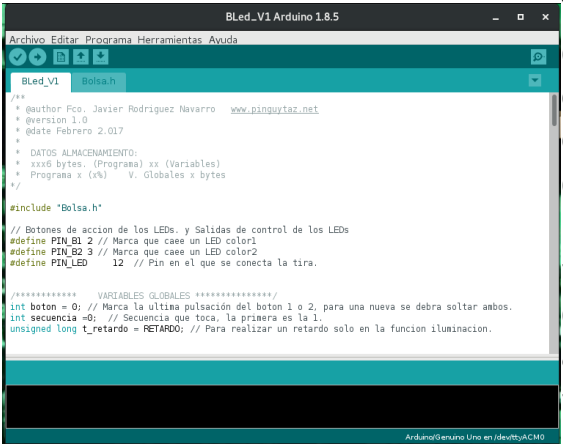
- **BQ Zum Core**

http://diwo.bq.com/product/placa-zum-core

- **Freeduino**
  https://www.freeduino.org/about.html
- **Zigduino**
  http://www.logos-electro.com/zigduino
- **Funduino**
  https://www.funduinoshop.com

No obstante para iniciarse lo mejor es no complicarse la vida y comprar un Arduino-UNO original y después ya ver las características de tamaño, salidas, entradas, etc que se necesita y ver cual nos interesa por modelo-precio-y-confianza.

4. IDE "ENTORNO DE DESARROLLO INTEGRADO"



Ya comentamos que arduino contaba con un entorno de desarrollo propio, que nos facilita el desarrollo de arduino, facilitándonos un conjunto de herramientas como son: el editor, el compilador, consola serie que nos permite tracear y ver datos obtenidos, cargador del programa en arduino, gestión de la librerías, etc.

El entorno de programación nos lo bajaremos de la pagina de arduino (https://www.arduino.cc/en/Main/Software) como casi todo lo necesario para empezar a trabajar con el, y ya que en ella también tenemos foros y tutoriales que nos permiten ampliar nuestros conocimientos.

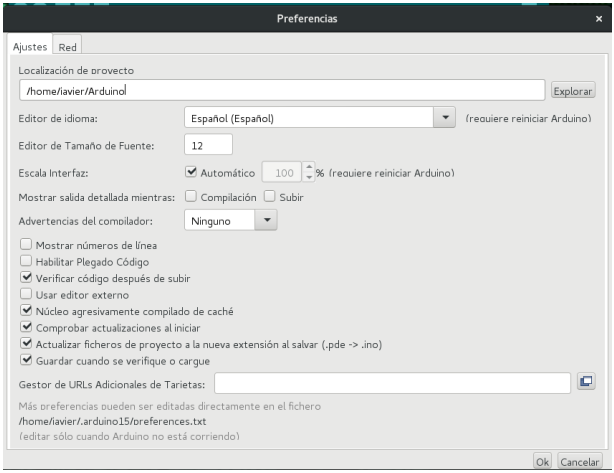
Como podemos ver después de las opciones de

menú tenemos 5 iconos que nos permitirán:

- Verificar la sintaxis del programa, lo compila pero no realiza la subida a arduino.
- Compilar el programa y subir este a nuestro arduino.
- Crear un nuevo programa.
- Abrir un programa existente.
- Salvar el programa actual.
- Totalmente a la derecha la apertura del monitor serie que nos ayudara a depurar y comprobar el funcionamiento de nuestro programa.

Después podemos ver los ficheros que componen ese programa por regla general un fichero con extensión “.ino” pero en ocasiones tambien ficheros cabecera con extensión “.h”

Debajo de las carpeta el editor donde escribiremos nuestro programa y por ultimo la



consola de error donde podremos ver nuestros mensajes de compilación y carga de programa de arduino. Tanto el entorno de edición como los mensajes que nos aparecen en la consola de error se pueden configurar si vamos a la opción “Archivos → Preferencias”. Por ejemplo:

- Tamaño de la fuente
- Si deseamos o no que la compilación o subida del programa de mensajes detallados de lo que estan haciendo.
- Si queremos que al lado de nuestro código muestre el numero de lineas.

4.1. Carga de un programa

Una vez que lo hemos verificado nos quedara subirlo, antes de dar al boton de compilar y subir deberemos definir el modelo de arduino desde la opción “Herramientas → Placa”

Después en que puerto tenemos conectado nuestro arduino “Herramientas → Puerto” y con esto ya estamos en disposición de compilar y cargar el programa, es en este momento donde entra a trabajar el “*bootloader*” para recoger la información a través del puerto definido de forma sencilla.

Una vez se ha subido todo el código sin problemas se iniciara el programa.

4.2. Entornos graficos

http://s4a.cat/

https://descubrearduino.com/alternativas-graficas-programar-arduino/

BQBloq

5. PROGRAMACIÓN

Un programa arduino es un fichero con extensión ".ino" que se encontrara en una carpeta de de igual nombre pero sin la extensión, en esta carpeta también podremos encontrar otros ficheros por ejemplo extensión ".h" que tendrán la definición de estructuras, constantes, variables incluso definición de clases que se complementaran con archivos de extensión ".cpp".

En principio solo .ino y .h, ya que lo otro nos lo reservamos para la creación de nuestras librerías.

5.1. Estructura de programa

Un programa de Arduino se compone de dos partes:

1. *setup()* que contiene el código inicial que se ejecuta al arrancar el Arduino.
2. *loop()* que contiene el código que se ejecuta ciclicamente hasta que se apaga o resetea el Arduino.

Antes de *setup()* definimos la variables y

definiciones globales, luego en el *setup()* definimos los pines ya que esto se ejecuta solo al inicio.

Después del *loop()* definiremos la funciones.

```
/*********************************
 * Autor:
 * Versión:
 *
 * Descripción:
 *
 *  DATOS ALMACENAMIENTO:
 *  Programa: XXX bytes
 *  Variables YYY bytes
 ************************/
#include <??????>

#define ????? ¿¿¿¿¿¿ // Definición VALOR

void setup()
{
    // Definición de pines
    // Inicializaciones
}
void loop()
{
   // Definición variables locales.
   // Programa
}

/************************
 * func(par, ...)
 *
 * Descripción
 *
 * @param Descripción parametro
 * @return  Descripción de retorno
 ********************************/
 tipo o void func(tipo var,...)
 {
 }
```

También podremos definir clases y heredarlas, pero esto se comentara en otro apartado.

Los ***comentarios*** pueden ser:

- De bloque encerrado entre /* ……. */
- Linea // Hasta final de linea

5.2. Variables y Constantes

Una variable contiene datos y pueden ser globales, definidas antes del bloque *setup()* o al inicio de los bloques o funciones y son locales a ese bloque o función.

Se definen como es lógico antes de utilizarse y se definen <tipo> <nombre>

a) Tipos de variables

- **byte**

  Almacena un valor numérico de 8 bits sin decimales. Tienen un rango entre 0 y 255.

- **int**

  Son un tipo de datos que almacenan valores numéricos de 16 bits sin decimales comprendidos en el rango 32.767 to -32.768.

  Si anteponemos *unsigned* sera un entero sin signo y el rango sera entre 0-65.535,

- **long**

  Números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango -2147483648 a 2147483647.

Al igual que los enteros si anteponemos unsigned el rango estara entre 0-4.294.967.295

- **float**

  Números con decimales,con un rango comprendido 3.4028235E +38 a +38-3.4028235E.

  También llamado *double*.

- **void**

  Un tipo especial que significa es nada y se utiliza para indicar que no existen parámetros o que una función no retorna nada.

- **boolean**

  Un valor binario que puede ser 0, 1, *true* o *false*.

- **char**

  Ocupa un carácter y contiene un carácter, se puede asignar con su código ASCII (0-255) directamente o con el carácter ‘x’.

- **word**

  Lo mismo que un *unsigned int*.

b) Calificadores

Las variables pueden calificarse antes del tipo y estos calificadores pueden ser:

- **volatile**

  Fuerza que la variable se almacene en memoria y no en un posible registro.

- **static**

  El valor persiste entre llamadas, util como contador de llamadas a una función por ejemplo.

  Una variable *static* sera local a la función pero a diferencia de las otras que se crean y se destruyen, las estáticas se mantienen y solo se crean una vez.

- **const**

  Hace que la variable sea solo de lectura. Por regla general las constantes se suele definir con las directivas *#define <var> <valor>.*

- **PROGMEM**

  Indica que se almacene en la FLASH(memoria de programa) y no en la memoria RAM. Al contrario de los otros

calificadores este se pone antes del igual.

Para almacenar constantes de caracteres lo mejor es usar el macro F(<cadena>)

c) Constantes

- **HIGH, LOW** Estado de pines digitales
- **INPUT, OUTPUT** Configuración de pines digitales
- **true, false**
- Los numero que se inicia con 0, son en octal (018).
- Los números que se inician con 0b son binarios (0b101001)
- 0x para los e¡hexadecimales (0xFA)
- Una ***U*** al final fuerza que sea unsigned
- Una ***L*** al final que sea long
- Una ***UL*** al final que sea unsigned long.

d) Punteros

Las variables se almacenan en una dirección de memoria, esta la podemos conocer mediante el operador ‘&’ así una variable definida por ejemplo ‘a’ le anteponemos ‘&a’

indicara que es la dirección de esta. Esto es muy útil para pasar parámetros variables a las funciones.

Si queremos tener una variable que contenga estas direcciones no tendremos más que definirla de la siguiente forma

```
int *a;   // Puntero a un entero
float *b   // Puntero a un flotante
int c=12;  a= &c;  // a contiene la direccion de la variable entera c.
*a = 87;   // c pasa a tener el valor 87.
// OJO a = 9 hace que a apunte a la dirección 9, esto es el tipico error que no destroza un programa.
sizeof(c) // tamaño que ocupa la variable c que es un entero.
```

Como se ve en el código si el operador '&' es la dirección, el operador '*' representa el contenido.

Ya comentamos que los punteros son ideales para pasar valores por valor y así tener varios valores que se retornen de una función y no solo uno mediante *return*, recordemos esto cuando hablemos de las funciones.

```
int k = 9;
funcion1(&k); // Al finalizar la k tendrá el valor 10. Se envía la dirección

funcion1(int *a) { *a = *a+2;} // Llega un puntero o dirección a un entero.
```

Hablaremos más de los punteros al hablar de los arrays y en especial cadenas de caracteres, y es que muchas veces huimos de ellos y al final, exagerando muy poco, todo es

un puntero, vamos si se quiere uno divertir y conocer la potencia de ellos leer sobre punteros a funciones en C para rizar el rizo.

NOTA:

Aunque los punteros son divertidos, se deben usar con cabeza y precaución pues pueden producir errores que nos vuelvan locos.

5.3. Estructuras de control

- **if**(lógica) {cumple} **else** {no cumple}
- **while**(lógica) {Mientras cumple}
- **do** {mientras cumple} **while** (lógica)
- **for**(inicia;condicion;avance){Realiza}
- **break** Sale del bucle.
- **continue** Salta a la el resto de las lineas y va a la siguiente iteraccion.
- **switch** (variable) case valor: {Realiza si variable=valor; break} case valor2: {Realiza si variable=valor2; break} default: {Si ninguna variable cumple}

5.4. Operadores

Los típicos operadores aritméticos de suma(**+**), resta(**-**), multiplicación(*****), división(**/**), modulo(**%**)que es el resto de una división y lógicamente el de asignación(**=**) como los aritméticos simples.

Tenemos los operadores de comparación que utilizaremos en sentencias condicionales y son: igual a (**==**), distinto de (**!=**), menor que (**<**), mayor que (**>**), menor igual que (**<=**), mayor igual que (**>=**), un AND (**&&**), OR(**||**) y negación que invierte el resultado lógico (**!**)otros que son aritméticos compuestos.

También tenemos los que realizan operaciones lógicas a nivel de bits y son: AND binario (**&**), OR binario (**|**), XOR binario (**^**),Negación (**~**), desplazamiento a la izquierda (**<<**) y desplazamiento a la derecha (**>>**).

Y como no los olvidados operadores compuestos:

- **++** y **–** que incrementa en uno la variable que le sigue.
- **+= -= *=** y **/=** que suma, resta, multiplica o divide a la variable destino el valor de la derecha.

- **&=** y **|=** lo mismo con los operadores de bits AND y OR.

5.5. Funciones

a) Funciones tiempo

- **unsigned long *millis()*** Número de milisegundos desde que se inicio Arduino, el máximo es de 50 días, en ese momento empieza desde cero.
- ***unsigned long micros()*** Micro segundos desde que se inicio Arduino el máximo 70 minutos, en ese momento empieza de cero.
- ***delay(unsigned long)*** Realiza una pasa de los milisegundos indicados.
- ***delayMicroseconds(unsigned int)*** Pausa en microsegundos.

b) Funciones matemáticas

- ***min(x,y) max(x,y)*** Retorna el menor o mayor de los dos numeros.
- ***abs(x)*** El valor absoluto.

- ***constrain(x, Vmin, Vmax)*** Limita la variable a los rangos elegidos, se utiliza en conjunto con la función *map()*
- ***map(V,DeBajo,DeAlto,Abajo,AAlto)*** Retorna un valor escalado de un rango ABajo y AAlto, calculado según el rango DeBajo-DeAlto de entrada. Función utilizada en el retorno de las entradas analógicas por ejemplo.
- ***pow(base, exponente)*** Retorna la base elevada al exponente.
- ***sqrt(X)*** Raiz cuadrada del parámetro.
- ***double sin(float), double cos(float), double tan(float)*** Funciones trigonometricas de seno, coseno y tangente.

c) Funciones aleatorias

- ***random(max) random(min, max)*** retorna un numero aleatorio, con un maximo o en un rango min-max.
- ***randomSeed(semilla)*** Genera la semilla de la funcion random.

d) Funciones de Bits y Bytes

- ***byte low(x) byte hight(x)*** Retorna el byte de más a la derecha(menos significativo) o el más a la izquierda(más significativo).
- **word word() word word(Significativo, menossignificativo)** Retorna un word.
- **boolean bitRead(x,pos)** Retorna el bit de la posición indicada del número x.
- **bitWrite(x,pos,bit)** Pone el bit indcado en la posición 'pos' de la variable x.
- **bitSet(x,pos)** Pone un 1 en la posición uno de la variable x.
- ***bitClear(x,pos)*** Pone un 0.

e) Funciones caracteres

- ***boolean isAlphaNumeric (c)*** Indica si el carácter es alfanumerico.
- ***isWhitespace*** Si es un especio en blanco.
- ***isControl*** Carácter de control.
- ***isDigit*** Es un digito.
- ***isGraph*** Es un carácter imprimible.
- ***isLowerCase*** Si es minúscula.

- ***isUpperCase*** Si es mayusculas.
- ***isPunct*** Signo de puntuación.
- ***isHexadecimalDigit*** Si es un digito.

5.6. Arrays y cadenas de caracteres

Un array es un conjunto de datos de un tipo a los que se accede con un índice que va de ‘0’ hasta el tamaño del array menos uno. Los array no tienen que se de una sola dimensión sino que puede tener 2, 3 o las dimensiones que deseemos.

```
int unArray[4]; // Array de 4 valores enteros es decir irán del 0-3
char caracteres[4][2]; // De caracteres y tendrán 8 valores.

int a[4] = {1,2,3,4} // Inicializacion de un array.
```

Ya comentamos que hablaríamos más de los punteros al llegar a los arrays y es que interesa saber que la variable array sin los corchetes es la dirección a la primera posición del array.

```
int a[4] = {11,22,33,44}; // Declaración del array.
int *p;  // Puntero a entero.
p = a; // p tendrá la dirección de inicio al array a, y contiene 11.

Serial.print(*p); // imprimirá un 11 que es el valor de a[0].
p++; // Ahora apuntamos a a[1]. OJO con esto que podemos perder el control.
```

Esto debemos tenerlo muy claro pues los arrays se pasan a las funciones como la dirección al primer elemento del array.

```
f1(a); // Pasamos el array a cuyo contenido es {11,22,33,44}
void f1(int *p) { a = p[2]; } // a contendrá 33.
}
```

a) String (Cadena de caracteres)

Las cadenas de caracteres *'String' es eso un array de caracteres cuyo tamaño no se define directamente con el tamaño sino con un carácter de terminación que es '\0' también llamado n*ull. Debemos tener claro que un carácter se define con simples comillas ('a') y una cadena con dobles comillas ("Prueba") para que no tengamos dudas cuando la cadena sea de un solo carácter.

```
String c1 = "Hola";
String c2 = String("Otra forma");
c2 = c1 + " Tito"; // c2 contendra "Hola Tito";
```

Tenemos diversas funciones (Métodos al tratarse de una clase) que podemos usar con las cadenas, ya que String es una clase lo que definimos son objetos (hablaremos de esto más adelante.

```
String c = "Cadena de prueba";
String d = "";
int a;

a = c.indexOf('a');  // a sera 1 posición de la primera 'a' que encuentra.
a = c.indexOf('n');  // a sera 4 posición de la primera 'n' que encuentra.
a = c.lastIndexOf('e');  // a sera 13 posición de la ultima 'e' que encuentra.

c = " HOLA \n";
a = c.length();  // a sera 7 longitud de la cadena.
d = c.trim();  // d sera "HOLA" limpia principio y fin de espacios, etc.
c = d.toLowerCase(); // c se convierte en "hola" minusculas.
c = d.toUpperCase(); // c se convierte en "HOLA" mayusculas.

d.replace('H','P'); // Remplaza la H por la P. "POLA"
```

Tenemos varios métodos para el

tratamiento de cadenas:

- ***charAt(c)***: Accedo al carácter indicado.
- ***setCharAt(i,c)***: Pone c en la posición i.
- ***endsWith(S2):*** Mira si la cadena termina con la cadena S2.
- ***equalsIgnoreCase(S2)***: Compara ignorando mayusculas y minusculas.
- ***getBytes(byte b[],i):*** copia a b i caracteres.
- ***indexOf(c)***: busca la primera ocurrencia c
- ***lastIndexOf(c)***: ultima ocurrencia c.
- ***length()***: Longitud de la cadena.
- ***toInt()***: Convierte a entero.
- ***toFloat()***: Convierte a flotante
- ***toLowerCase()***: Convierte a minusculas.
- ***toUpperCase()***: Convierte a mayusculas.

5.7. Estructuras (struct)

El tipo estructura es muy interesante para contener en una sola variable todos los datos por ejemplo de un sensor, podemos decir que es un registro de una base de datos.

Una estructura (struct) es un conjunto de campos de diversos tipos y lo mejor es con un ejemplo, tanto su creación como uso.

```
struct Sensor
{
   string nombre ;
   int pinE ; // Pin de entrada
   int pinS ; // pin de salida
} ;

Sensor s1;
s1.nombre = “El primer sensor”;
s1.pinE = 12;
s1.pinS = 9;
```

Y como siempre es bueno dar un paso más en pro de la organización del código surgieron las clases que es una estructura que no solo tiene datos sino las funciones (métodos) que tratan a esos datos.

5.8. Clases

Una clase es una estructura con datos(propiedades), privados o publicos según los conozca el exterior, y funciones(métodos)que muchas veces son los únicos que podrán acceder a los datos de las estructuras.

Como norma el acceso a las variables de una clase se realiza con los métodos *‘set_<var>’* y *‘get_<var>’*.

La clase es la definición y esta se podrá incluso heredar para que de una base podamos

crear otras más complejas, y los objetos son instancias de las clases.

```
class MiClase
{
    private:
            propiedades y funciones que solo se acceden desde dentro.

    public:
            propiedades y funciones  que se pueden acceder desde fuera.
       MiClase();
       void f1(int);
} ;
MiClase::MiClase() { // Constructor se ejecuta al crear el objeto }
void MiClase::f1(int i){  // Definicion f1 }

MiClase c1;
```

Las clases las crearemos en un archivo ".cpp".

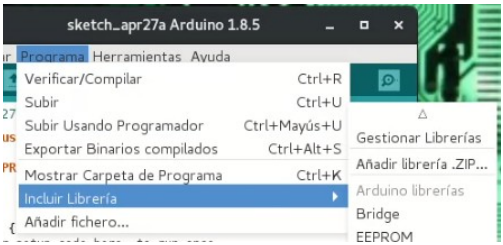
6. LIBRERÍAS

Una librería es un código que podemos incorporar a nuestro programas y que nos aporta nuevas funcionalidades como funciones para usar de forma abstracta sensores, en muchos casos esas librerías incorporan clases que permiten el uso de esos sensores y actuadores externos.

Un ejemplo de librería puede ser para usar motores paso-a-paso, que nos permite abstraernos de como debemos activar el motor y solo decimos que queremos que pase.

"http://www.pinguytaz.net/index.php/liberia-mpap/" en este enlace podemos ver un ejemplo de librería.

Podemos tener precardas y solo es necesario escribir en el código "#include <xxxx.h>" donde xxxx es el nombre de la librería o desde



"Programa→Incluir Librería" la seleccionamos.

En el caso de que no sea una librería estandar y nos l

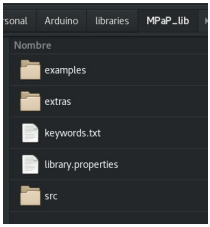
hayamos descargado de algún sitio como por ejemplo la librería de motores paso a paso (http://www.pinguytaz.net/M_Archivos/MPaP_lib.zip) la cargaríamos en la opción “añadir librería ZIP” y se descomprimirá de forma automática en nuestro directorio de librerías.

También podemos cargar otras librerías desde “Gestionar Librerías” además de ver las versiones y cuales pueden ser actualizadas.

6.1. Creando nuestras librerías

Podemos crear nuestras funciones, que si son de una funcionalidad tipo las podríamos agrupar en fichero, pero lo mejor es realizar nuestras librerías y solo cargarlas cuando lo requiera nuestro proyecto.

Nuestra librería se agruparan en un fichero “.zip” con la siguiente estructura y ficheros.



El directorio “**examples**”, tendremos código de ejemplo de la

librería y lo podremos cargar de la misma forma que los ejemplos originales de arduino desde la opción “Archivo→Ejemplos” de forma que nos cargara los ficheros “.ino”. Este directorio no es obligatorio pero ser buena idea tener una ayuda de como usar nuestra librería.

El directorio “**extras**” donde tendremos ficheros no necesarios para la librería pero si que pueden aportar información como esquema típico de conexión si es que definimos una librería de un sensor, un fichero de histórico de versiones así en cualquier momento podremos ver los cambios y también si queremos el manual de uso, pero ojo no carguemos muchas cosas y hagamos una librería impracticable, yo recomiendo poner el documento de librería por separado.

Directorio “**src**” las fuentes de las librerías y como mínimo nos encontraremos con dos ficheros, uno con extensión *.cpp* que tendrá el código fuente de nuestra librería. El otro fichero sera extensión *.h* (llamado fichero cabecera) donde estará la declaración de las funciones y clases definidas en nuestro fichero *.cpp* que estará la lógica

El nombre del fichero cabecera es el que

pondremos nuestra clausula *#include<….h>*

En el directorio raíz tendremos los siguientes ficheros:

- **keywords.txt**: Tiene las palabras claves de nuestra librería para que al escribir el código las palabras claves cambien de color.
  - **KEYWORD1**: Para los tipos de datos y clases.
  - **KEYWORD2**: Para las funciones o métodos de las clases.
  - **LITERAL1**: Las constantes (#define)
- **library.properties**: Este fichero describe la librería, esta información aparecera en el gestor de librerías y su formato es el siguiente.

```
name=MPaP
version=1.0.1
author=Fco. Javier Rodriguez Navarro
maintainer=Javier <webmaster@pinguytaz.net>
sentence=Gestión de motores paso a paso unipolares
paragraph=Libreria para la utilización de motores paso a paso UNIPOLARES, con solo indicar los pines de conexión podremos avanzar tanto por pasos como por angulos.
category=Device Control
url=http://www.pinguytaz.net/index.php/liberia-mpap/
architectures=*
```

- **name**: El nombre de la librería.

- **Version:** Versión.
- **Author:** Autor de la librería, si existen varios se separan por comas.
- **Maintainer:** Nombre y mail del que mantiene la librería.
- **Sentence:** Frase que describe la librería.
- **Paragraph:** Descripción larga de la librería, aparece después de *sentence* sin negrita.
- **Category:** Categoria a la que pertenece la librería, nos permitirá filtrar en el gestor de librerías y los valores posibles son:
  - Display
  - Communication
  - Signal Input/Output
  - Sensors
  - Device Control
  - Timing
  - Data Storage
  - Data Processing
  - Other

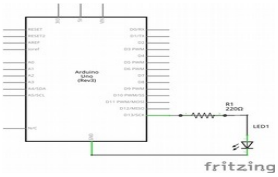
- **url:** La URL del proyecto.
- **Architectures:** Arquitecturas separadas por coma, un asterisco (*) para cualquiera.

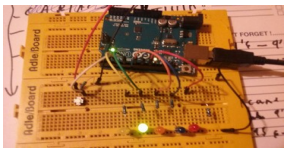
7. E/S DIGITALES

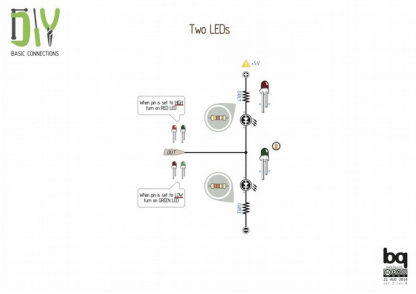
Las salidas y entradas digitales, son aquellos pines del Arduino que se soportan 0 (0V) o 1(depende del modelo pero suelen ser 5V o 3,3V) y una corriente que depende del modelo. Aunque los típicos son 40 mA, se recomienda realizar los cálculos con 20mA por PIN.

Antes de usarlas deberemos indicar si van a ser entrada o de salida.

```
pinMode(13,OUTPUT);  // Pone el PIN13 digital como salida.
PinMode(5,INPUT); // Pone el PIN5 digital como entrada.
```







7.1. Salida digital

La salida digital se activa o desactiva

con el siguiente comando.

```
digitalWrite(13 , HIGH); // Pone a ALTO el PIN13 (5V o 3,3V según modelo)
digitalWrite(13 , LOW);  // Pone a BAJO el PIN13 (0V)
```

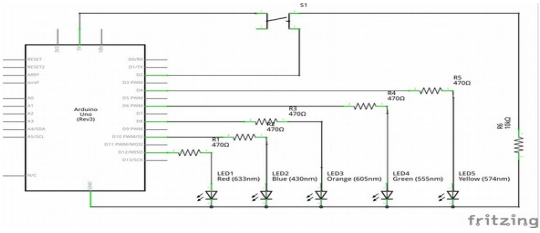
El típico ejemplo y más visual es poner un diodo led para ver el funcionamiento.

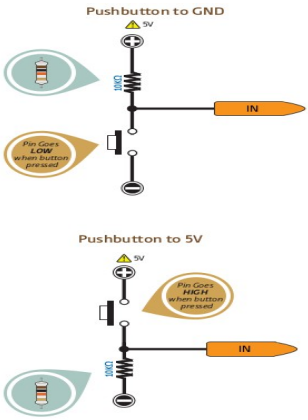
Un ejemplo de conexión con dos LED para que uno encienda y otro apague puede ser la siguiente, ese LED puede ser cualquier otro dispositivo que se active con un sencillo voltaje de 5V y hasta 20mA.

Si el dispositivo a encender o apagar necesita más corriente o tensión deberemos apoyarnos en transistores, reles u otros dispositivos.

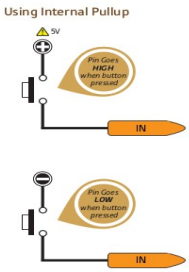
7.2. Entrada digital

La entrada digital se activa con 5V o 3,3V según modelos.





Debemos tener en cuenta que una entrada abierta sin más nos puede dar valores aleatoreos al estar en alta impedancia, para evitar esto en los botones utilizaremos una resistencia de 10K al positivo “pullup” o una resistencia a masa “pulldown”.



Se puede conectar directamente pero en este caso deberemos configurar el pin a “INPUT_PULLUP” para usar la resistencia interna. Normalmente pondremos nosotros la resistencia.

8. ENTRADA ANALOGICA

Una señal analógica es una señal que recibimos en forma de tensión que en un intervalo –Vcc y + Vcc. En el caso de Arduino esta entre 0V y 5V.

En el arduino UNO tenemos 6 entradas analógicas (A0-A5) y que nos pueden dar valores entre 0-1023, esto es así porque el ADC(Convertidor Analogico Digital) con el que cuenta Arduino para poder recoger esa señal analógica y convertirla a binario para así poderla procesar tiene una precisión de 10 bits.

La resolución en el arduino UNO es fija a 10bits pero otros modelos como ZERO o DUE tienen esta resolución por defecto pero se puede cambiar a 12bits con la función **“analogReadResolution()”**

8.1. Precisión

La precisión teniendo en cuenta que medimos entre 0-5V y con 10bits sera de 4,88mV (5/1024)pero esto lo podemos cambiar si el

rango de medición lo reducimos y modificamos la tensión tomada como referencia con la función **analogReference(typo)**:

- DEFAULT: 5V o 3,3V según la placa.
- INTERNAL: 1.1V en placas Atmega168 y 328 y en las placas ATmega32U4 y ATmega8 sera 2,56V.
- EXTERNAL: Aplicado en el Pin AREF(0-5V) pero debemos asegurarnos que la señal no lo superara.

NOTA:

Si deseamos cambiar entre voltaje de referencia externo e interno, deberemos poner el voltaje de referencia con una resistencia de 5K.

Tendremos en cuenta que la resistencia alterará el voltaje que se usa como referencia porque hay una resistencia interna de 32K en el pin AREF. Los dos actúan como un divisor de voltaje, por lo que, por ejemplo, 2.5V aplicado a través de la resistencia producirá 2.5 * 32 / (32 + 5) = ~ 2.2V en el pin AREF.

8.2. Lecturas sensores analógicos

En un sensor lineal(ejemplo TMP36 para temperatura) nos dará una salida proporciona a la medida del sensor que es la que se conectara al pin de entrada analogica.

Cuando el sensor da unos valores de salida entre 0-5V lo podremos conectar directamente, en el caso de que tener tensiones superiores usaremos divisores de tensión.

Antes de actuar con el valor leído con la función “V=Lectura*5v/1024” el 5v se cambiara por el voltaje de referencia.

También podremos usar las función map(valor, deMin(0),deMax(1023),aMin,aMax)

9. ANEXO "ENLACES"

- Pagina oficial: https://www.arduino.cc
- Componentes: https://www.sparkfun.com
- Herramientas para diseño esquemas y PCB

  http://fritzing.org
- Información conexiones: http://www.pighixxx.com/test
- ***Ejemplos***

  S-Digitales: http://fritzing.org/projects/ejemplo-salidas-y-entrada-digitales

  E/S-Analog: http://fritzing.org/projects/rgb-con-recogida-de-datos-de-tres-potenciometros

  PWM: http://fritzing.org/projects/pruebas-pwm-lectura-serie
- Varios

  http://diymakers.es

  http://www.prometec.net

  http://panamahitek.com/category/arduino

10. ANEXO "OTRAS HERRAMIENTAS"

Fritzing

opencad

Tipos de arduinos con sus caracteristicas, ventajas y desventajas.
Tambien se comenta la programación basica de este que se complementa con los otros cuadernos tecnicos en el que se explican los sensores y actuadores
Este cuaderno es solo comandos basicos, creación de librerias, etc. que aplican a los arduinos en comun.