AUTOMATIZACIÓN DE LA CLIMATIZACIÓN DE ACUARIO Y HOGAR

MEMORIA PRESENTADA POR:
José Gil López

Grado en Ingeniería Informática

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

Índice

1-. Introducción

Este proyecto nace ante la necesidad de controlar una calefacción radiante a través de una bomba de calor y/o placas termo solares dado que el instalador no ofrece soluciones automáticas.

Básicamente necesitamos alternar cada una de estas tecnologías para que no funcionen simultáneamente, así utilizar estas energías más eficientemente y tener un ahorro energético mayor que básicamente es uno de los fines de la instalación de placas termosolares en una vivienda, además del ecológico para aprovechar la abundante energía del sol y poder amortizar la instalación más rápidamente.

Además, tenemos que decir que al utilizar hardware y software libre favorece a la reducción de costes y de cualquier método de espionaje en nuestro hogar, ya que estamos en unos momentos de dudosa privacidad.

Para finalizar, el trabajo tendrá un enfoque teórico-práctico, ya que expondremos el diseño y su implementación en esta memoria y nos dispondremos a montarlo tipo prototipo para testear su funcionamiento.

1.1-. Leyendas

Aparecerá el texto en negrita cuando se mencione términos de IDE arduino, librerías, electrónica y programación.

Ejemplo: "***Dallas DS18B20***: Esta es la librería de los sensores DS18B20".

Cuando se cite documentación oficial aparecerá entrecomillado.

Ejemplo: "Una variable debe ser declarada *volatile* cada vez que su valor pueda ser…".

2-. Justificación del proyecto

Realmente en el mercado existen sistemas de climatización avanzados de hogar y muy asequibles a nivel económico y fáciles de instalar, pero solo se dedican a controlar la climatización del hogar, con nuestro proyecto podemos agregar mediante módulos muchas más funciones a este sistema de control de climatización, como el control de climatización de acuarios que también hemos desarrollado en este proyecto.

En mi caso, no existe ningún sistema de control de climatización del hogar que administre eficientemente las placas termosolares y la típica bomba de calefacción radiante, porque se puede dar la situación de que se esté calentando el agua en las placas termosolares por un día soleado y nuestra máquina esté en funcionamiento, es una situación en la que estamos realizando un gasto innecesario de energía ya que las placas termosolares por sí solas ya están calentado el agua y queremos evitar esta situación.

A nivel de climatización inteligente de acuarios existe un numero escaso de estos sistemas a nivel doméstico, únicamente a nivel de grandes acuarios que son extremadamente caros.

Básicamente queremos implementar un sistema centralizado de control de climatización ya que la temperatura del hogar influye en gran medida en la temperatura del agua de cualquier acuario que esté dentro de su ámbito y gracias a esta situación vamos a intentar aprovechar el calor (o frío) generado por la climatización del hogar para que nuestro acuario tenga la temperatura correcta y no tener un gasto innecesario de energía.

Este proyecto no contempla la posibilidad de comercializar el sistema de climatización, ya que es un proyecto de hogar creado con la filosofía de “hazlo tú mismo” pero en otras secciones estudiaremos la posible amortización en caso de una hipotética comercialización.

3-. Objetivos

El objetivo implementado es la climatización de hogar y acuarios que abarcamos en este proyecto y a medio y largo plazo es generar un sistema domótico inteligente que gestione todo el hogar.

- Objetivos implementados.
  - Climatización de hogar y acuarios.
- Objetivos medio y largo plazo.
  - Control lumínico de hogar.
  - Control de ventanas y persianas.
  - Control de puertas.
  - Control de seguridad y vigilancia.
  - Control de limpieza.
  - Control de alimentación.

4-. Metodología

La metodología esta dividida en 4 partes:

- 4.1-. Requisitos y funcionamiento
- 4.2-. Metodología climatización acuario (Arduino Uno)
- 4.3-. Metodología Servidor (RaspBerry pi)
- 4.4-. Metodología climatización hogar (Raspberry Pi)

4.1-. Requisitos y funcionamiento

4.1.1-. Requisitos

Para poder realizar este proyecto tenemos que elegir unos componentes que puedan controlar todos los procesos y, como el sistema se irá ampliando en el futuro, necesitamos un centro de control muy versátil.

Hemos elegido como cerebro del sistema por su potencia, versatilidad, actualizaciones, compatibilidad y soporte del fabricante el computador **Raspberry Pi**, ya que son componentes muy fáciles de encontrar.

Para parte de implementación de la climatización del acuario hemos elegido la placa de **Arduino Uno** por las mismas razones que la **Raspberry Pi,** ya que se puede averiar y necesitaríamos recambios rápidamente.

Uno de los requisitos indispensables en este proyecto es que el acuario sea autónomo, ya que puede fallar nuestro sistema operativo, así que nuestro computador solo recibirá información para mostrar, y no podrá actuar sobre él.

4.1.2-. Funcionamiento

Vamos a incluir un esquema del funcionamiento de nuestro proyecto, que será detallado a lo largo del desarrollo del mismo en las diferentes secciones. Ilustración 1: 4.4.2.5

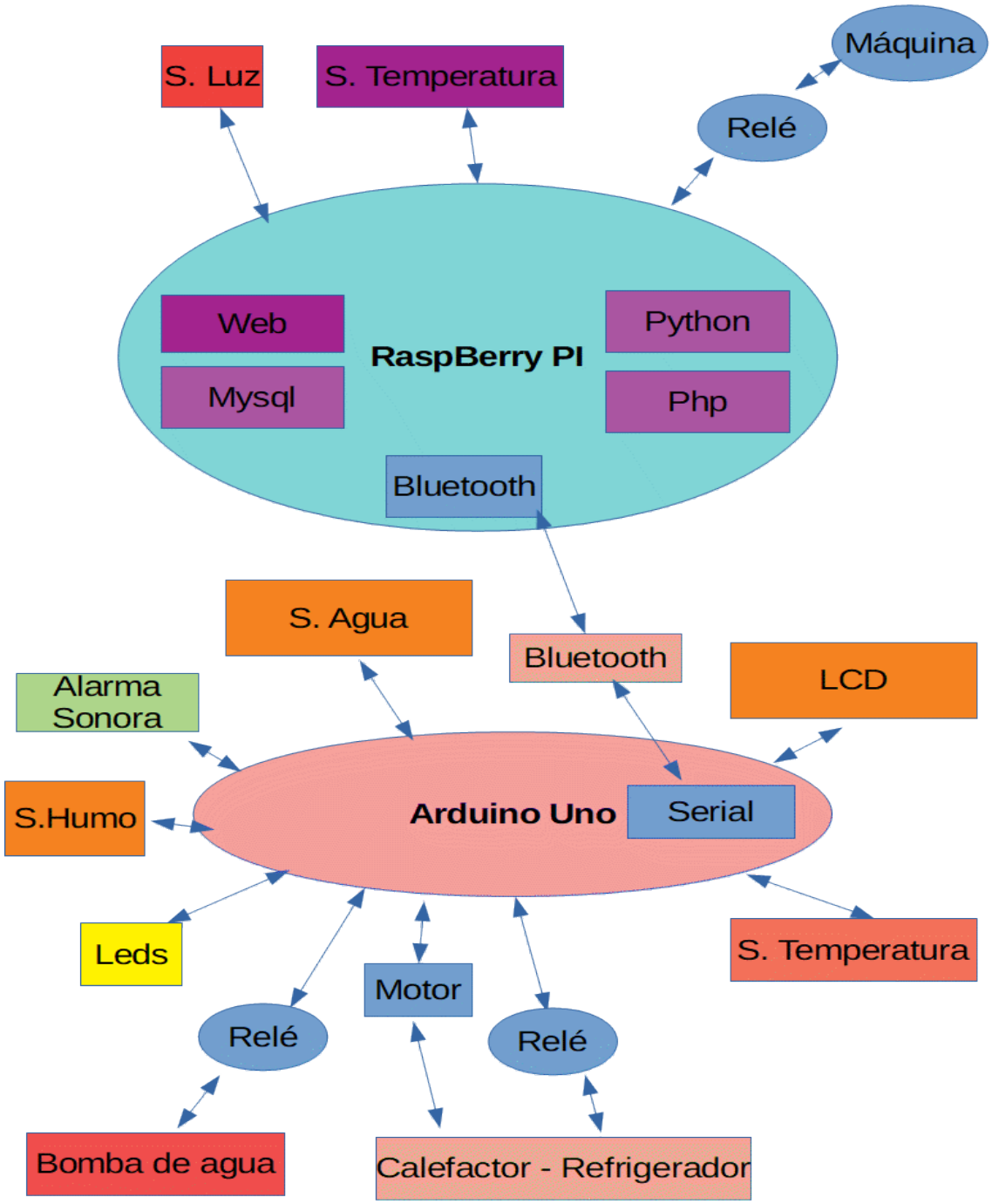


Ilustración 1: 4.1.2

4.2-. Metodología climatización acuario (Arduino Uno)

En este apartado nos vamos a dedicar a la creación de la parte del proyecto respectiva al acuario, donde nos dispondremos a conectar en la placa **Arduino Uno** (Ilustración 2: 4.2) los componentes para llevar a cabo esta parte del proyecto.

Características técnicas:

- **Microcontrolador: ATmega328**
- **Voltaje: 5V**
- **Voltaje entrada (recomendado): 7-12V**
- **Voltaje entrada (límites): 6-20V**
- **Digital I/O Pins: 14 (de los cuales 6 son salida PWM)**
- **Entradas Analógicas: 6**
- **DC Current per I/O Pin: 40 mA**
- **DC Current parar 3.3V Pin: 50 mA**
- **Flash Memory: 32 KB (ATmega328)**
- **SRAM: 2 KB (ATmega328)**
- **EEPROM: 1 KB (ATmega328)**
- **Clock Speed: 16 MHz**

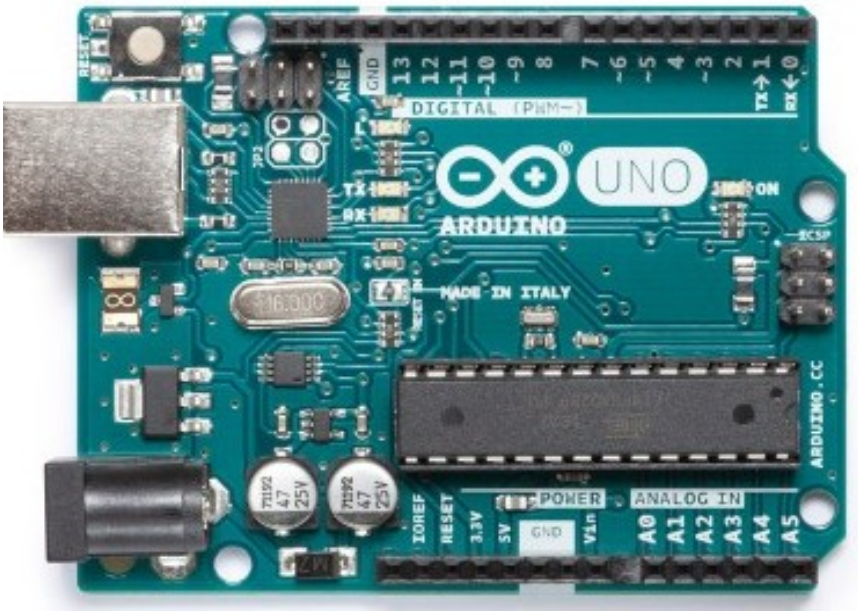


Ilustración 2: 4.2

Descripción de pines:

- Alimentación (**POWER**):
  - **Vin**: Para alimentar la placa usando una fuente externa.
  - **5V**: Este pin da una tensión de 5V a los periféricos que queramos conectar de 5V.
  - **3.3V**: Este pin da una tensión de 3.3V a los periféricos que queramos conectar de 3.3V.
  - **GND**: Conexión a masa GND.
  - **IOREF**: Diferencia de voltaje para las entradas analógicas.

- **Entradas y salidas**: Cada uno de los 14 pines digitales del Uno se puede usar como entrada o salida, usando las de las funciones reservadas pinMode(), digitalWrite(), digitalRead(), analogRead() y analogWrite().
  - **Serial**: 0 (**DX**) y 1 (**TX**), se usan principalmente para enviar y recibir los datos **TTL** al serial, pero también se pueden usar como pines normales y conectar los periféricos, pero para cargar el programa debemos desconectarlos ya que por ahí subimos el **sketch** al la **EEPROM.**
  - **Interrupciones externas**: 2 y 3. Estas pines se pueden configurar para activar una interrupción a nivel bajo, un flanco ascendente o descendente, o un cambio en el valor, con la función **attachInterrupt()** para más detalles.
  - **PWM**: 3, 5, 6, 9, 10 y 11. Proporcionan salida **PWM** de 8 bits con la función **analogWrite()**.
  - **SPI**: 10 (SS), 11 (**MOSI**), 12 (**MISO**), 13 (**SCK**). Estos pines admiten comunicación **SPI** utilizando la biblioteca **SPI**.
  - **LED**: 13. Hay un **LED** integrado accionado por el pin digital 13.
  - **TWI**: pin **A4** o **SDA** y pin **A5** o **SCL**. Soporte de comunicación **TWI** utilizando la biblioteca **Wire** (explicación en la siguiente referencia **1-Wire**).
  - **RESET:** Un pequeño botón que reinicia la placa.

4.2.1-. Sensor de temperatura DS18B20

Se trata de un sensor digital (Ilustración 1: 4.2.1) que tiene dos cualidades muy útiles, por un lado disponemos de una versión encapsulada y cableada que permite su uso en exteriores e incluso en contacto con líquidos, de hecho es sumergible, que es lo que me llevó a decidirme por utilizar este sensor ya que el acuario, al ser marino, se oxidarían rápidamente los sensores. Por otro lado, utiliza un protocolo llamado **1-Wire** que nos permite utilizar varios sensores de forma simultánea conectándolos a un mismo pin del **arduino;** esta característica me fue muy útil ya que la placa de **arduino uno,** que fue la que elegí para realizar esta parte del proyecto, está muy limitada en puertos I/O siendo capaces de identificar la lectura de cada uno de los sensores de forma independiente.

Ilustración 1: 4.2.1

4.2.1.1-. Esquema del montaje

En la Ilustración 1: 4.2.1.1 podemos ver el esquema que nos muestra el montaje solo de este componente, el color rojo para el voltaje, en este caso 5v; el cable negro GND y el amarillo para la línea de datos, también hemos usado una resistencia “pull up” de 4K7 Ohm y 1/4w entre el voltaje y la línea de datos, para el segundo sensor es suficiente solo con puntear.

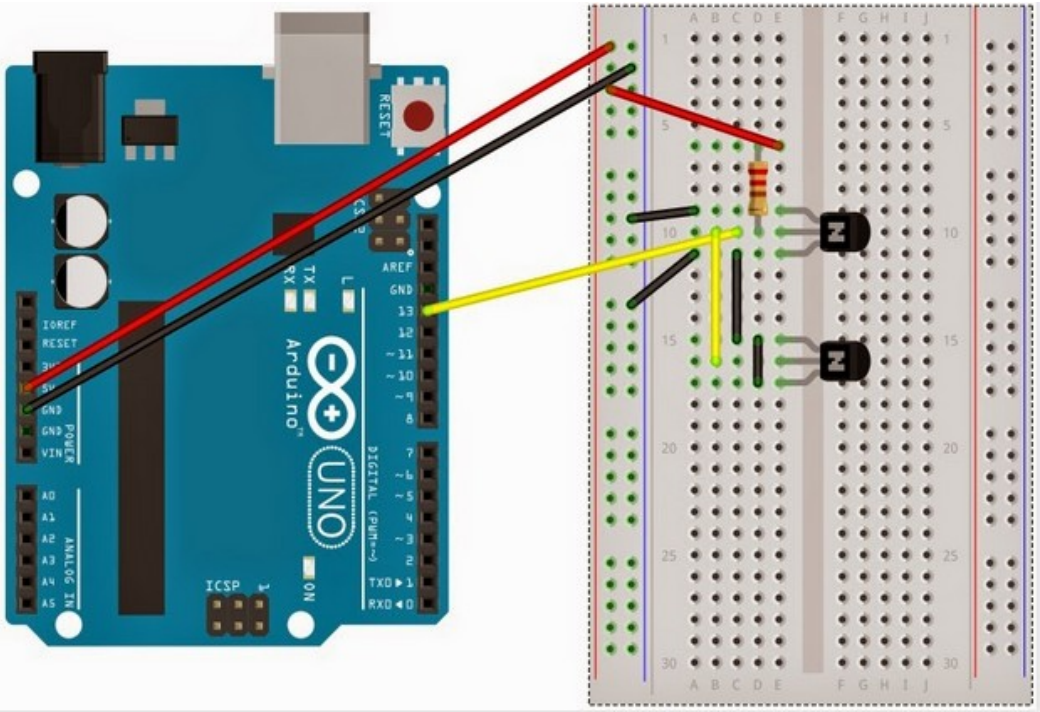


Ilustración 1: 4.2.1.1

4.2.1.2-. Skertch Arduino

Ahora vamos a incluir el código de los sensores de temperatura:

Para utilizar este tipo de sensores vamos a utilizar dos librerías que vamos a insertar en el proyecto mostrado en el código de la Ilustración 1: 4.2.1.2:

- ***1-Wire***: Que la utilizaremos para asignar las direcciones y las lecturas a una variable, realmente no sabemos qué lectura pertenece a qué sensor, así que tendremos que averiguar el orden de colocación de los sensores.
- ***Dallas DS18B20***: Esta es la librería de los sensores DS18B20.

```
#include <DallasTemperature.h>
#include <OneWire.h>
```

Ilustración 1: 4.2.1.2

Definimos el pin que vamos a utilizar del circuito, en este caso será el 13.

Utilizamos **OneWire** y creamos un objeto indicándole el pin que tiene que utilizar para conectarse con el sensor de temperatura.

Creamos una variable del tipo **DallasTemperature** y le asignamos la dirección de memoria de nuestra variable del tipo **OneWire**.

Almacenamos en un *array* del tipo **VariableAddress** las direcciones de nuestros dos sensores, que se almacenarán en la posición 0 y 1 respectivamente.

```
OneWire ourWire(Pin); //Se establece el pin declarado como bus para la comunicación OneWire
DallasTemperature sensors(&ourWire); //Se instancia la librería DallasTemperature
DeviceAddress Sensor;  // Array donde almacenamos la direccion del sensor DS18B20

float TempAguaPecera;
float TempAguaSalidaNevera;
```

Ilustración 2: 4.2.1.2

Por último, creamos un par de variables de tipo *float* para guardar nuestras temperaturas y poder trabajar con ellas.

La Ilustración 3: 4.2.1.2 es el inicio de la función reservada **setup()**, donde se inicia todas la variables.

Iniciamos el Serial con **Serial.begin(9600)**, que inicia la comunicación de serie para tener una idea aproximada de lo que está pasando en nuestro programa, lo iniciamos a 9600, que es el estándar.

También iniciamos nuestra variable **sensors** que es del tipo **DallasTemperature** para iniciar la lectura de nuestros sensores de temperatura.

Ahora, por último, vamos a comprobar que no tenemos ningún fallo en la lectura de nuestros sensores con un condicional, si funciona como es esperado no nos dará ningún mensaje en nuestro puerto de comunicación y se tomarán las lecturas normalmente.

```
void setup() {

  Serial.begin(9600);
  sensors.begin(); //Se inician los sensores
   if (!sensors.getAddress(Sensor, 0))
  //Si no es posible determinar la direccion nos da un mensaje de error
    Serial.println("Imposible encontrar direccion del sensor.");

}
```

Ilustración 3: 4.2.1.2

En la Ilustración 4: 4.2.1.2 empezamos con la función reservada de **loop().**

Con **sensors.RequestTemperatures()** nos preparamos para registrar las temperaturas que hemos realizado su instancia en Ilustración 2: 4.2.1.2 del tipo de **DallasTemperature**.

Por último, las guardamos en las variables que hemos creado en la Ilustración 2: 4.2.1.2 para poder trabajar con ellas en el siguiente apartado.

```
sensors.requestTemperatures(); //Prepara el sensor para la lectura
TempAguaPecera = sensors.getTempCByIndex(0);//Sensor 1 Agua Pecera
TempAguaSalidaNevera = sensors.getTempCByIndex(1);//Sensor 2 Agua del Motor
```

Ilustración 4: 4.2.1.2

4.2.2-. Pantalla de LCD

LCD viene de *Liquid Crystal Display* (Pantallas de cristal líquido), no es una tecnología de última generación, pero para nuestro propósito es más que suficiente.

Las características de este tipo de pantallas es que por medio de unos filtros y las propiedades de la luz polarizada se consigue mostrar una serie de datos por medio de una iluminación de fondo.

Nosotros vamos a trabajar con las típicas LCD 16x2 que se venden en cualquier tienda de electrónica, esto quiere decir que mostraremos datos en la LCD de 2 filas y 16 caracteres por fila.

Ahora vamos a señalar los pines que tienen este tipo de pantallas:

Como vemos en la Ilustración 3: 4.2.2 tenemos la imagen de esta **LCD** por detrás, tenemos 16 pines, de los cuales los 14 primeros son para la visualización de los datos y los dos últimos el 15-16 son para la luz de fondo.

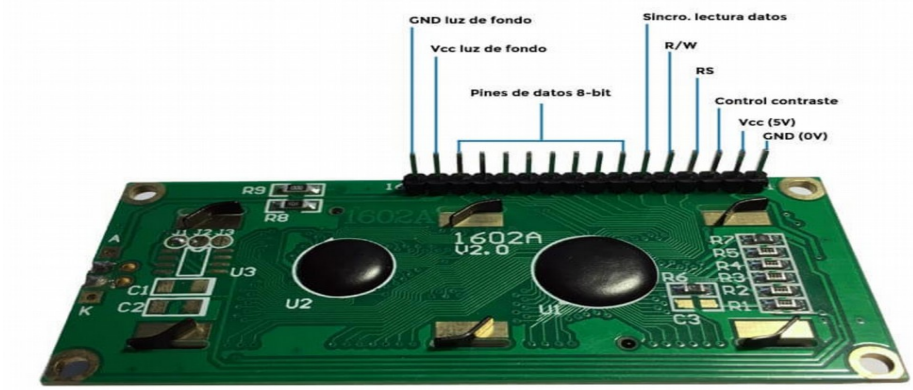


Ilustración 3: 4.2.2

En la Ilustración 4: 4.2.2 vemos una LCD de 16x2 con datos imprimidos en la pantalla.

Ilustración 4: 4.2.2

4.2.2.1-. Esquema de montaje

Vamos a identificar los componentes que necesitamos para este montaje:

- Arduino uno
- Protoboard
- Cables
- Pantalla LCD 16x2
- Potenciómetro
- Resistencia

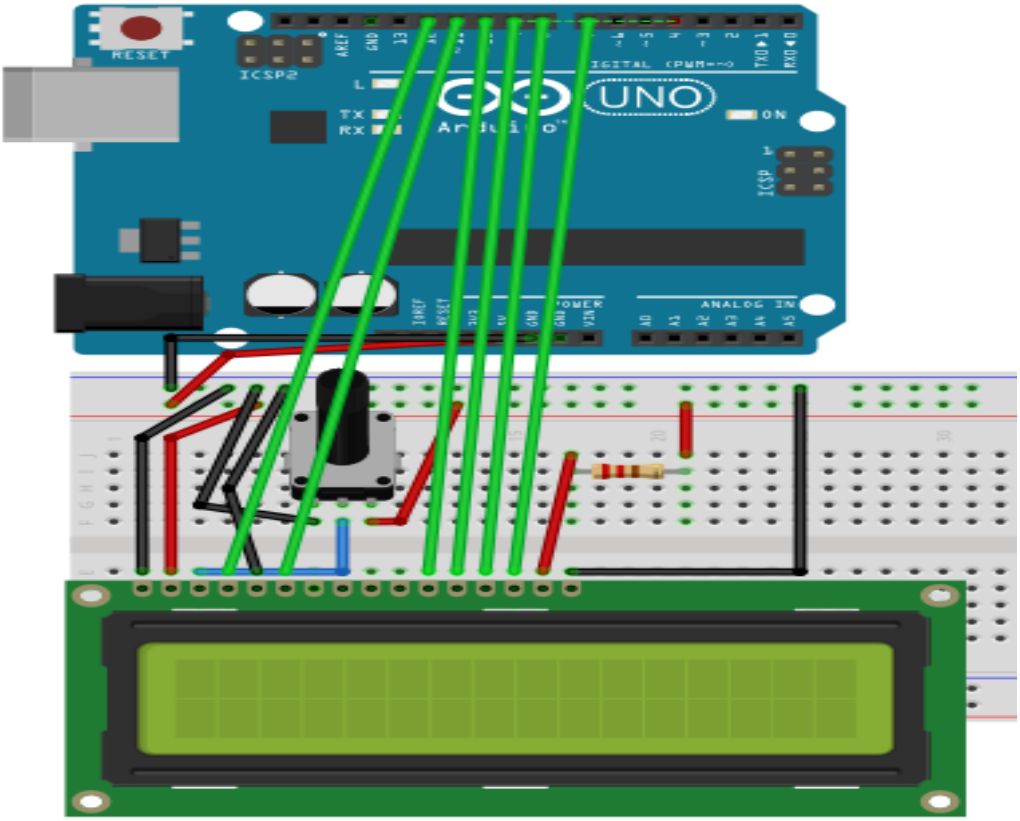


Ilustración 1: 4.2.2.1

4.2.2.2-. Sketch arduino

Para poder utilizar la pantalla LCD vamos a tener que utilizar una librería nativa de arduino, normalmente la tendremos incluida en el entorno, lo único que tenemos que hacer es incluirla con un *define* en el programa de arduino.Ilustración 1: 4.2.2.2

Esta librería no es exclusiva para este tipo de LCD, es genérica puede utilizase para cualquier LCD.

```
#include <LiquidCrystal.h>
```

Ilustración 1: 4.2.2.2

Como debemos asignar las columnas y filas de nuestra **LCD 16x2** (ya que la librería es genérica) que en este caso es de 2 filas (líneas) y 16 columnas (caracteres), debemos definir las constantes que utilizaremos más adelante.

También vamos a definir la velocidad de movimiento porque al tener que poner muchos datos en la **LCD** no nos bastan con 16 caracteres y deberemos mover los caracteres para tener toda la información a la vista.

También vamos a definir los textos que necesitamos para cada caso (agua fría, agua caliente, etc..), todo esto lo vamos a definir en un *array* de 5 dimensiones, después asignaremos las dimensiones a una variable tipo *array* y le añadiremos las frases de información de estado del agua, esta información se mostrará también en la pagina web.

Vamos a definir una variable índice que nos va a indicar el texto a mostrar por la pantalla LCD.

Ahora inicializamos la librería indicando los pines que utilizaremos con una variable de tipo **LiquidCrystal**.

Por último, en la parte de inicio y definición de parámetros publicamos 4 variables de tipo *string* para guardar la información y mostrarla por pantalla. Ilustración 2: 4.2.2.2

```
#define COLS 16 // Columnas del LCD
#define ROWS 2 // Filas del LCD
#define VELOCIDAD 300 // Velocidad para mover el texto
#define ARRAYTXT 5 // Número de textos a escribir

String textos[ARRAYTXT] = {"Muy Caliente", "Caliente", "OK", "Fria", "Muy Fria"};
int indice;

LiquidCrystal lcd(7, 8, 9, 10, 11 , 12);

String primeraFilaAguaAcuario;
String segundaFilaAguaAcuario;
String primeraFilaSalidaAgua;
String segundaFilaSalidaAgua;
```

Ilustración 2: 4.2.2.2

Ahora vamos a la función reservada **setup()**, para iniciar nuestra LCD.

Como siempre iniciamos el serial con **Serial.begin(9600)**, que como ya hemos visto anteriormente, es para hacer ***debug*** en nuestra programa.

Después deberemos inicializar la pantalla LCD con sus dimensiones reales y dentro de la función deberemos poner las dimensiones que hemos definido en la Ilustración 2: 4.2.2.2 con sus respectivas definiciones como muestra la Ilustración 5: 4.2.2.2

```
void setup() {

  Serial.begin(9600);
  lcd.begin(COLS,ROWS);//Iniciaizamos la pantalla LCD

}
```

Ilustración 5: 4.2.2.2

Ahora vamos a rellenar la función **loop()**, para insertar datos y actualizar información en la pantalla **LCD**.

Debemos recordar las variables de las temperaturas guardadas de nuestros sensores de temperatura que son las que vamos a mostrar en nuestra pantalla de LCD. Ilustración 4: 4.2.1.2

En esta parte del código vamos a mostrar los parámetros de la temperatura del agua de la pecera en nuestra pantalla LCD.

Lo primero que vemos es que guardamos información en las variables ya mencionadas anteriormente:

- En la variable **primeraFilaAcurioAgua** vamos a guardar un mensaje que aparecerá en la fila de arriba del LCD, que identifica la temperatura que vamos a mostrar.
- En la variable **segundaFilaAcurioAgua** vamos a guardar la temperatura del sensor que hemos guardado en la variable **TempAguaPecera**(Ilustración 4: 4.2.1.2) para poder mostrarla en la pantalla.
- **Lcd.setCursor(0,0)** y **Lcd.setCursor(0,1)** es una función de la librería de la LCD, a la que le indicamos dónde debe empezar a escribir los datos en la pantalla, en la primera propiedad de la función le indicamos que empiece en la posición ‘0’, o sea, la primera columna y, en la segunda propiedad, le indicamos ‘0’ para que lo coloque en la fila primera; si ponemos ‘1’ lo escribiría en la segunda fila. Así que la **primeraFilaAcurioAgua** se mostrará en la fila 1 y **segundaFilaAcurioAgua** en la segunda.
- **Lcd.print(primeraFilaAcurioAgua), Lcd.print(segundaFilaAcurioAgua)** son funciones de la librería de la LCD que sirve para mostrar por pantalla las variables que tenemos dentro.
- **Delay(5000)** sirve para hacer una pausa de 5 segundos para que le dé tiempo a la pantalla a mostrar los datos, ya que como **loop** de **arduino** va muy rápido no le daría tiempo a mostrar la información y mezclaría todos lo datos.
- Por último, **lcd.clear()** borramos los datos para mostrar la siguiente temperatura y no mezclar los datos. El resultado final está en la Ilustración 7: 4.2.2.2.

Esta explicación es para la temperatura del agua del acuario, pero para la salida del agua de la temperatura de bomba de agua es igual pero con diferentes variables.

```
//Mostramos primera temperatura la del agua de la pecera
primeraFilaAguaAcuario = "Temperatura Agua";
segundaFilaAguaAcuario = String(TempAguaPecera)+ "C";
lcd.setCursor(0,0);
lcd.print(primeraFilaAguaAcuario);
lcd.setCursor(0,1);
lcd.print(segundaFilaAguaAcuario);

delay(5000);
lcd.clear();

//Mostramos segunda temperatura la del agua de del motor
primeraFilaSalidaAgua = "Temp.Salida Agua";
segundaFilaSalidaAgua = String(TempAguaSalidaNevera)+ "C";
lcd.setCursor(0,0);
lcd.print(primeraFilaSalidaAgua);
lcd.setCursor(0,1);
lcd.print(segundaFilaSalidaAgua);
Serial.print(primeraFilaSalidaAgua);
Serial.print(segundaFilaSalidaAgua);
```

Ilustración 6: 4.2.2.2

Ilustración 7: 4.2.2.2

En este paso del programa de LCD vamos a mostrar y activar los mensajes de alarma según la temperatura del agua del acuario, pero no vamos a definir las funciones utilizadas para activar alarmas (lumínicas y sonoras) o cómo realizamos el cambio para calentar o en enfriar el agua; utilizaremos las funciones pero no las explicaremos ahora, sino más adelante, en otra sección.

Para realizar esta tarea vamos a usar condicionales:

- Si la temperatura del agua (**TempAguaPecera**) es mayor de 28ºC activaremos la alarma que iniciará un led y un sonido, para avisar que hemos sobrepasado la temperatura que podría ocasionar la muerte de los seres vivos del acuario, además asignaríamos un ‘0’ a la variable índice para mostrar el valor del *array* ‘**textos**’ (Ilustración 2: 4.2.2.2) que recordemos es donde están introducidos los

mensajes de información. También activaríamos con la función **ServoFrio()** el agua fría para enfriar el agua de la nevera.

- Si la temperatura del agua de la pecera (**TempAguaPecera**) es mayor de 26, solo activaríamos la función **ServoFrio()** para enfriar el agua lentamente para que la fauna del acuario no note los cambios bruscos que tanto afectan a este tipo de acuario. Cambiaríamos el índice a 1 para mostrar el segundo mensaje de información de nuestro *array*.
- Si la temperatura del agua de la pecera (**TempAguaPecera**) es mayor de 24, con la función **ServoConfort()** apagaríamos la nevera-calefactor de agua y se apagaría la bomba de agua porque la temperatura del agua está en rango óptimo. Cambiaríamos el índice a 2 para mostrar el segundo mensaje de información de nuestro *array*.
- Si la temperatura del agua de la pecera (**TempAguaPecera**) es menor de 22, con la función **ServoCalor()** encendería la nevera-calefactor de agua y la bomba de agua para subir la temperatura con el agua caliente. Cambiaríamos el índice a 3 para mostrar el cuarto mensaje de información de nuestro *array*.
- Si la temperatura del agua de la pecera (**TempAguaPecera**) es menor de 20, con la función **ServoCalor()** encendería la nevera-calefactor de agua y la bomba de agua para subir la temperatura con el agua caliente, activaríamos la alarma con la función **Alarma()** y se activaría el led y el sonido para avisar de una situación grave. Cambiaríamos el índice a 4 para mostrar el quinto mensaje de información de nuestro *array*. Todo esto en la Ilustración 9: 4.2.2.2
- Como última acción de este apartado mostraríamos la información por el LCD del estado del acuario los siguientes mensajes: "Muy Caliente", "Caliente", "OK", "Fría", "Muy Fría". Ilustración 8: 4.2.2.2

```
delay(5000);
lcd.clear();
//Escribimos estado del agua
lcd.setCursor(0 ,0);
// Escribimos mensaje
lcd.print("Estado del Agua:");
//Escribimos en la fila de abajo
lcd.setCursor(0 ,1);
// Escribimos mensaje
lcd.print(textos[indice]);
```

Ilustración 8: 4.2.2.2

```
 //Instrucciones para determinar el rango de temperatura
if (TempAguaPecera > 28){
    indice = 0;
    Alarma();
    ServoFrio();
}
if (TempAguaPecera > 26){
    indice = 1;
    ServoFrio();
  }
if (TempAguaPecera > 24){
    indice = 2;
    ServoConfort();
  }
if (TempAguaPecera < 22){
    indice = 3;
    ServoCalor();
  }
if (TempAguaPecera < 20){
    indice = 4;
    Alarma();
    ServoCalor();
  }
```

Ilustración 9: 4.2.2.2

4.2.3-. Sensor de agua

El sensor de agua que vamos a utilizar es muy barato y son de los más utilizados en entornos de arduino, realmente no son de muy buena calidad en lo que respecta a su duración pero las lecturas son correctas, pero en el entorno de una pecera de arrecife (marina) no duran mucho (se oxidan) pero, de todos modos, simplemente lo vamos a utilizar para alertarnos al primer indicio de humedad y así poder detener toda actividad eléctrica.

Uno de estos lo vamos a utilizar para saber si tenemos alguna fuga de agua en la nervera-calefactor, para desconectar la bomba y que no tengamos problemas eléctricos, el otro lo utilizaremos para llenar el acuario cuando baje su nivel de agua y que los peces no noten ningún cambio en su entorno.

Son muy simples, lo único que tenemos que hacer en conectar la tensión 5V al primer pin, GND al segundo pin y una señal analógica al tercer pin que será proporcional a la cantidad de agua detectada. Ilustración 1: 4.2.3

Ilustración 1: 4.2.3

4.2.3.1-. Esquema de montaje

Vamos a identificar los componentes que necesitamos para este montaje:

- Arduino uno
- Cables
- Sensor de agua

En este apartado vamos a conectar este sensor de agua a nuestra placa de **arduino uno** como se muestra en la Ilustración 1: 4.2.3.1, en este montaje obviamos el segundo sensor, la conexión sería similar pero el pin de la señal lo conectaremos en otro puerto diferente de la placa **arduino uno** ya que estos sensores no son **1-wire** (Ilustración 1: 4.2.1) como los sensores de temperatura anteriormente mencionados.

Como advertencia de seguridad debemos tener cuidado con este sensor ya que el agua y la electricidad no son compatibles y podemos provocar un cortocircuito, no tenemos suficiente tensión pero es recomendable tener cierta precaución, debemos tener cuidado de no mojar nuestra **arduino uno**.

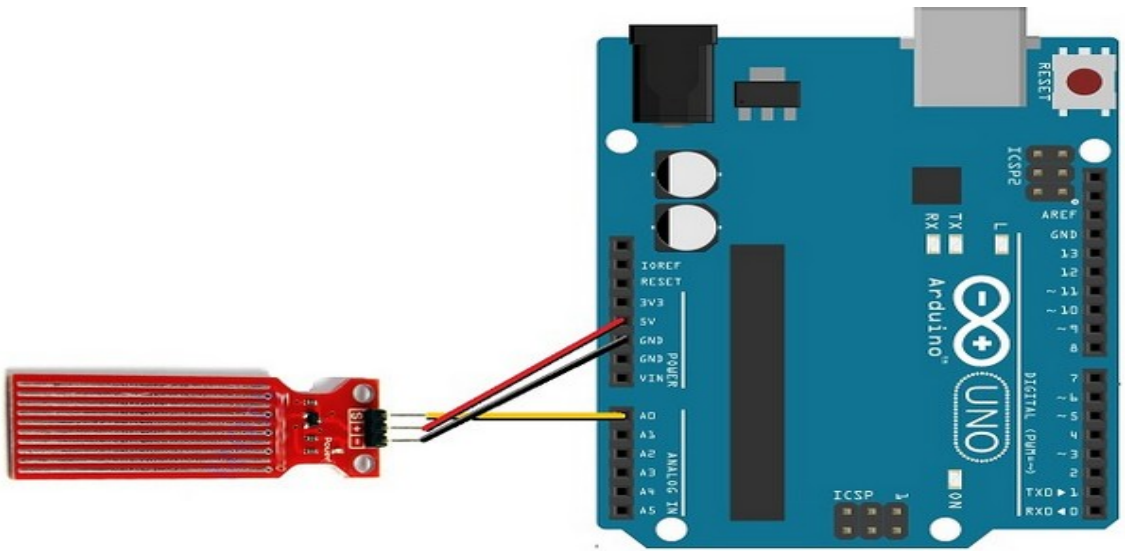


Ilustración 1: 4.2.3.1

4.2.3.2-. Sketch arduino

Para poder utilizar este componente en nuestro programa no utilizaremos ninguna librería, simplemente leeremos la señal analógica, en este caso A4 y A5 donde están conectados y simplemente nos da una señal con un número que cuanto más mayor sea, más agua detecta el valor según el **DataSheet**, va de 0 a 100 donde 100 es la máxima detección de agua.

En la siguiente Ilustración 1: 4.2.3.2 vamos a definir esta variables que nos servirán para guardar los datos de nuestros sensores de agua.

```
//Sensores de agua
int sensorAguaPecera;
int sensorAguaNevera;
int sensorAguaNev_Cal = A4;//pin analogico
int sensorAguaAcuario = A5;//pin analogico
```

Ilustración 1: 4.2.3.2

En la función reservada de **arduino setup()** vamos a iniciar como siempre nuestro serial para poder hacer *debug* para averiguar los posibles fallos.Ilustración 2: 4.2.3.2

```
void setup() {

  Serial.begin(9600);

}
```

Ilustración 2: 4.2.3.2

En la función reservada **loop()** vamos a leer los datos del sensor y los vamos a tratar:

- Lo primero que hacemos es utilizar una función de arduino que es **AnalogRead(pin)** que lee el valor del pin analógico especificado y retornaría un número entero (int) de 0 a 1023, esta lectura es lo que detecta nuestro sensor y lo guardamos en una variable de tipo entera **sensorAguaPecera** que nos informará del estado o datos de nuestro sensor, hacemos lo mismo con la variable del otro sensor. Después, si queremos mostrar que las variables son correctas, la mostramos en el puerto de comunicaciones de nuestra **auduino uno**. Con estos pasos ya podemos trabajar con nuestras variables.Ilustración 2: 4.2.3.2

```
//Desactivo-Activo motores o alimentacion por  alarmas, por precaucion de cortocircuito
sensorAguaPecera = analogRead(sensorAguaAcuario);//este es el de la pecera
sensorAguaNevera = analogRead(sensorAguaNev_Cal);//este es el de la nevera
Serial.print("Sensor Agua nevera: ");
Serial.println(sensorAguaNevera);
Serial.print("Sensor Agua Pecera: ");
Serial.println(sensorAguaPecera);
```

Ilustración 2: 4.2.3.2

Ahora vamos a tratar con los datos que nos dan los sensores de agua, el sensor más importante y que prevalecerá sobre todos los demás es el que monitoriza que no haya ningún tipo de humedad en la nevera, ya que no queremos tener ningún cortocircuito y la menor señal de humedad desactivará todo el sistema por medio de un relé, que está conectado a la red eléctrica y que explicaremos en otro apartado. También añadir que en la nevera-calefactor tenemos un poco de humedad, así que si le ponemos el mínimo de humedad detectable estará siempre parado, por lo que así hemos decidido poner como mínimo 50 de humedad:

- Ahora tenemos un condicional que preguntará a la variable de **sensorAguaNevera** si el sensor de la nevera es mayor de 50, si es mayor cambiará con **digitalWrite()** el estado del pin del relé (**4.2.5-. Relés**) de la nevera a **HIGH** y apagará la bomba, esto desencadena que todos los relés se desconecten por seguridad: el de la bomba de agua y el de llenado de pecera, y activaríamos la alarma con la función de **alarma().** Si no ocurre esta situación, lo dejaríamos todo igual en modo **LOW**, pero preguntando con otro condicional si el sensor del agua del acuario, guardado en **sensorAguaPecera,** indica que el valor es menor de 1 con **digitalWrite()** lo activamos con **LOW** y llenará la pecera a su nivel correcto; si están bien los valores, es decir, son superiores a 1 lo pasará a **HIGH** y dejará desactivado el relé y no se activa la bomba de llenado.Ilustración 3: 4.2.3.2

```
//Si tenemos perdidas de agua, apagamos la bomba o si tenemos humo tambien apagamos
if(sensorAguaNevera > 50 || (MQvalor > 300 && MQValFinal > 1.50)){
  digitalWrite(releNevera, HIGH);
  digitalWrite(releBombaAgua, HIGH);
  digitalWrite(releLlenadoPecera,HIGH);
  Alarma();
}
else{
  digitalWrite(releNevera, LOW);
  digitalWrite(releBombaAgua, LOW);

      if(sensorAguaPecera < 1) // se encenderia la bomba de llenado
    {
      digitalWrite(releLlenadoPecera,LOW);
    }
    else
    {
      digitalWrite(releLlenadoPecera,HIGH);
    }
}
```

Ilustración 3: 4.2.3.2

4.2.4-. Sensor de humo (MQ-2)

Lo sensores **MQ-2** mostrados en la Ilustración 1: 4.2.4 están compuestos por una membrana electro-química que varía su estado al estar en contacto con las sustancias.

Para que la lectura de esta membrana electro-química sea fiable necesitamos que se mantenga con una temperatura elevada, por eso, este tipo de componentes están siempre calientes mientras están en funcionamiento, si no llega a la temperatura de trabajo las lecturas no serán fiables.

El consumo de estos sensores es elevado porque tienen que tener una temperatura de trabajo alta, necesitan 5V y su consumo puede llegar a ser de 800mW.

Estos dispositivos son de alta inercia que quiere decir que para tomar un valor estable necesita tiempo.

Para saber cómo recoger la lectura de este sensor tenemos que consultar su **DataSheet**.

Ilustración 1: 4.2.4

4.2.4.1-. Esquema de montaje

En esta parte vamos a mostrar el montaje en nuestra placa.

Como vemos en la Ilustración 1: 4.2.4.1 tenemos una entrada de tensión de 5V y la tierra GND y dos pines de los cuales vamos a utilizar el A0 que es para señales analógicas y D0 es para las señales digitales.

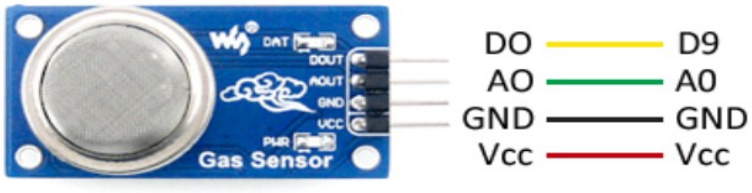


Ilustración 1: 4.2.4.1

Como vemos en la Ilustración 2: 4.2.4.1 es muy sencillo conectar nuestro sensor de humo a nuestra placa.

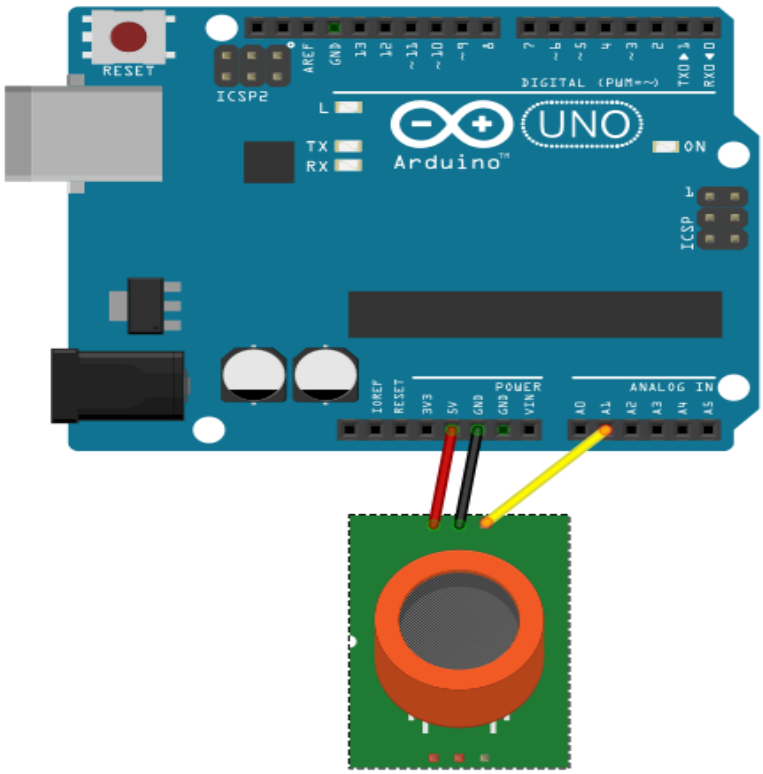


Ilustración 2: 4.2.4.1

4.2.4.2-. Sketch arduino

Al utilizar la entrada analógica para analizar la concentración de humo, tenemos que realizar algunas operaciones matemáticas, según dice su **DataSheet**.

Este sensor recoge la concentración de humo por **ppm** (partes por millón) que permite obtener este dato a partir de la resistencia del sensor y la resistencia media, por eso es necesario conocer la resistencia media de la membrana electro-química del sensor.

Nosotros no vamos a realizar las operaciones matemáticas porque lo que necesitamos es que se detenga todo al menor indicio de humo.

En primer lugar, vamos a crear unas variables, para almacenar los datos y después trabajar con ellas:

- La variable **Mqvalor** es una variable de tipo *integer* en la que guardaremos el valor que nos da el sensor.
- Después, según el **DataSheet** del sensor de **MQ-2,** se nos aconseja que debemos aplicar una pequeña operación, que la guardaremos en una variable de tipo *float* que es **MQValFinal**.
- Por último, le asignamos un pin de la placa que en este caso será el pin analógico A1.Ilustración 1: 4.2.4.2

```
//Variables Sensor de humo
int MQvalor;
float MQValFinal;
const int MQ_PIN = A1;
```

Ilustración 1: 4.2.4.2

Ahora vamos a utilizar el valor **Mqvalor** y **MQValFinal** para detener el motor:

- Ahora tenemos un condicional que preguntará a la variable de **Mqvalor y MQValFinal** si el sensor de humo es mayor de 300 para **Mqvalor** y 1.50 si es **MQValFinal**, si es mayor cambiará con **digitalWrite()** el estado del pin del relé (**4.2.5-. Relés**) de la nevera a **HIGH** y apagaría la bomba, esto desencadena que todos los relés se desconecten por seguridad: el de la bomba de agua y el de llenado de pecera, y activaríamos la alarma con la función de **alarma().** Si no ocurre esta situación, lo dejaríamos todo igual en modo **LOW**, pero preguntando con otro condicional si el sensor del agua del acuario, guardado en **sensorAguaPecera,** indica que el valor es menor de 1 con **digitalWrite()** lo activamos con **LOW** y llenará la pecera a su nivel correcto; si están bien los

valores, es decir, son superiores a 1 lo pasará a **HIGH** y dejará desactivado el relé y no se activa la bomba de llenado.Ilustración 2: 4.2.4.2

```
//Si tenemos perdidas de agua, apagamos la bomba o si tenemos humo tambien apagamos
if(sensorAguaNevera > 50 || (MQvalor > 300 && MQValFinal > 1.50)){
  digitalWrite(releNevera, HIGH);
  digitalWrite(releBombaAgua, HIGH);
  digitalWrite(releLlenadoPecera,HIGH);
  Alarma();
}
else{
  digitalWrite(releNevera, LOW);
  digitalWrite(releBombaAgua, LOW);

      if(sensorAguaPecera < 1) // se encenderia la bomba de llenado
    {
      digitalWrite(releLlenadoPecera,LOW);
    }
    else
    {
      digitalWrite(releLlenadoPecera,HIGH);
    }
}
```

Ilustración 2: 4.2.4.2

4.2.5-. Relés

Un relé es un interruptor con el que activamos una señal eléctrica. El relé esta compuesto por una bobina que al pasar una corriente eléctrica genera un campo electromagnético que hace mover su placa abriendo o cerrando el circuito eléctrico por donde circula la corriente, en nuestro caso 220V para poder controlar los componentes de este proyecto.

En nuestro proyecto tenemos aparatos eléctricos que requiere una tensión que no puede alimentar nuestra placa y requerimos de un componente que se pueda conectar a la red eléctrica y a nuestra **arduino uno** para que la placa controle esa conexión.

Para este proyecto son necesarios tres relés para:

- Bomba de agua nevera-calefacción.
- Alimentación de nevera-calefacción.
- Bomba de agua llenado acuario.

He elegido este tipo de relés por su comodidad, ya tiene todo lo necesario incluido en la placa: el diodo emisor para evitar picos de corriente y foto-transistor para ajustar la corriente. En nuestro proyecto vamos a utilizar este tipo de relés. Ilustración 1: 4.2.5

Ilustración 1: 4.2.5

4.2.5.1-. Esquema de Montaje

Como podemos ver tenemos tres pines (derecha): uno es el de la señal que cambiaría su estado, también tenemos los de Vcc que es el de tensión y también el de tierra GND. A la izquierda tendríamos las conexiones a la red de alta tensión, una conexión común y dos más: una en abierto y otra en cerrado. Ilustración 1: 4.2.5.1

Ilustración 1: 4.2.5.1

En el esquema de montaje necesitaremos: Ilustración 2: 4.2.5.1

- 1 módulo de 4 relés.
- Arduino uno.
- Cables.
- 3 enchufes.

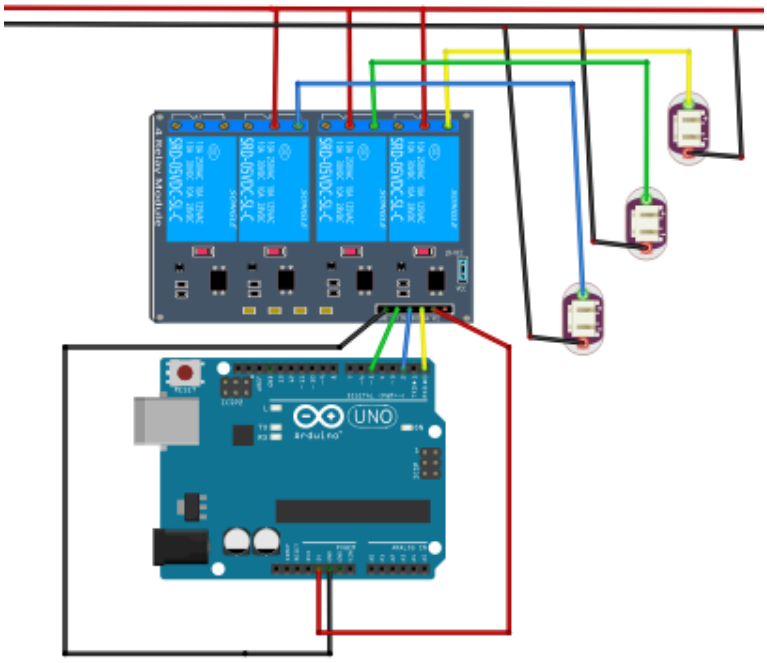


Ilustración 2: 4.2.5.1

4.2.5.2-. Sketch arduino

Antes de implementar la parte de programación referente a los relés en el **sketch**, vamos a explicar los niveles **HIGH, LOW** y las entradas y salidas I**NPUT y OUTPUT**.

Definición de **HIGH y LOW**:

Cuando leemos o escribimos en un pin digital, solo tenemos dos valores posibles **HIGH** que es un 1 y **LOW** que sería un 0.

HIGH:

En referencia a un pin, dependiendo de como esté configurado, puede estar configurado como entrada (**INPUT**) o como salida (**OUTPUT**).

- Si un pin está configurado con **pinMode** en modo **INPUT** y leemos con **digitalRead HIGH** que la tensión es mayor de 5V, esto activará las resistencias **pull up** internas y cambiará el pin y se activará a 1 (**HIGH**).
- Si un pin esta configurado con **pinMode** en modo **OUTPUT** y leemos con **digitalRead HIGH** que la tensión es 5V, entonces este pin actuará como fuente de corriente.

LOW:

En referencia a un pin, según como esté configurado, puede estar configurado como entrada (**INPUT**) o como salida (**OUTPUT**).

- Si un pin esta configurado con **pinMode** en modo **INPUT** y es puesto con **digitalRead** a nivel **LOW,** cuando, la tensión sea menor de 3v en el pin en placas de 5v.
- Si un pin está configurado con **pinMode** en modo **OUTPUT** y leemos con **digitalRead** a nivel **LOW** cuando la tensión es 0V en el pin en placas de 5V, entonces este pin actuará como drenador para las señales débiles.

Definición de I**NPUT y OUTPUT:**

INPUT:

- Los pines configurados en el estado **INPUT** (entrada) están en estado de alta impedancia, básicamente demandan corriente, por esta razón se utilizan para la lectura.

OUTPUT:

- Los pines configurados en el estado **OUTPUT** (salida) están en estado de baja impedancia, básicamente proporcionan corriente, por esta razón se utilizan para la escritura.

Con esta explicación entramos ya de lleno en la parte de la programación de los relés.

Como siempre debemos definir los pines que van a utilizar los relés. Vamos a asignar pines digitales. Ilustración 1: 4.2.5.2

```
//Rele Bomba de agua
const int releBombaAgua = 2;
//Rele2 Nevera
const int releNevera = 5;
//Rele 3 Bonde de agua llenado pecera
const int releLlenadoPecera = 0;
```

Ilustración 1: 4.2.5.2

En función reservada **setup()** vamos a iniciar los pines en modo **OUTPUT**(Salida): Ilustración 2: 4.2.5.2

- **pinMode(releBombaAgua,OUTPUT);**
- **pinMode(releNevera,OUTPUT);**
- **pinMode(releLlenadoPecera,OUTPUT);**
- Iniciamos también el serial para comunicarnos con el puerto **serial** de nuestra placa.

```
Serial.begin(9600);
pinMode(releBombaAgua,OUTPUT); //ajustar el modo digital IO pin
pinMode(releNevera,OUTPUT); //ajustar el modo analog IO pin
pinMode(releLlenadoPecera,OUTPUT); //ajustar el modo analog IO pin
```

Ilustración 2: 4.2.5.2

En la función **loop()** vamos a trabajar con los relés para activar y desactivar la tensión de los diferentes elementos con alimentación externa que no puede alimentar nuestra placa.

Como hemos explicado en la sección 4.2.4.2-. Sketch arduino activaremos o desactivaremos los relés cuando sea necesario. Ilustración 3: 4.2.5.2

```
//Si tenemos perdidas de agua, apagamos la bomba o si tenemos humo tambien apagamos
if(sensorAguaNevera > 50 || (MQvalor > 300 && MQValFinal > 1.50)){
  digitalWrite(releNevera, HIGH);
  digitalWrite(releBombaAgua, HIGH);
  digitalWrite(releLlenadoPecera,HIGH);
  Alarma();
}
else{
  digitalWrite(releNevera, LOW);
  digitalWrite(releBombaAgua, LOW);

      if(sensorAguaPecera < 1) // se encenderia la bomba de llenado
    {
      digitalWrite(releLlenadoPecera,LOW);
    }
    else
    {
      digitalWrite(releLlenadoPecera,HIGH);
    }
}
```

Ilustración 3: 4.2.5.2

4.2.6-. Altavoz y led de alarma

En esta sección de la memoria vamos a exponer los dispositivos que nos van a informar de algún problema en nuestro acuario.

Estas alarmas serán del tipo luminoso (Led) y sonoro (Altavoz).

También vamos a implementar un pulsador para desactivar la alarma sonora en cuanto averigüemos que tenemos un problema en el acuario y no tener la alarma todo el tiempo activada hasta que se equilibren los parámetros, puesto que puede tarda un tiempo, ya que en una acuario de arrecife cambiar los parámetros del agua rápido puede resultar perjudicial para toda la fauna del entorno.

Este led se limita a parpadear con distintos colores como indica el fabricante. Tiene 3 pines: uno GND, Vcc y otro de señal, que es el que conectaremos a nuestra placa. Ilustración 1: 4.2.6

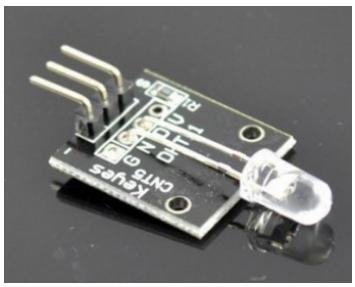


Ilustración 1: 4.2.6

Este pequeño altavoz o zumbador se utiliza para todo tipo de dispositivos. Tiene 3 pines: uno GND, Vcc y otro de señal, que es el que conectaremos a nuestra placa. Ilustración 2: 4.2.6

Ilustración 2: 4.2.6

La resistencia del pulsador la hemos configurado para que el botón esté en **pull down,** entonces fuerza **LOW** cuando el pulsador está abierto, y cuando está cerrado el pin se pone a **HIGH**, y la intensidad que circula es más baja por la resistencia. Para conectar este tipo de pulsadores solo necesitamos 2 patas de la parte más estrecha, una a **vcc** la otra a la resistencia (**pulldown**) y a la señal en el pin de la placa y la otra parte de la resistencia a **GND**. Solo decir que con la resistencia arreglamos el problema del rebote (**bouncing**[1]). Ilustración 3: 4.2.6

Ilustración 3: 4.2.6

1 Por ruido se inhiben o anulan mediante un lapsus de tiempo las señales.

4.2.6.1-. Esquema de montaje

En el esquema de montaje necesitaremos los siguientes dispositivos: Ilustración 1: 4.2.6.1

- Arduino KY-034 Automatic flashing colorful LED module.
- Arduino KY-006 Small passive buzzer module.
- Pulsador switch 12mm
- Cables.
- Arduino Uno.
- ProtoBoard.
- 1 Resistencia 4K.

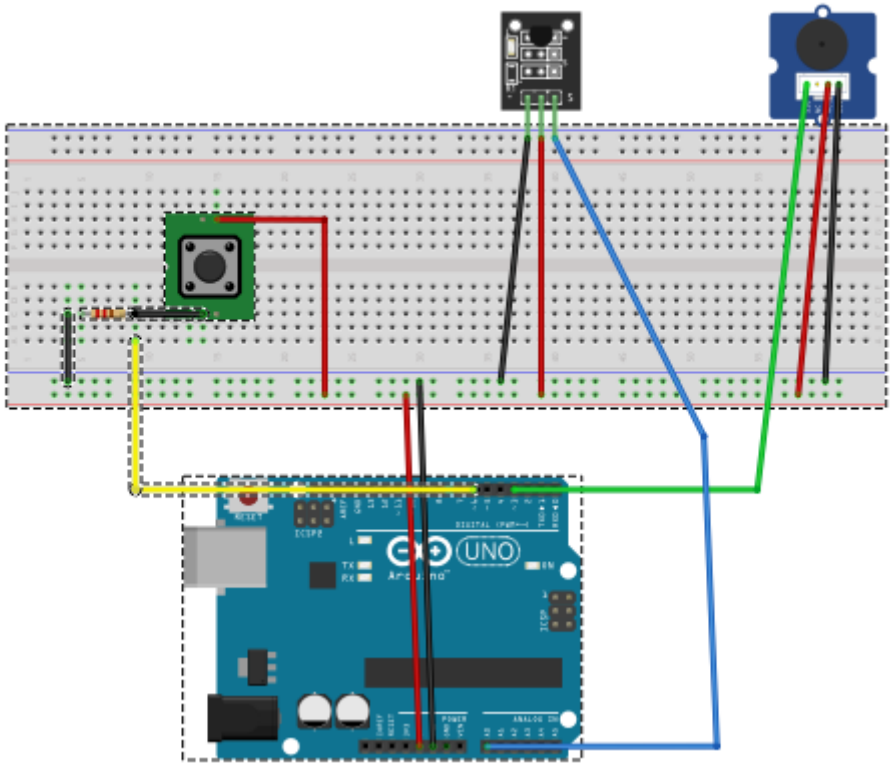


Ilustración 1: 4.2.6.1

4.2.6.2-. Sketch arduino

En esta sección de montaje explicaremos el programa para hacer funcionar estos dispositivos.

En primera Ilustración 1: 4.2.6.2 estamos definiendo variables:

- A las variables **buzzer** y **botonPin** de tipo *integer* se les asignan los pines 3 y 6 de la placa.
- Después tenemos unas variables (**val,old_val y state**) que nos servirán para transformar este pulsador en un interruptor por software.
- Por último, a la variable **ledPin** le asignaremos la variable A0 analógica de la placa.

```
//Sonido
const int buzzer = 3 ;// Pin del altavoz o zumbador
//Boton quitar sonido
const int botonPin = 6;
int val = 0; //val se emplea para almacenar el estado del boton
int state = 0; // 0 LED apagado, mientras que 1 encendido
int old_val = 0; // almacena el antiguo valor de val
//Led Alarma
const int ledPin1 = A0;
```

Ilustración 1: 4.2.6.2

Ahora en la sección de la función reservada de **setup()** iniciaremos los pines en los modos que nos interesen: Ilustración 2: 4.2.6.2

- Para el led en modo **OUTPUT** (salida).
- Para el pulsador **INPUT** (entrada).
- Para altavoz **OUTPUT** (salida).

Estos modos están explicados en la sección Definición de INPUT y OUTPUT:

```
pinMode(ledPin1,OUTPUT);// ajustar el modo digital IO pin
pinMode(botonPin, INPUT);//ajustar el modo digital IO pin
pinMode (buzzer, OUTPUT) ;// ajustar el modo digital IO pin
```

Ilustración 2: 4.2.6.2

En esta parte del desarrollo vamos ha desarrollar la función de **Alarma()** ya mencionada en secciones anteriores pero explicada aquí. Ilustración 3: 4.2.6.2

Para que nuestro altavoz emita un sonido tenemos que darle una frecuencia en este caso le hemos dado 150 repeticiones por medio de un **for**.

Para que emita un sonido, tenemos que cambiar su pin con **digitaWrite** a **HIGH** y claro para que no se caliente y se averie el altavoz deberemos desactivarlo **digitalWrite** a **LOW** y que para no haya **lag**[2] deberemos dar una pausa entre los **digitalWrite** con **delay(1)**.

Con el led debemos hacer la misma acción que con el sonido.

```
//Alarma por si sube mucho la temperatura o baja
    void Alarma(){
     for (int i = 0; i <150; i++) // Frecuencia
                    {
                      if(state == 1)
                      {
                      digitalWrite (buzzer, HIGH) ;// Sonido
                      delay (1) ;// delay 2ms
                      digitalWrite (buzzer, LOW) ;// No Sonido
                      delay (1) ;// delay 2ms
                      }
                    }
      digitalWrite (ledPin1, HIGH); // set the LED on
      delay (2000); // wait for a second
      digitalWrite (ledPin1, LOW); // set the LED off
      delay (2000); // wait for a second

    }
```

Ilustración 3: 4.2.6.2

Para trabajar con el pulsador y transformarlo en un interruptor, solo necesitamos crear una variable que le guarde el valor cuando pulsamos, Lo haremos de forma que cuando pulsamos la variable pasará de 0 a 1 o viceversa dependiendo del estado en el que se encuentre. Después encenderemos el led partiendo del estado de esta variable, no del pulsador en sí. De esta forma hemos convertido un pulsador en un interruptor. Esta es la función de **ParарAlarma()**. Ilustración 4: 4.2.6.2

2 Tiempo de respuesta del circuito.

```
void ParararAlarma(){
  //Boton
  val= digitalRead(botonPin); // lee el estado del Boton
  if ((val == HIGH) && (old_val == LOW))
  {
    state=1-state;
    delay(10);
  }
    old_val = val; // valor del antiguo estado
}
```

Ilustración 4: 4.2.6.2

- En la función **loop()** debes agregar **ParaAlarma()** para que cuando pulsemos el pulsador se pare la alarma. Ilustración 5: 4.2.6.2

```
ParararAlarma();
```

Ilustración 5: 4.2.6.2

4.2.7- Servo Motor para frío, calor o neutro.

Para la configuración del método que cambia los diferentes estados de tratamiento del agua en el enfriador-calefactor hemos optado por un motor para cambiar el interruptor de apagado a frío o calor.

Como vemos en la imagen, Ilustración 1: 4.2.7, hemos fijado en una zona resistente del enfriador-calefactor el motor y hemos fijado un brazo del rotor al interruptor y, al hacer girar al rotor varios grados positivos o negativos, pasamos de apagado a caliente o a frío según los requerimientos de temperatura del agua del acuario, siempre pasando por apagado y realizando una pausa para que no se averíe el enfriador-calefactor de un estado a otro por recomendación del fabricante.

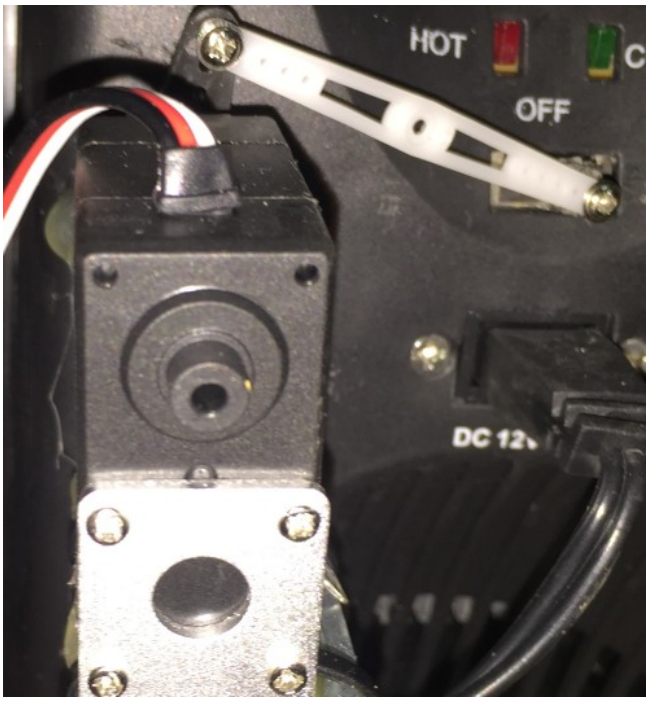


Ilustración 1: 4.2.7

4.2.7.1-. Esquema de montaje

En este montaje necesitamos los siguientes elementos:

- Motor RDS3115M6 5v
- Cables.
- Arduino Uno.
- 2 ejes de rueda.
- 3 tornillos.

Estos motores funcionan con una señal **PWN**, con un pulso de 1ms. entre 2ms. y con un periodo de 20ms., esto quiere decir que es la velocidad máxima que tiene este motor. De todos modos esto no es muy relevante porque el motor va a realizar pequeños desplazamientos en un sentido u otro para mover el interruptor del enfriador-calefactor

Como vemos en la Ilustración 1: 4.2.7.1, el montaje del motor en nuestra placa se realiza conectando el **Vcc** al **pin** de **5V**, el cable negro de masa al pin **GND** y el cable de señal del motor al **pin 4** digital de la placa **arduino uno**.

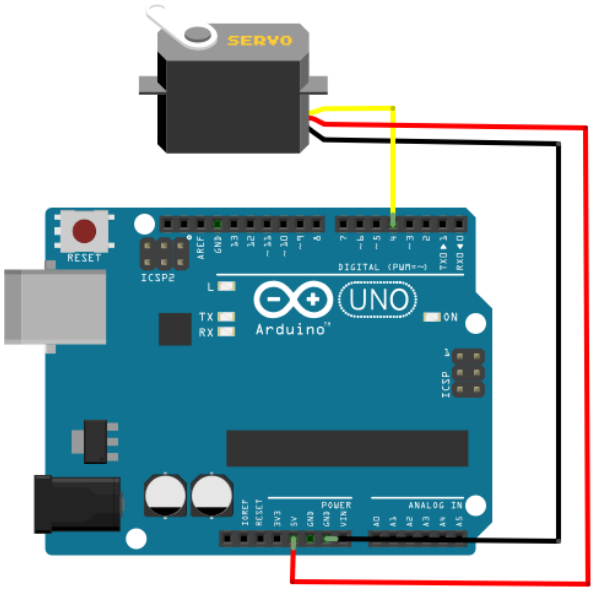


Ilustración 1: 4.2.7.1

4.2.7.2-. Sketch arduino

Lo primero que vamos a hacer es incluir la librería, si no la tenemos la instalamos con el instalador del **IDE de arduino**, este paso se realizará automáticamente. Ilustración 1: 4.2.7.2

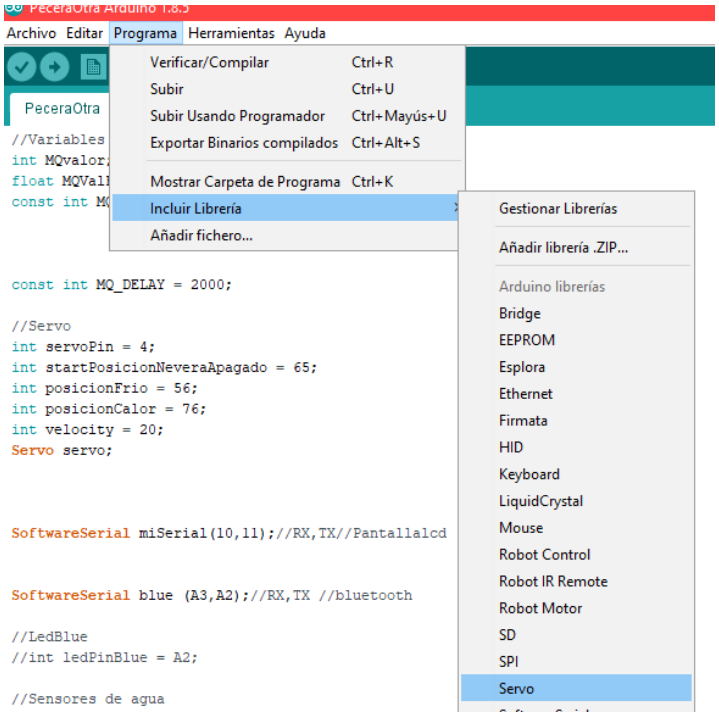


Ilustración 1: 4.2.7.2

Continuamos incluyendo la librería en nuestro programa. Ilustración 2: 4.2.7.2

```
//Libreria Servo
#include <Servo.h>
```

Ilustración 2: 4.2.7.2

En esta parte de nuestro código, definiremos nuestras variables para poder trabajar con nuestro motor. Ilustración 3: 4.2.7.2

- Tenemos la variable **servoPin** que es un entero que asignamos al pin de nuestra placa.
- A la variable **startPosicionNeveraApagado,** que es de tipo entero, le asignamos la posición inicial del servomotor, en este caso es 65°; no lo ponemos a 90° porque,

como hemos visto en la imagen de la instalación, el motor está al lado izquierdo del interruptor y tenemos que corregir ese error (lo hemos fijado ahí porque era una zona fija).

- A la variable **posicionFrio** de tipo *int* le indicamos la posición en grados que tiene que girar el rotor para que el interruptor ponga el enfriador-calefactor en modo frío.
- A la variable **posicionCalor** de tipo *int* le indicamos la posición en grados que tiene que girar el rotor para que el interruptor ponga el enfriador-calefactor en modo calor.
- Finalmente hemos definido la variable de tipo servo para poder trabajar con el motor.

```
//Servo
int servoPin = 4;
int startPosicionNeveraApagado = 65;
int posicionFrio = 56;
int posicionCalor = 76;
Servo servo;
```

Ilustración 3: 4.2.7.2

Iniciamos el servo con la función **servo.attach(servoPin)** que será iniciado en el pin indicado, en este caso el 4. Ilustración 4: 4.2.7.2

```
servo.attach(servoPin);
ServoConfort();
```

Ilustración 4: 4.2.7.2

Tenemos las funciones que cambiarán el modo de trabajar al enfriador-calefactor y son:

- **ServoFrio()**: pasar al modo enfriar.
- **ServoCalor()**: pasar al modo calentar.
- **ServoConfort():** para parar el enfriador-calefactor.

Las tres funciones son iguales lo único que cambia es la posición del rotor con las variables: **posicionCalor, PosicionFrio y startPosicionNeveraApagado**. Ilustración 5: 4.2.7.2

```
void ServoFrio(){

   servo.write(posicionCalor);
   delay(5000);
}

void ServoCalor(){

    servo.write(posicionFrio);
    delay(5000);

}

void ServoConfort(){
    servo.write(startPosicionNeveraApagado);
    delay(5000);
}
```

Ilustración 5: 4.2.7.2

El siguiente código que aparece Ilustración 6: 4.2.7.2 ya lo hemos explicado en la sección Ilustración 9: 4.2.2.2.

```
 //Instrucciones para determinar el rango de temperatura
if (TempAguaPecera > 28){
    indice = 0;
    Alarma();
    ServoFrio();
}
if (TempAguaPecera > 26){
    indice = 1;
    ServoFrio();
  }
if (TempAguaPecera > 24){
    indice = 2;
    ServoConfort();
  }
if (TempAguaPecera < 22){
    indice = 3;
    ServoCalor();
  }
if (TempAguaPecera < 20){
    indice = 4;
    Alarma();
    ServoCalor();
  }
```

Ilustración 6: 4.2.7.2

4.2.8-. Transmisión Bluetooth

La transmisión de datos de la temperatura la vamos a realizar con el protocolo bluetooth 2.0 para guardar los datos en el servidor y mostrarlos en web.

Vamos a utilizar el módulo bluetooth HC-06 con las siguientes características: Ilustración 1: 4.2.8

- Especificación bluetooth v2.0 + EDR (Enhanced Data Rate)
- Modo esclavo (Solo puede operar en este modo)
- Puede configurarse mediante comandos AT (Deben escribirse en mayúscula)
- Chip de radio: CSR BC417143
- Frecuencia: 2.4 GHz, banda ISM
- Modulación: GFSK (Gaussian Frequency Shift Keying)
- Antena de PCB incorporada
- Potencia de emisión: ≤ 6 dBm, Clase 2
- Alcance 5 m a 10 m
- Sensibilidad: ≤ -80 dBm a 0.1% BER
- Velocidad: Asincrónica: 2 Mbps (max.)/160 kbps, sincrónica: 1 Mbps/1 Mbps
- Seguridad: Autenticación y encriptación (Password por defecto: 1234)
- Perfiles: Puerto serial Bluetooth
- Módulo montado en una tarjeta con regulador de voltaje y 4 pines suministrando acceso a VCC, GND, TXD, y RXD
- Consumo de corriente: 30 mA a 40 mA
- Voltaje de operación: 3.6 V a 6 V
- Dimensiones totales: 1.7 cm x 4 cm aprox.
- Temperatura de operación: -25 ºC a +75 º

Como características especiales resaltaremos que es un módulo que trabaja solo en modo esclavo de comunicación y que la información se transmite de forma encriptada.

Ilustración 1: 4.2.8

4.2.8.1-. Esquema de montaje

Para este montaje serán necesarios los siguientes componentes:

- Arduino Uno.
- Cables.
- Modulo esclavo bluetooth HC-06.

En el esquema vemos como conectamos el módulo bluetooth a la placa **arduino uno**: Ilustración 1: 4.2.8.1, que realizamos del siguiente modo:

- conectamos el cable de alimentación de la **HC-06** de 3,3V al pin de tensión de la placa.
- El pin de **GND** lo conectamos al pin de **GND** de nuestra placa.
- El pin de salida **TX** lo conectamos al pin de entrada que hemos asignado a nuestra placa **arduino** A3 como entrada de datos del serial.
- El pin de salida **RX** lo conectamos al pin de salida que hemos asignado a nuestra placa **arduino** A2 como salida de datos del serial.

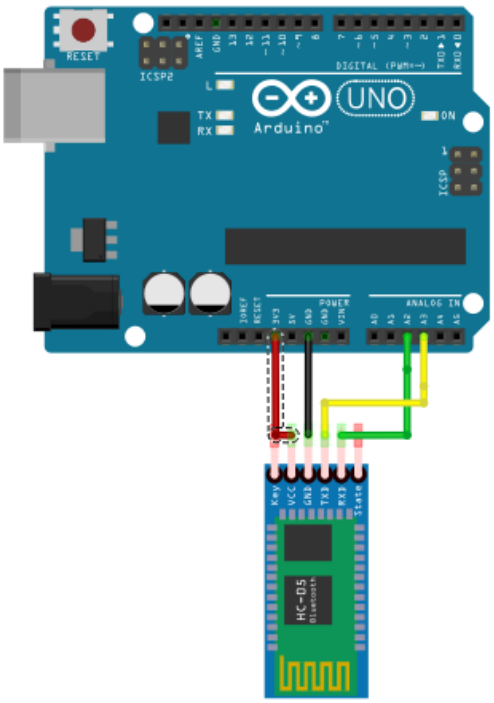


Ilustración 1: 4.2.8.1

4.2.8.2-. Sketch arduino

En esta fase de la metodología de la configuración vamos a programar cómo enviar la información de nuestros sensores a la **raspberry pi** (servidor) por bluetooth para mostrar los datos.

Lo primero es incluir la librería **SoftwareSerial**, esta librería tiene soporte incorporado para comunicación serial en los pines 0 y 1. El soporte serial nativo ocurre a través de una pieza integrada en el chip llamada UART[3]. Este hardware permite que el chip **Atmega** reciba comunicación en serie incluso mientras se trabaja en otras tareas. Ilustración 1: 4.2.8.2

```
#include <SoftwareSerial.h>
```

Ilustración 1: 4.2.8.2

Definimos la variable de tipo **SoftwareSerial** que llamaremos **blue** y asignaremos los pines de salida (**TX**) con **A2** y los de entrada (**RX**) **A3** para poder pasar datos del módulo **bluetooth** a la placa arduino. Ilustración 2: 4.2.8.2

3 Es un dispositivo de hardware para comunicación **asíncrona** en el que el formato de datos y las velocidades de transmisión son configurables.

```
SoftwareSerial blue (A3,A2);//RX,TX //bluetooth
```

Ilustración 2: 4.2.8.2

Para comunicarnos con él, necesitamos configurar nuestro serial a **9600,** que es la configuración que trae por defecto nuestro módulo bluetooth, así que lo iniciaremos con **blue.begin(9600)**; Ilustración 3: 4.2.8.2

```
blue.begin(9600);//Serial del Bluethoot
```

Ilustración 3: 4.2.8.2

Ahora vamos a enviar las temperaturas de nuestros sensores recordando que son: **TempAguaPecera**, **TempAguaSalidaNevera** y el índice que indica el estados del agua, los vamos a guardar en una variable del tipo *string* con la variable **volatile**[4] (*cualificadora*): cuando queremos leer la variable directamente de la memoria **ram**, no de los registros.

Después de definir nuestra variable procedemos a enviar nuestros datos de las temperaturas.

Para enviar los datos al servidor hemos optado por darle un formato fijo de 13 caracteres, que básicamente es lo que necesitamos: 5 caracteres para la temperatura del agua del acuario, separada por un espacio; 5 caracteres para la temperatura de la salida del agua y un espacio para el índice del estado del agua, que se guardará en la base de datos del servidor, así que el formato final sera “00.00 00.00 0” con un total de 13, si es más grande la cadena de caracteres, mandará “00.00 00.00 4” y se tomará por un error en el servidor.

Todo esto lo comprobaremos con un condicional, asegurándonos de que lo que hemos guardado en nuestra variable tiene menos de 13; esto será asegurado por la función **blueEnvio.length()** que comprueba el número de letras que tiene la cadena de caracteres **blueEnvio.**

Si todo es correcto enviará con la función de **SoftwareSerial blue.println(blueEnvio)** a nuestro servidor todos los datos necesarios para tratar los datos de la temperaturas. Ilustración 4: 4.2.8.2

4 Para modificar la manera en que el compilador y el programa deben tratar a esta variable.

```
volatile String blueEnvio = String(TempAguaPecera) + " "
                          + String(TempAguaSalidaNevera) + " "
                          + String(indice);//Meto en variable

if(blueEnvio.length() > 13){//compruevo si esta bien
  blue.println("00.00 00.00 4");
}else{
  blue.println(blueEnvio);
}
```

Ilustración 4: 4.2.8.2

4.2.9-. Instalación de circuito de agua calefacción - refrigeración.

La instalación de este circuito de agua es muy básica. Tenemos que crear un circuito cerrado que recoja el agua del acuario por medio de la bomba de agua y que conecte el tubo transparente a los disipadores, que estarán conectados en serie por los tubos, y para terminar, el último disipador llevará su salida otra vez por medio de los tubos en dirección al acuario y así terminará el circuito depositando otra vez el agua al acuario con otra temperatura.

Pero para calentar o enfriar el agua debemos insertar los disipadores dentro del enfriador-calefactor para que caliente o enfrié el agua según convenga.

4.2.9.1-. Componentes necesarios

Los componentes que necesitamos son:

- Bomba de agua: Ilustración 1: 4.2.9.1
  - Potencia: 7W
  - Voltaje: AC220 ~ 240V
  - Material: Plástico
  - Máxima ascensión: 2.0m

Ilustración 1: 4.2.9.1

- Disipador CPU: Ilustración 2: 4.2.9.1
  - Material: Aluminio.
  - Dimensiones: 40x40x12mm
  - Espesor interno: 5mm.

Ilustración 2: 4.2.9.1

- Refrigeración - Calefacción eléctrico: Ilustración 3: 4.2.9.1
  - Capacidad: 6L.
  - Refrigeración: 15ºC.
  - Calefacción: 60ºC.
  - Material: PVC.

Ilustración 3: 4.2.9.1

- Tubo transparente: Ilustración 4: 4.2.9.1
  - Material: PVC
  - Diámetro del tubo: 3mm

Ilustración 4: 4.2.9.1

4.3-. Metodología Servidor (RaspBerry pi)

En esta parte del proyecto vamos a instalar el sistema operativo que básicamente es descargar de la web de **raspberry,** una compilación que nosotros hemos elegido y que se denomina **RaspBian Lite**[5]**,** ya que va a ser un servidor y no necesitamos escritorio.

Es un sistema operativo basado en GNU/Linux.

También vamos a instalar el paquete **LAMP** que tiene el famoso servidor web **Apache**, el servidor de Bases de datos **Mysql** y el servidor **PHP**, que son las herramientas necesarias para controlar nuestro sistema de climatización.

El sitio web tendrá botones para conectar algunas funciones típicas de los termostatos, como apagado/encendido manual y automático por temperatura.

4.3.1-. Instalación de Sistema Operativo.

En este paso vamos a instalar el sistema operativo de la **raspberry pi** de nuestro servidor.

Vamos a grabar la imagen desde Linux Mint 18.

Para instalar el SO seguiremos estos pasos:

- Insertaremos una **microsd** de 4gb recomendada por **raspberry** en nuestro lector de tarjetas.
- Abriremos el terminal de Linux y escribiremos **lsblk** que es un comando de linux que nos muestra las unidades y la particiones de tu pc.
    - Como vemos en la Ilustración 1: 4.3.1 tenemos dos unidades: la principal sda que es el disco de sistema y sdb que es nuestra sd.

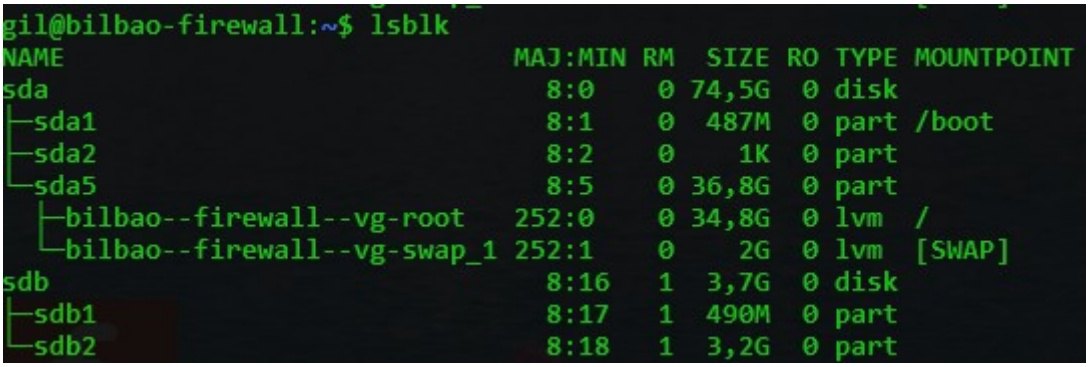


Ilustración 1: 4.3.1

- Localizada nuestra **sd** buscamos nuestra imagen del sistema operativo que nos hemos descargado de la página de **raspberry pi,** descomprimimos el archivo con **unzip** y tendremos el archivo de imagen. Ilustración 2: 4.3.1

5 Web de descarga en bibliografía.

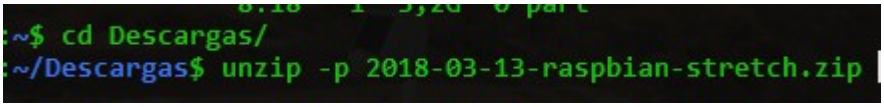


Ilustración 2: 4.3.1

- Ya tenemos la imagen descomprimida que debería ser de un formato similar al de este archivo "**2018-03-13-raspbian-stretch.img**".
- El siguiente paso es el de copiar la imagen del sistema operativo a nuestra sd con el comando de linux **dd** que es un comando que duplica datos cuya sintaxis es:
  - **sudo dd if=origen of=destino:**
    - Donde **origen** es: nuestra imagen **2018-03-13-raspbian-stretch.img.**
    - Donde **destino** es: **/dev/sdb** que es nuestra unidad microsd.
  - Así que el comando definitivo quedaría así: Ilustración 3: 4.3.1
    - **dd if=2018-03-13-raspbian-stretch.img of=/dev/sdb**

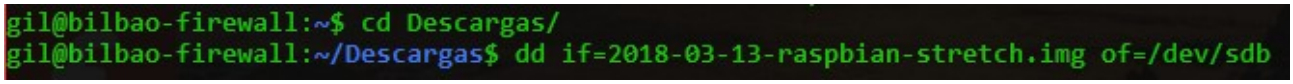


Ilustración 3: 4.3.1

  - Solo tendríamos que esperar a que termine la tarea.
  - Por último, insertar la **sd** con el nuevo sistema operativo en la **raspberry pi**.

4.3.2.- Instalación de programas.

En este apartado vamos a instalar la **BBDD**[6] (Mysql), **php** (servidor PHP) y **Apache** (servidor web).

- Accederemos remotamente a este sistema operativo recientemente instalado por ssh (**secure shell**) con el siguiente comando: abrimos terminal de nuestro **linux** y tecleamos ***ssh -C dirección_Red_Servidor -l pi*** entonces nos preguntará la contraseña. Por defecto, en la instalación, ***pi*** es el usuario y el **password** es

6 Bases de datos.

raspberry que son los parámetros para los usuarios por defecto del nuevo sistema operativo. Ilustración 1: 4.3.2

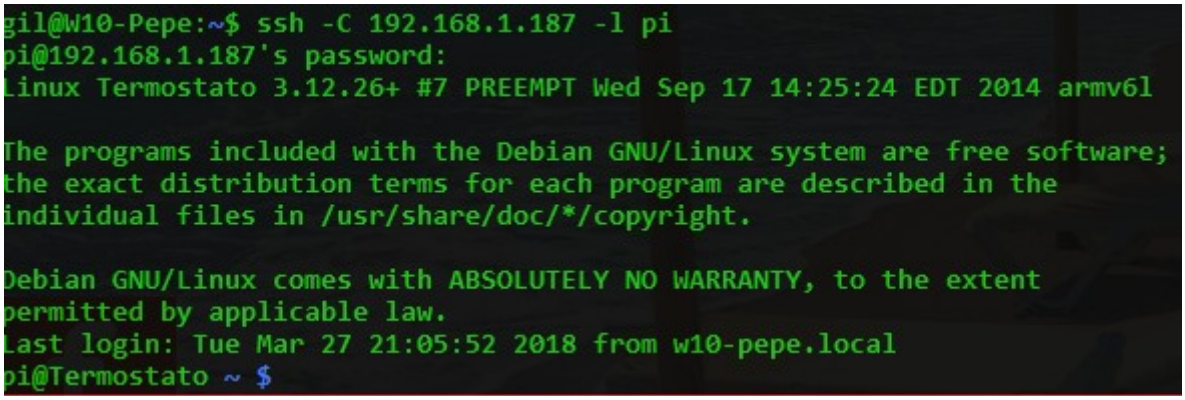


Ilustración 1: 4.3.2

- Ahora ya estamos en nuestro servidor, ahora vamos a empezar a instalar todos los paquetes requeridos para que funcione bien y podamos acceder a todo lo necesario. Lo instalaremos con **apt-get** que es el comando de instalación de paquetes de **distros** basadas en **debian**.
    - Lo primero es instalar los paquetes **python-pid** y **python-mysqldb** para conectar **python** con las bases de datos **mysql.**

      ```
      pi@Termostato ~ $ sudo apt-get install python-pip python-mysqldb
      ```

    - Ahora con el instalador de librerías y paquetes de **python pip** vamos aa instalar **request** y **pymysql.** La primera de estas librerías sirve para ofrecer soporte http y la segunda, para conectar con el servidor de bases de datos **mysql**.

      ```
      pi@Termostato ~ $ sudo pip install requests pymysql
      ```

    - Por último, vamos a instalar el servidor de base de datos **mysql-server, mysql-client** que nos servirá para mantener la base de datos mysql y **php5-mysql** y el módulo **php5** para conectar con bases de datos **mysql**. En la instalación de **mysql-server,** el instalador nos pedirá que le indiquemos una contraseña de la base de datos; esta misma contraseña es la que necesitaremos para realizar cualquier cambio en nuestra base de datos. Con todo esto ya tenemos instalado el servidor **LAMP**.

      ```
      pi@Termostato ~ $ sudo apt-get install mysql-server mysql-client php5-mysql
      ```

- Por último, instalar la librería serial de **python** que nos será de utilidad para comunicarnos con el **arduino uno** por **bluetooth** más adelante.

```
pi@Termostato ~ $ sudo apt-get install python-serial
```

- En este paso vamos a crear la base de datos con las tablas y los campos de las tablas que necesitamos para poder controlar nuestra climatización en el servidor. Todo lo vamos a realizar con una aplicación llamada **Mysql WorkBench** cuya función es configurar de forma gráfica las bases de datos **mysql**.
  - Crearemos la conexión de nuestra aplicación al servidor. Iniciamos nuestro programa **Mysql WorkBench** buscamos en el centro de la pantalla una imagen como la Ilustración 2: 4.3.2, entonces se iniciará un asistente que nos guiará en la configuración de nuestra conexión a la base de datos.

Ilustración 2: 4.3.2

  - Se iniciará el asistente y rellenaremos los datos igual que en la plantilla de la Ilustración 3: 4.3.2 con la definición de cada una de las casillas por cuadrados de colores:
    - Rectángulo azul: El nombre de la conexión para distinguirla de otras conexiones.
    - Rectángulo rojo: Tipo de conexión; nosotros utilizaremos la **TCP/IP** por túnel **secure shell.**
    - Rectángulo verde: Aquí tendremos que poner la **ip** de nuestro servidor añadiendo con los ‘:’ el puerto de acceso a nuestra máquina en este caso el 22.
    - Rectángulo amarillo: Aquí insertaremos el usuario que da acceso a nuestro servidor que en nuestro caso será pi.
    - Rectángulo violeta: Aquí, si queremos le podemos agregar la contraseña de nuestro servidor o servicio **mysql.**
    - Rectángulo cyan: Es la conexión a la ip de **mysql** que en este caso es **127.0.0.1** (**localhost**), que es un método de conexión al propio servidor.
    - Rectángulo naranja: Es el **puerto** o **socket** de conexión a la base de datos **mysql**. Es el puerto por defecto.

- Rectángulo lima: es el usuario administrador de la base de datos que normalmente se llamará **root**.
- Para terminar la conexión y guardarla pulsamos ok.

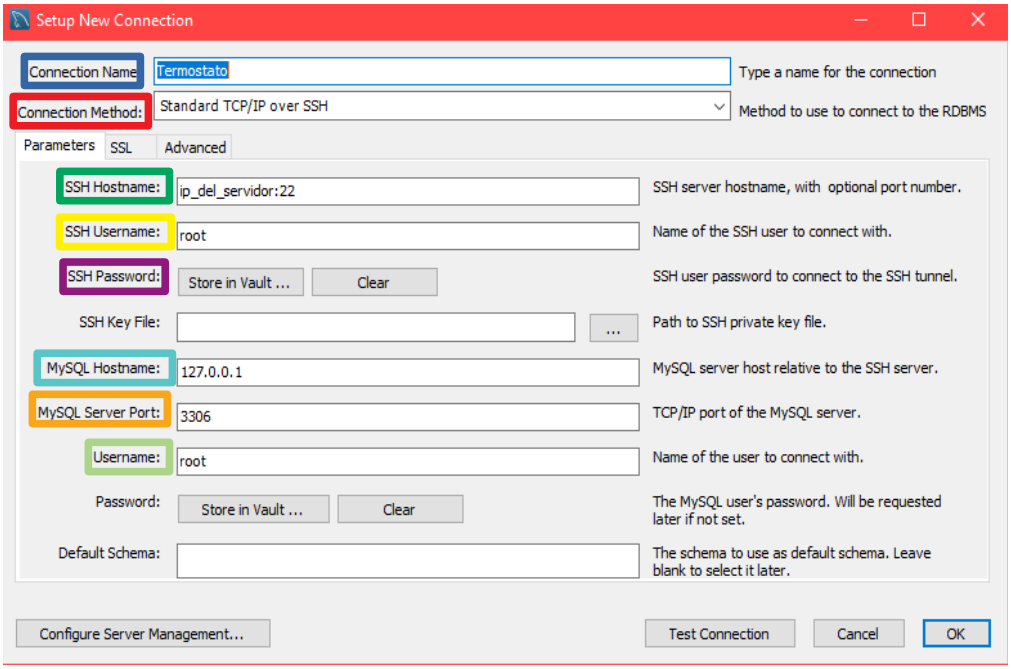


Ilustración 3: 4.3.2

- Ahora tenemos que hacer doble clic en la conexión(Ilustración 4: 4.3.2) que acabamos de configurar.

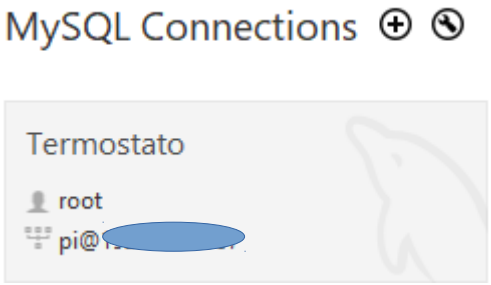


Ilustración 4: 4.3.2

- Con esta acción abriremos una pestaña con la configuración de nuestro servidor **mysql**, con los parámetros mínimos para que funcione. Ilustración 5: 4.3.2

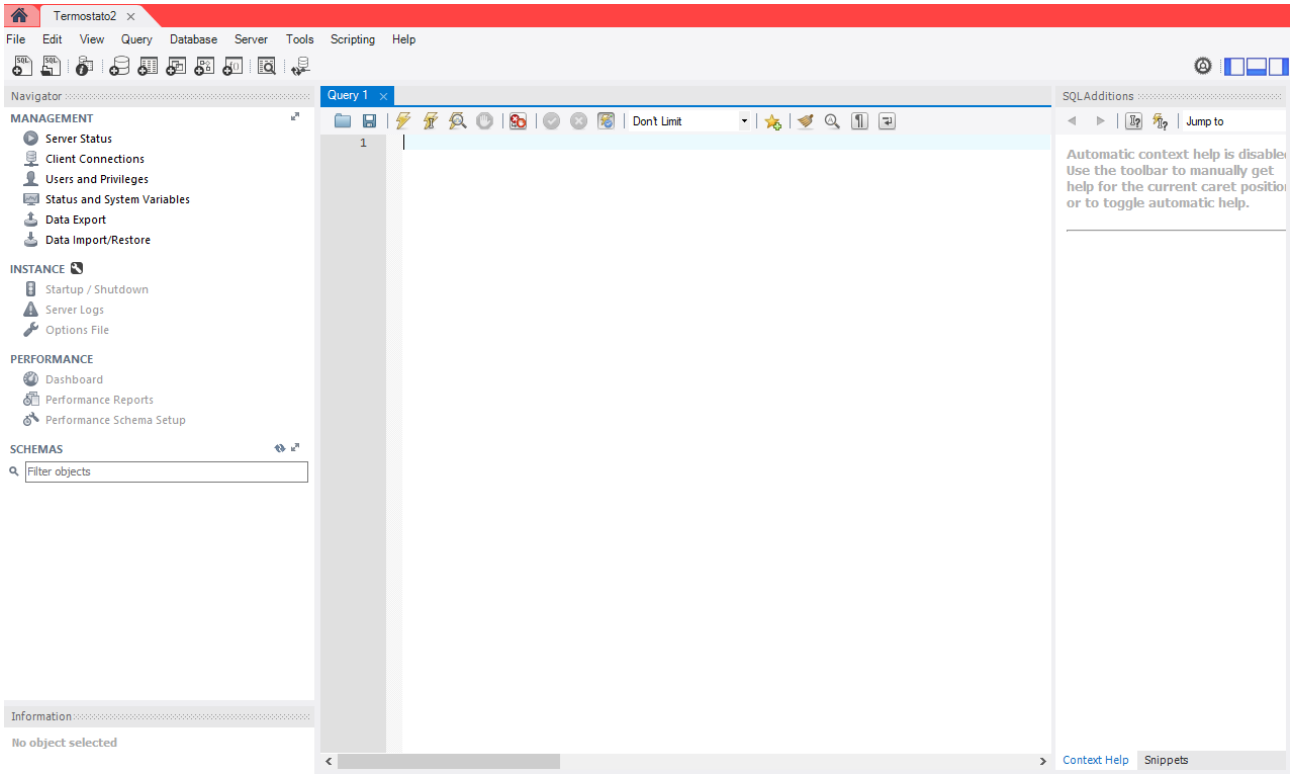


Ilustración 5: 4.3.2

- Ahora en la pestaña de **Query 1** vamos a teclear todo el código **SQL** que es necesario para crear la base de datos, las tablas y los campos necesarios; la definición de las líneas **SQL** las desarrollaremos por rectángulos de colores. Ilustración 6: 4.3.2
  - Rectángulo negro: Con **CREATE DATABASE** creamos la base de datos **Temperaturas**. Con **USE** básicamente le decimos al motor **mysql** que vamos a trabajar con la base de datos **Temperaturas**. La clave **Inno** es un mecanismo de almacenamiento y de integridad referencial aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil.
  - Rectángulo naranja: Con **CREATE TABLE** creamos la tabla **temperatura_invierno** que alojará las temperaturas que queremos en la estación de invierno (frío). Serán cuatro variables de tipo **int** que pueden ser **NULL**(nulas), menos la de i**dtemperatura_invierno** porque es la **PRIMARY KEY** (clave principal) y auto incremental. La clave **Inno** es un mecanismo de almacenamiento y de integridad referencial aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil.
  - Rectángulo rojo: Con **CREATE TABLE** creamos la tabla **temperatura_verano** que alojará las temperaturas que queremos en la estación de verano (calor). Serán cuatro variables de tipo **int** que pueden ser **NULL** (nulas), menos la de i**dtemperatura_verano** porque es la **PRIMARY KEY** (clave principal) y auto incremental. La clave **Inno** es un mecanismo de

almacenamiento y de integridad referencial aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil.

- Rectángulo amarillo: Con **CREATE TABLE** creamos la tabla **maquinaEncendida** que guardará el estado de la máquina que vamos a conectar a este termostato. Tendremos dos estados: encendida(1) y apagada(0) que se guardará en el campo **maquinaEncendidacol** que será de tipo **tinyint** y podrá ser **NULL** (nula), aquí también tenemos una clave **PRIMARY KEY** (clave principal) **idmaquina_Encendida** que no podrá ser **NULL**. La clave **Inno** es un mecanismo de almacenamiento y de integridad referencial, aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil.

- Rectángulo verde: Con **CREATE TABLE** creamos la tabla **maquina_manual** que guardará el estado de la máquina si está en modo manual. Tendremos dos estados: encendida(1) y apagada(0) que se guardará en el campo **maquina_manualcol** que será de tipo **tinyint** y podrá ser **NULL** (nula), aquí también tenemos una clave **PRIMARY KEY** (clave principal) **idmaquina_manual** que no podrá ser **NULL**. La clave **Inno** es un mecanismo de almacenamiento y de integridad referencial, aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil.

- Rectángulo lila: Con **CREATE TABLE** creamos la tabla **estación** que guardará la estación del año y tendremos una **PRIMARY KEY** que será **idestacion** que se autoincrementará y no podrá ser **NULL** y el campo **estación** de tipo **tinyint** guardará la estación del año. La clave **Inno** es un mecanismo de almacenamiento y de integridad referencial, aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil.

- Rectángulo marrón: Con **CREATE TABLE** creamos la tabla **tempHogar** que guardará las temperaturas de la vivienda para mantener un seguimiento de las temperaturas, en esta tabla tenemos tres campos: **tempInterior,** para las temperatura del interior de la vivienda; **tempExterior,** para las temperaturas del exterior de la vivienda, ambos campos son del tipo **float** y pueden ser **NULL;** y el campo **fechaCaptura**, que indica la fecha de la toma de las temperaturas, que es del tipo **TIMESTAM** (formato fecha) y no podrá ser **NULL** (con **CURRENT_TIMESTAMP** la fecha se guardará automáticamente al agregar las temperaturas con la fecha del sistema (servidor)). La clave **Inno** es un mecanismo de almacenamiento y de integridad referencial, aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil.

- Rectángulo verde oscuro: Con **CREATE TABLE** creamos la tabla **TemperaturaPecera** que guardará las temperaturas del acuario para mantener un seguimiento de las mismas, en esta tabla tenemos cinco campos: **temperaturaPecera** que registra las temperatura del agua del acuario, **temperaturaSalidaAgua** que registra las temperatura de la salida

del agua del enfriador-calefactor (ambos son del tipo **float** y pueden ser **NULL)**, también tenemos **creadaFecha**, que es el campo de la fecha en la que se han tomado las temperaturas y es del tipo **TIMESTAM** (formato fecha) que no podrá ser **NULL** y con **CURRENT_TIMESTAMP** la fecha se guardará automáticamente al agregar las temperaturas con la fecha del sistema (servidor). Por último, contamos con dos campos: **estacion** y **idTemperaturaPecera** que no se usan por ahora pero en próximas implementaciones serán útiles. La clave **Inno** es un mecanismo de almacenamiento y de integridad referencial, aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil.

- Rectángulo verde claro: Con **CREATE TABLE** creamos la tabla **tabla_temperatura** que guardará el estado de la máquina, si está apagada o encendida. Tendremos una **PRIMARY KEY** que será **idtabla_temperatura** que se autoincrementará y no podrá ser **NULL** y el campo **aver** de tipo **tinyint** guardará **1** (encendida) o **0** (apagada). La clave **Inno** es un mecanismo de almacenamiento y de integridad referencial, aunque en esta base de datos no lo vamos a utilizar, pero en un futuro puede ser útil. Ilustración 6b: 4.2.3

Ilustración 6: 4.3.2

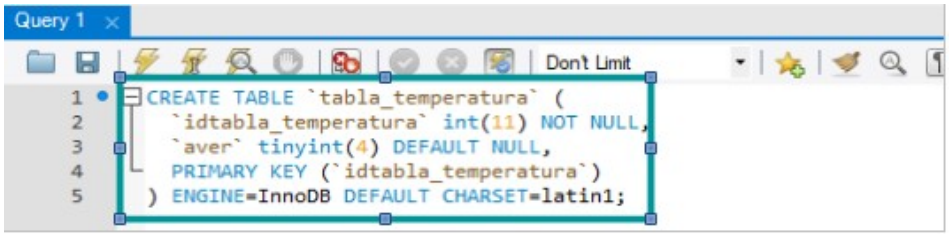


Ilustración 6b: 4.2.3

- Cuadrado azul: Pulsamos el botón del relámpago y ejecutamos la sentencia **SQL** y todo quedará como en la Ilustración 7: 4.3.2 .

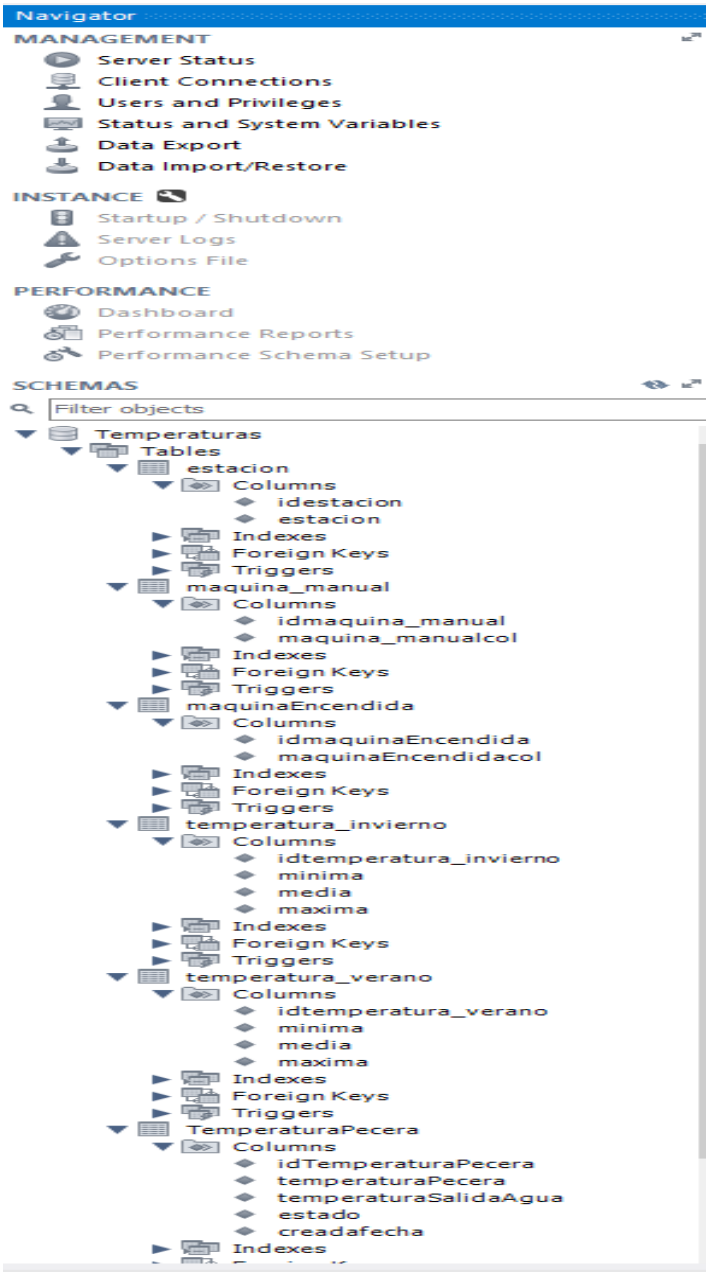


Ilustración 7: 4.3.2

- En este paso vamos a instalar el servidor web **Apache**, este servidor solo vamos a utilizarlo como servidor **web- php**.
  - Vamos a instalar los paquetes: **apache2** que es el servidor web, **php5** que es servicio **php** y las librerías y módulos de **apache** y **php5**.

```
pi@Termostato ~ $ sudo apt-get install apache2 php5 libapache2-mod-php5
```

  - Ahora solo tendremos que entrar en la ruta de directorios que se muestran en la siguiente imagen y aquí podremos alojar la web que tendrá los datos de nuestro proyecto.

```
pi@Termostato ~ $ cd /var/www/
pi@Termostato /var/www $
```

- Nos queda la instalación de **Rpi.GPIO**, que son las **librerías** para poder acceder a este periférico de la raspberry pi:
  - Entramos en **terminal** de **linux,** después tecleamos **sudo python** para entrar en la consola de **python**, importamos la librería con **import Rpi.GPIO** y después, para asegurarnos, comprobamos la versión con **Rpi.GPIO.VERSION**.

```
pi@Termostato ~ $ sudo python
Python 2.7.3 (default, Jun 22 2016, 03:14:32)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO
>>> RPi.GPIO.VERSION
'0.6.2'
>>>
```

  - Si nos muestra algún error no la tendremos instalada, tenemos que instalarla:
    - Descargamos el archivo de la librería de su dirección de descarga.

```
pi@Termostato ~ $ wget https://pypi.python.org/packages/RPi.GPIO-0.6.3.tar.gz
```

    - Descomprimimos el archivo con tar.

```
pi@Termostato ~ $ tar zxf RPi.GPIO-0.6.3.tar.gz
```

    - Entramos en el directorio descomprimido.

```
pi@Termostato ~ $ cd RPi.GPIO-0.6.3/
pi@Termostato ~/RPi.GPIO-0.6.3 $
```

    - Finalmente instalamos la librería.

```
pi@Termostato ~/RPi.GPIO-0.6.3 $ sudo python setup.py install
```

4.4-. Metodología climatización hogar (Raspberry Pi)

En este apartado vamos a implementar la parte de la programación e instalación de periféricos en nuestro servidor en la **raspberry pi**.

Empezaremos con una breve descripción de este pequeño computador, cuyos componente están en una placa (CPU, GPU, Ethernet, sonido, etc.). Diseñado y mantenido en Reino Unido por una fundación llamada **Raspberry Pi** (Ilustración 4: 4.4), su principal fin es la de utilizarla en la enseñanza.

El lenguaje de programación principal que se utiliza en las **raspberry** es **python** aunque también se pueden utilizar otros lenguajes de programación.

El sistema operativo principal de estos computadores es **GNU/Linux,** una versión adaptada de **Debian** que se llama **Raspbian** y como es típico de Linux se puede instalar cualquier versión o distro adaptada a la arquitectura **ARM.** También en la actualidad podemos instalar **windows 10** con la herramienta de **Microsoft Windows IOT** pero solo para las últimas versiones (modelo 3) de estos pequeños computadores.

Características principales:

- SoC Broadcom BCM2836
- CPU: ARM11 ARMv7 ARM Cortex-A7 4 núcleos @ 900 MHz
- GPU: Broadcom VideoCore IV 250 MHz. OpenGL ES 2.0
- RAM: 1 GB LPDDR2 SDRAM 450 MHz
- USB 2.0: 4
- Salidas de video: HDMI 1.4 @ 1920x1200 píxeles
- Almacenamiento: microSD
- Ethernet: 10/100 Mbps
- Tamaño: 85,60x56,5 mm
- Peso: 45 g
- Consumo: 5v, 900mA

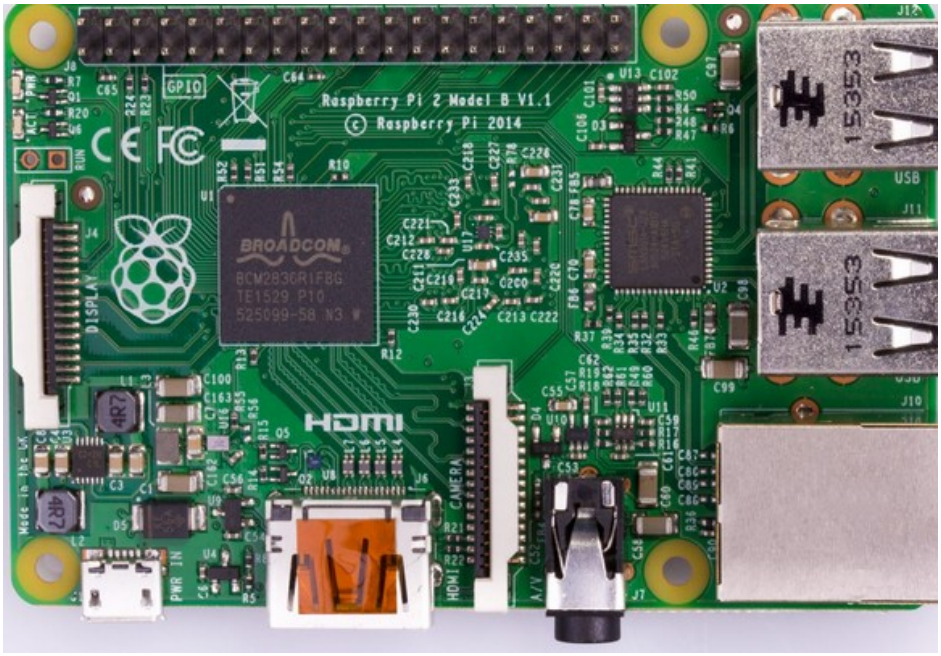


Ilustración 4: 4.4

Todo el algoritmo del **script** está explicado en esta sección:

- Toda las instrucciones para controlar las temperatura está dentro de un **while True** con un **time.sleep(180)** (una pausa de 3 minutos para que se ejecuten todas las funciones). Ilustración 1: 4.4

```
while True:
    time.sleep(180)
```

Ilustración 1: 4.4

- El script se ejecuta al iniciarse el sistema operativo con el **demonio cron**[7] de **GNU/Linux** que se configura con el comando **crontab -e**. Ilustración 2: 4.4

```
pi@Termostato ~ $ crontab -e
```

Ilustración 2: 4.4

- Debemos agregar la ruta absoluta del ejecutable en este caso '***/usr*/bin/python'** y la ruta absoluta de dónde está el **script** que se debe ejecutar, en este caso '**/home/pi/temperatura.py'** y cuándo se debe ejecutar con '**@reboot',** es decir, en cada reinicio**. Ilustración 3: 4.4**
-

7 **cron** es un administrador regular de procesos en segundo plano (demonio) que ejecuta procesos a intervalos regulares (cada minuto, día, semana o mes).

```
@reboot    /usr/bin/python /home/pi/temperatura.py
```

Ilustración 3: 4.4

4.4.1-. Instalación de periféricos.

En la **Raspberry pi** tenemos una herramienta muy potente, los pines **GPIO** que son entradas y salidas de uso general, podemos verlos en la parte superior de la Ilustración 4: 4.4, son 40 pines macho y como podemos ver en la Ilustración 1: 4.4.1, nosotros solo vamos a utilizar algunos pines. Vamos a enumerar los más utilizados:

Tipos de pines:

PWM (modulación de duración de impulsos)
Software PWM disponible en todos los pines
Hardware PWM disponible en GPIO12, GPIO13, GPIO18, GPIO19

SPI
SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18);
CE1(GPIO17); CE2 (GPIO16)

I2C
Datos: (GPIO2); Reloj (GPIO3)
Datos EEPROM: (GPIO0); Reloj EEPROM (GPIO1)

Serial
TX (GPIO14); RX (GPIO15)

Raspberry Pi2 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Ilustración 1: 4.4.1

4.4.1.1-. Instalación de sensores de temperatura.

Vamos a instalar los mismos sensores de temperatura que en el acuario, explicado en la sección 4.2.1-. Sensor de temperatura DS18B20.

4.4.1.1.1-. Esquema de montaje.

En esta sección vamos a mostrar el esquema de montaje y vamos a necesitar los siguientes componentes:

- Raspberry Pi Model 2.
- Cables.
- 2 Sensores DS18B20.
- Resistencia 4.7K.
- ProtoBoard.

Como vemos en la imagen vamos a conectar el pin 1 de 3.3V a la resistencia en **pull-up**, la misma resistencia la vamos a conectar a los sensores de temperatura por medio de su pin de señal, y su pin de señal lo conectaremos al pin 7 de la **raspberry pi** que recogerá las mediciones de los sensores; por último, la masa de la **raspberry pi** la conectaremos a la masa del sensor. Ilustración 1: 4.4.1.1.1

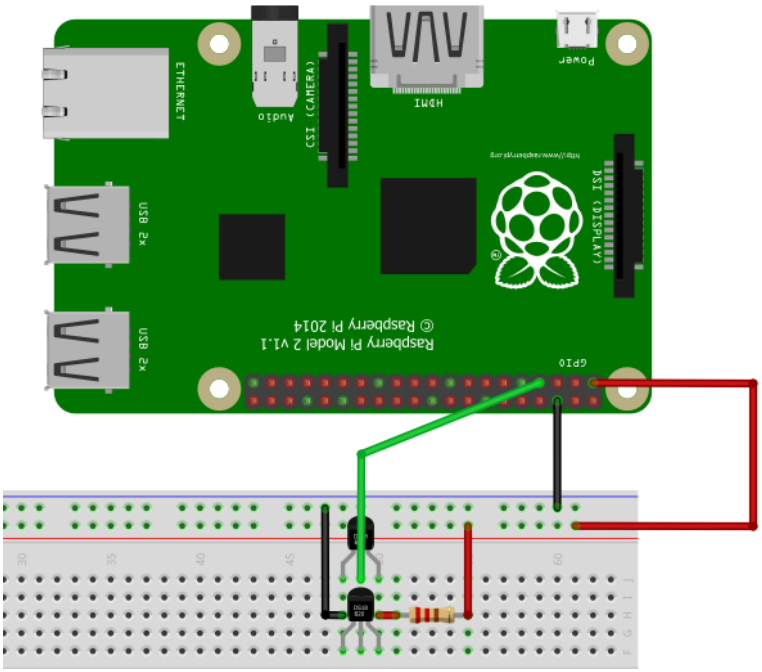


Ilustración 1: 4.4.1.1.1

4.4.1.1.2-. Script de programación de sensores de temperatura.

Ahora vamos a desarrollar la parte del **script** que se hace cargo de la parte de lectura de datos de sensores de temperatura y la parte de inserción, actualización y consulta en la base de datos.

En esta parte programaremos en **Python 2.8** que es, por defecto, el lenguaje de programación de **raspberry pi** aunque como ya hemos dicho antes podremos elegir otros lenguajes.

- Empezaremos con la definición de librerías:Ilustración 1: 4.2.1.2
  - **Request**: Librería que nos ayuda a comunicarnos con la web.
  - **Hashlib**: Librería de seguridad en las comunicaciones.
  - **Time**: Librería que nos da acceso a las funciones de tiempo: fecha, hora, etc.
  - **MySQLdb**: Librería que nos facilita la comunicación con la **BBDD** de **mysql**.
  - **RPI.GPIO**: Librería de acceso a **GPIO**.
  - **Commands**: Librería de acceso a terminal desde **script python**.
  - **Os**: Librería que nos permite acceder a funcionalidades dependientes del sistema operativo.

```
import requests
import hashlib
import time
import MySQLdb
import RPi.GPIO as GPIO
import commands
import os
```

Ilustración 1: 4.4.1.1.2

- Lo primero es guardar los **ids** de los sensores en un **array**, los **ids** de los sensores los obtenemos cuando se conectan los sensores a la **raspberry** y con la terminal de **linux** nos movemos hasta el directorio que indicamos en la Ilustración 2: 4.4.1.1.2, a continuación, definimos un **array** de **avgtemperatures** para realizar los cálculos y transformar los datos obtenidos en el sensor a un número que podamos entender (un número decimal) . Después con el primer **for**, vamos a transformar el **array sensoids** en un lista con la función **len()** para poder leer el dato uno por uno y también iniciamos la variable **temperatures[]**. En el segundo **for** vamos a coger solo los 4 primeros datos del sensor ya que el sensor nos dará muchos más y solo nos interesan los 4 primeros y también iniciamos la variable **text**. En el **while** vamos a leer los datos del sensor y los vamos a guardar en la variable **text.** Después vamos a trabajar con la variable **text** para eliminarle los datos que nos hacen falta con **text.split('t=')** le quitamos la **t=** que no nos hace falta y la guardamos en la variable **secondline.** Seguidamente, le quitamos los espacios a **secondline** con otro **split** y lo guardamos en **temperaturedata,** ahora esta variable la transformamos en un **float** y la guardamos en **temperatures** dividiendo por 1000. Después tenemos que hacer la media de todas la temperaturas con un cálculo que nos indica en el **datashet** del sensor. Guardaremos las temperaturas en las variables en **temperatura_interior** y **temperatura_exterior**.

  Este algoritmo será realizado las veces que sea necesario según el número de sensores que tengamos en la placa. Ilustración 3: 4.4.1.1.2

```
pi@Termostato /sys/bus/w1/devices $ cd /sys/bus/w1/devices/
pi@Termostato /sys/bus/w1/devices $
```

Ilustración 2: 4.4.1.1.2

```
sensorids = ["28-0000067ea33c","28-0000062af3eb"]
avgtemperatures = []

for sensor in range(len(sensorids)):
    temperatures = []
    for polltime in range(0,3):
        text = '';
        while text.split("n")[0].find("YES") == -1:
            tfile = open("/sys/bus/wl/devices/"+ sensorids[sensor] +"/wl_slave")
            text = tfile.read()
            tfile.close()
            time.sleep(1)

    secondline = text.split("t=")[1]
    temperaturedata = secondline.split("\n")
    temperature = float(temperaturedata[0])
    temperatures.append(temperature / 1000)

    avgtemperatures.append(sum(temperatures) / float(len(temperatures)))


temperatura_interior = avgtemperatures[0] -1
temperatura_exterior = avgtemperatures[1]
```

Ilustración 3: 4.4.1.1.2

- En este paso vamos a subir las temperaturas a la base de datos:
  - Creamos la conexión con **MySQLdb** a la base de datos, en **localhost** (nuestro servidor) insertamos usuario y contraseña y le decimos a la base de datos que deseamos insertar información que en nuestro caso en **Temperaturas**.
  - Creamos un **cursor** con **db.cursor()** y lo guardamos en la variable cursor.
  - Ejecutamos la inserción de las temperaturas con **cursor.execute()**.
  - Con **db.commit()** hacemos efectiva la escritura en la base de datos.
  - Con **db.close()** cerramos la conexión.

  Con estos código ya tenemos guardadas las temperaturas en nuestra base de datos. Ilustración 4: 4.4.1.1.2

```
db = MySQLdb.connect("localhost","root","contrasenya","Temperaturas")
cursor = db.cursor()
cursor.execute("""INSERT INTO temps (temp1,temp2) VALUES (%s,%s) """, (temperatura_interior, temperatura_exterior))
db.commit()
db.close()
```

Ilustración 4: 4.4.1.1.2

- Vamos a conectar con la base de datos y crearemos un **cursor** nuevo para realizar una serie de consultas para rellenar las variables que nos hacen falta para nuestro seguimiento de temperaturas; lo llamaremos **cur,** con este cursor vamos a realizar todas las consultas necesarias y extraer los datos necesarios de la base de datos. Ilustración 5: 4.4.1.1.2

```
db = MySQLdb.connect("localhost","root","contrasenya","Temperaturas")
cur = db.cursor()
```

Ilustración 5: 4.4.1.1.2

- En la primera consulta que ejecutamos, leemos del campo **estación** incluido en la tabla **estación** que, a su vez, pertenece a la base de datos **Temperaturas,** si es **1** (frío-invierno) o si es **0** (verano-calor) y lo guardamos en la variable **estación,** esta variable nos informa de si tenemos la máquina en modo frío o en modo calor; según sea el modo escogido el aparato trabajará con temperaturas de frío o calor. En la segunda consulta hacemos lo mismo pero con la tabla **maquina_manual** y el campo **maquina_manualcol** si es **1** (encendido) o si es **0** (apagado) y lo guardamos en la variable **maquina_manualcol,** esta variable conectará la máquina en modo manual. Ilustración 6: 4.4.1.1.2

```
cur.execute("SELECT estacion FROM Temperaturas.estacion;")
for row in cur.fetchall():
    estacion = row[0]

print estacion

cur.execute("SELECT maquina_manualcol FROM Temperaturas.maquina_manual;")
for row in cur.fetchall():
    maquina_manualcol = row[0]

print maquina_manualcol
```

Ilustración 6: 4.4.1.1.2

- En las siguientes consultas estamos leyendo de la base de datos las temperaturas que vamos a asignar en la época de calor, las cuales serán tres: **minimaVerano, mediaVerano** y **maximaVerano**; esta consulta es igual que las anteriores pero cambiando su tabla y su campo, y almacenándolas en distintas variables. Ilustración 7: 4.4.1.1.2

```
#Verano
cur.execute("SELECT minima FROM Temperaturas.temperatura_verano;");
for row in cur.fetchall():
    minimaVerano = row[0]

cur.execute("SELECT media FROM Temperaturas.temperatura_verano;");
for row in cur.fetchall():
    mediaVerano = row[0]

cur.execute("SELECT maxima FROM Temperaturas.temperatura_verano;");
for row in cur.fetchall():
    maximaVerano = row[0]
```

Ilustración 7: 4.4.1.1.2

- En las siguientes consultas estamos leyendo de la base de datos las temperaturas que vamos a asignar en la época de frío y que serán tres: **minimaInvierno, mediaInvierno** y **maximaInvierno.** La consulta es igual que las anteriores pero cambiando su tabla y su campo, y almacenándolas en distintas variables. Ilustración 8: 4.4.1.1.2

```
# Invierno

cur.execute("SELECT minima FROM Temperaturas.temperatura_invierno;");
for row in cur.fetchall():
    minimaInvierno = row[0]

cur.execute("SELECT media FROM Temperaturas.temperatura_invierno;");
for row in cur.fetchall():
    mediaInvierno = row[0]

cur.execute("SELECT maxima FROM Temperaturas.temperatura_invierno;");
for row in cur.fetchall():
    maximaInvierno = row[0]
```

Ilustración 8: 4.4.1.1.2

- Esta consulta a la base de datos es básicamente el apagado y encendido de la máquina y lo guardamos en la variable **verTermostatoEncendido**. Ilustración 10: 4.4.1.1.2

```
cur.execute("SELECT aver FROM tabla_temperatura WHERE idtabla_temperatura='1'");
for row in cur.fetchall():
    verTermostatoEncendido = row[0]
```

Ilustración 10: 4.4.1.1.2

- En esta parte del código vamos a definir variables que nos servirán para diseñar la casuística del algoritmo del control de la máquina: Ilustración 11: 4.4.1.1.2
  - Las tres primeras variables son básicamente para dar información para unas futuras implementaciones. Utilizaremos estas variables, por ejemplo, para mandar correos electrónicos a la hora que se encienda o apague la máquina.
  - A la variable **comandoEncendidoGeneral** le asignamos el '**gpio -g mode 18 out**' que indica al **GPIO18**(Pin 10) que debe cambiar al modo salida.
  - A la variable **comandoApagadoGeneral** le asignamos **'gpio -g mode 18 in'** que le indica al **GPIO18** (Pin 10) que se cambie en modo entrada.
  - A la variable **comandoEncendido** le asignamos **'gpio -g write 18 0'** que indica que el **GPIO18** (Pin 10) se cambie a modo **0** (**LOW**).
  - A la variable **comandoApagado** le asignamos **'gpio -g write 18 1'** que indica que el **GPIO18** (Pin 10) se cambie a modo **1** (**HIGH**).
  - Falta completar en este apartado que '**gpio -g mode 18 out**', por ejemplo, son literales que quiere decir que si tecleamos ese comando en terminal de **linux** funcionará.

```
encendido = 'Maquina Encendida'
apagado =  'Maquina Apagada'
hora = datetime.datetime.fromtimestamp(hora).strftime('%H:%M:%S')
comandoEncendidoGeneral='gpio -g mode 18 out'
comandoApagadoGeneral='gpio -g mode 18 in'
comandoEncendido='gpio -g write 18 0'
comandoApagado='gpio -g write 18 1'
```

Ilustración 11: 4.4.1.1.2

- En este paso vamos a crear la función para encender la máquina: Ilustración 12: 4.4.1.1.2

- Para poder acceder a la terminal de linux desde script de **python** utilizaremos **commands.getoutput(comandoEncendidoGeneral)** y activaremos el pin 18 en modo salida.
- Como en el paso anterior, activaremos la máquina y pondremos su pin 18 a 0 con el comando **commands.getoutput(comandoEncendido).**
- Con **os.popen(comandoEncendido)** abrimos una tubería hacia el **comandoEncendido**.
- Para actualizar la base de datos del estado de la máquina, debemos insertar esta actualización y lo hacemos con el **cursor** definido anteriormente **cur.execute(),** entonces cambiamos el campo **maquinaEncendidocol** a 1 diciendo que está encendida, que lo veremos en la web de control del termostato.
- Con **db.commit()** validamos la escritura en la base de datos y subimos la actualización.
- Por último, **print encendido, hora** es información para **debug().**

```
def ponerEncender():
    commands.getoutput(comandoEncendidoGeneral)
    commands.getoutput(comandoEncendido)
    os.popen(comandoEncendido)
    cur.execute("""UPDATE maquinaEncendida SET maquinaEncendidacol=%s WHERE idmaquinaEncendida=%s """, (1, 1))
    db.commit()
    print encendido, hora
```

Ilustración 12: 4.4.1.1.2

- En este paso vamos a crear la función para apagar la máquina: Ilustración 13: 4.4.1.1.2
  - Para poder acceder a la terminal de linux desde script de **python** utilizaremos **commands.getoutput(comandoApagadoGeneral)** y activaremos el pin 18 en modo entrada.
  - Como en el paso anterior, activaremos la máquina y pondremos su pin 18 a 1 con el comando **commands.getoutput(comandoApagado).**
  - Con **os.popen(comandoApagado)** abrimos una tubería hacia el **comandoApagado**.
  - Para actualizar la base de datos del estado de la máquina, debemos insertar esta actualización y lo hacemos con el **cursor** definido anteriormente **cur.execute(),** entonces cambiamos el campo **maquinaEncendidocol** a 0

diciendo que está apagada, como podemos comprobar en la web de control del termostato.

- Con **db.commit()** validamos la escritura en la base de datos y subimos la actualización.
- Por último, **print encendido, hora** es información para **debug().**

```
def ponerApagar():
    commands.getoutput(comandoApagado)
    commands.getoutput(comandoApagadoGeneral)
    print apagado, hora
    cur.execute("""UPDATE maquinaEncendida SET maquinaEncendidacol=%s WHERE idmaquinaEncendida=%s """, (0, 1))
    db.commit()
```

Ilustración 13: 4.4.1.1.2

- Aquí, en este algoritmo vamos a utilizar todas la variables mencionadas en este apartado pero en lo respectivo a las variables de verano. Ilustración 14: 4.4.1.1.2
  - Con un condicional **if** vamos a comprobar que todas la variables estén en un determinado modo para encender la máquina y enfriar el ambiente, si la **estacion** está a **1** (modo calor) y la **maquina_manualcol** está a 0 (el modo manual está desconectado) y **verTermostatoEncendido** es igual a 1 (que el termostato está encendido); entonces, con otro condicional vamos a comprobar las temperaturas: si **temperatura_interior** es mayor que **minimaVerano,** encendemos la máquina con la función ponerEncender(); si no, comprobamos el otro rango de temperaturas que queremos y si la **temperatura_interior** es menor que la **maxima_verano** y **temperatura_interior** es mayor que **mediaVerano,** entonces, encendemos la máquina con la función **ponerEncender(),** y si no se da ninguna de las anteriores, apagamos la máquina con la función **panerApagar()**. Por último, si el primer **if** no se cumple mandamos un mensaje para informar que está apagado, lo que nos será útil para hacer **debug**.

```
#verano
if (estacion == 1 and maquina_manualcol == 0 and verTermostatoEncendido == 1):
    if (temperatura_interior > minimaVerano):
        ponerEncender()
    elif ((temperatura_interior < maximaVerano and temperatura_interior > mediaVerano):
        ponerEncender()
    else:
        ponerApagar()
else:
    print apagado, hora
```

Ilustración 14: 4.4.1.1.2

- Aquí, en este algoritmo vamos a utilizar todas la variables mencionadas en este apartado pero en lo respectivo a las variables de invierno. Ilustración 15: 4.4.1.1.2
    - Con un condicional **if** vamos a comprobar que todas la variables estén en un determinado modo para encender la máquina y calentar el ambiente, si la **estacion** está a **0** (modo frío), la **maquina_manualcol** está a 0 (el modo manual está desconectado), **verTermostatoEncendido** es igual a 1 (el termostato está encendido) y el **sensor de luz** (**timeLuz(10)** que explicaremos en otra sección 4.4.1.2.2-. Script de programación de sensor de luz.) es mayor de 500; entonces, con otro condicional, vamos a comprobar las temperaturas. Si **temperatura_interior** es menor que **minimaInvierno,** encendemos la máquina con la función **ponerEncender()**, si no comprobamos el otro rango de temperaturas que queremos y entonces si la **temperatura_interior** es menor que **mediaInvierno,** encendemos la máquina con la función **ponerEncender(),** de lo contrario, apagamos la máquina con la función **ponerApagar()**. Por último, si el primer **if** no se cumple mandamos un mensaje para informar que está apagado, lo que nos será útil para hacer **debug**.

```
#Invierno
if (estacion == 0 and maquina_manualcol == 0 and verTermostatoEncendido == 1 and timerLuz(10) > 500):
    if (temperatura_interior < minimaInvierno):
        ponerEncender()
    elif (temperatura_interior < mediaInvierno):
        ponerEncender()
    else:
        ponerApagar()
else:
    print apagado, hora
```

Ilustración 15: 4.4.1.1.2

- Por último, desconectamos la base de datos. Ilustración 16: 4.4.1.1.2

```
db.close()
```

Ilustración 16: 4.4.1.1.2

4.4.1.2-. Instalación del sensor de luz.

Es un sensor diminuto con dos pines y muy sencillo; es una fotorresistencia **LDR** que es sensible a la luz que recibe y ofrece una resistencia mayor o menor en función de la cantidad de luz que recibe.

Características: Ilustración 1: 4.4.1.2

- Resistencia (con luz) : ~1k Ohm
- Resistencia (oscuridad): ~10k Ohm
- Voltaje máximo : 150V
- Disipación: 100mW máxima

Ilustración 1: 4.4.1.2

4.4.1.2.1-. Esquema de montaje.

En esta sección vamos a mostrar el esquema de montaje y vamos a necesitar los siguientes componentes: Ilustración 1: 4.4.1.2.1

- Raspberry Pi Model 2.
- Cables.
- ProtoBoard
- Sensor LDR.
- Condensador 22nF.

Como podemos ver en Ilustración 1: 4.4.1.2.1, tenemos la **masa (GND)** de la placa conectada al condensador por su polo negativo, el polo positivo del condensador conectado a una parte del **sensor LRD,** y la otra pata del sensor está conectado al pin de 3,3V de la **raspberry pi,** además, entre el condensador y el sensor tenemos la señal al **pin 10** de la placa.

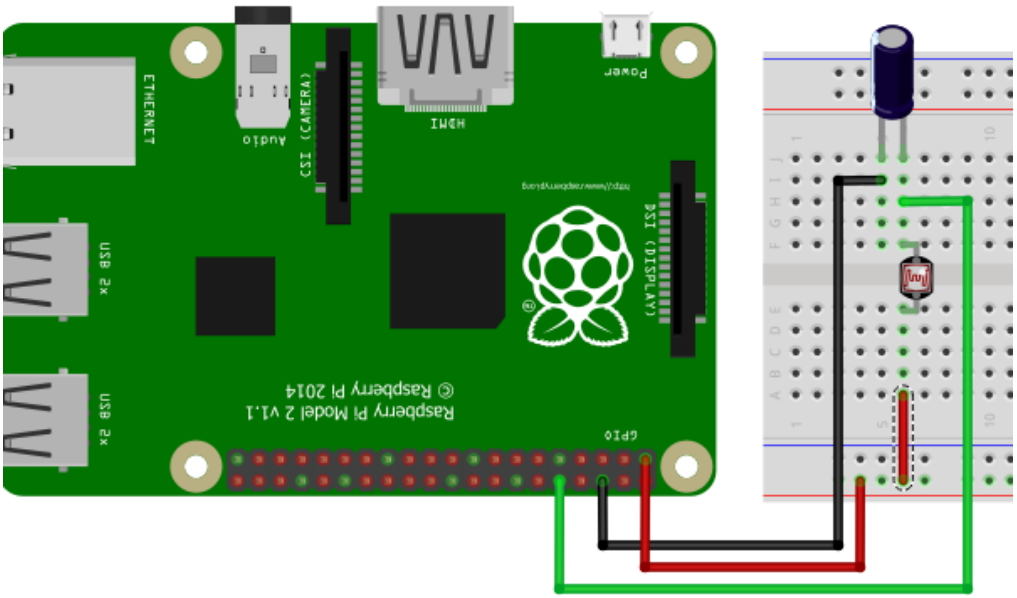


Ilustración 1: 4.4.1.2.1

4.4.1.2.2-. Script de programación de sensor de luz.

En esta parte del proyecto vamos a implementar la parte de programación del sensor de luz (**LDR**), que controlará el apagado de la máquina en invierno cuando la luz del sol caliente las placas termo solares y no haga falta activar la máquina y, de este modo, tener un gasto energético menor.

- Declaramos las librerías que vamos a necesitar para implementar el algoritmo, en este caso solo vamos a utilizar: Ilustración 1: 4.4.1.2.2
  - **Rpi.GPIO:** Librería que da acceso a **GPIO** y le vamos a dar un alias o etiqueta que será **GPIO**.
  - **Time**: Librería que da acceso al tiempo.
  - **Sys**: Librería que proporciona acceso a variables y funciones con el intérprete.

```
import RPi.GPIO as GPIO
import time
import sys
```

Ilustración 1: 4.4.1.2.2

- En esta parte del algoritmo vamos a definir la función que lee los datos del sensor **LDR**: Ilustración 2: 4.4.1.2.2
  - Definimos la variable **reading** y la iniciamos a **0**.
  - Con **GPIO.setup(pin, GPIO.OUT)** configuramos el pin elegido en modo salida.
  - Con **GPIO.output(pin, GPIO.LOW)** configura la salida del pin elegido en modo bajo(0).
  - Con **time.sleep(0.1)** le damos un tiempo para que se inicie.
  - Con **GPIO.setup(pin, GPIO.IN)** configuramos el pin elegido en modo entrada.
  - Con **while (GPIO.input(pin) == GPIO.LOW):** sumamos 1 a la variable **reading** mientras la entrada del pin seleccionado sea igual al modo **LOW**(0).
  - Cuando termine el **while,** retornamos la variable **reading** que nos dará un rango entre 1 (menos luz) y 1000 (más luz).

```
def timerLuz (pin):
    reading = 0
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)
    time.sleep(0.1)
    GPIO.setup(pin, GPIO.IN)
    while (GPIO.input(pin) == GPIO.LOW):
            reading += 1
    return reading
```

Ilustración 2: 4.4.1.2.2

- Ahora solo nos queda recordar dónde utilizamos esta función (rectángulo azul) ya que se ha explicado en la página 73. Ilustración 3: 4.4.1.2.2

```
#Invierno
if (estacion == 0 and maquina_manualcol == 0 and verTermostatoEncendido == 1 and timerLuz(10) > 500):
    if (temperatura_interior < minimaInvierno):
        ponerEncender()
    elif (temperatura_interior < mediaInvierno):
        ponerEncender()
    else:
        ponerApagar()
else:
    print apagado, hora
```

Ilustración 3: 4.4.1.2.2

4.4.1.3-. Instalación relé encendido/apagado del enfriador-calefactor.

En esta sección vamos a mostrar cómo conectamos el relé para activar la máquina enfriador-calefactor que muy parecido a la sección de 4.2.5-. Relés, pero el pin de señal es de otra placa aunque funciona igual, así que no nos vamos a extender. Ilustración 1: 4.4.1.3

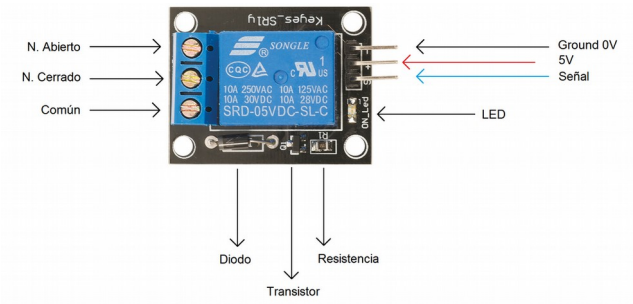


Ilustración 1: 4.4.1.3

4.4.1.3.1-. Esquema de montaje.

Este esquema de montaje es igual que el de la sección de 4.2.5-. Relés pero sobre una raspberry pi. Ilustración 1: 4.4.1.3.1

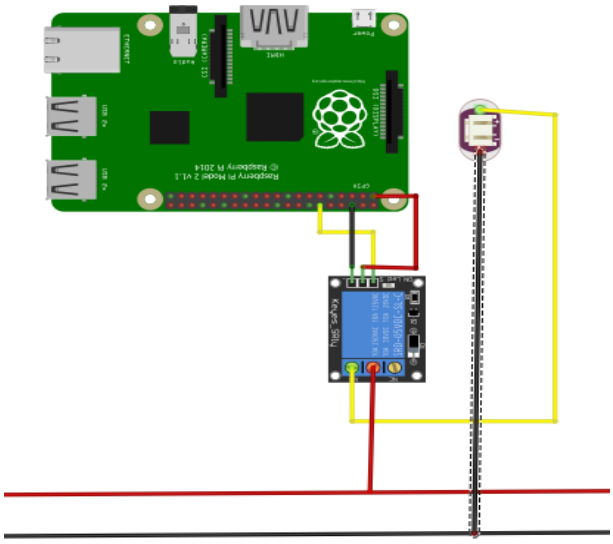


Ilustración 1: 4.4.1.3.1

4.4.1.3.2-. Script de programación de relé.

- El relé dejará pasar tensión o no según el pin GPIO18 (Pin 10) esté a 1 o a 0 con los siguientes comandos que, si recordamos, están explicados en la Ilustración 11: 4.4.1.1.2.
  - **'gpio -g write 18 0'** que indica que el **GPIO18** (Pin 10) se cambie a modo **0** (**LOW**).
  - **'gpio -g write 18 1'** que indica que el **GPIO18** (Pin 10) se cambie a modo **1** (**HIGH**).

4.4.1.4-. Instalación bluetooth.

La instalación del periférico de **Bluetooth** es bastante sencilla: es conectar por **usb** el dispositivo, entonces **linux** lo detectará en el momento y solo tendremos que configurarlo por terminal.

- En primer lugar, instalaremos los paquetes necesarios.

```
pi@Termostato ~ $ sudo apt-get install bluetooth bluez-utils blueman
```

- Con **hciconfig,** averiguamos la dirección del dispositivo bluetooth, en este caso es **hci0**.

```
pi@Termostato ~ $ hciconfig
hci0:   Type: BR/EDR  Bus: USB
        BD Address: [illegible]  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING PSCAN
        RX bytes:6802438 acl:163488 sco:0 events:247178 errors:0
        TX bytes:2590394 acl:127131 sco:0 commands:46080 errors:0
```

- Con **hcitool scan,** buscamos nuestro **bluetooth** que nos transmite por serial las temperaturas del acuario, en nuestro caso es **HC-06.**

```
pi@Termostato ~ $ hcitool scan
Scanning ...
        98:D3:32:10:EF:D4       HC-06
```

- Ahora tenemos que emparejar el **HC-06** al **usb** de nuestra placa, nos pedirá un pin predeterminado que es **1234** (esta contraseña se puede modificar en modo **AT)** y ya estará emparejado.

```
pi@Termostato ~ $ sudo bluez-simple-agent hci 98:D3:32:10:EF:D4
```

- Ahora tenemos dos dispositivos que se pueden comunicar entre sí a través del protocolo **bluetooth**. Sin embargo, todavía tenemos que configurar otro protocolo a través de **bluetooth**, que se denomina **RFCOMM** para emular la conexión en serie entre estos dos dispositivos. La configuración del protocolo se puede hacer modificando el archivo **"rfcomm.conf"**.
  - Hemos elegido como editor de texto el nano de GNU/Linux para poder añadir la emulación RFCOMM y poder comunicarnos.

```
pi@Termostato ~ $ sudo nano /etc/bluetooth/rfcomm.conf
```

  - Añadimos estas líneas con el protocolo de emulación (rfcomm1) que tienen configuración de la dirección **MAC** del dispositivo **HC-06** del acuario, el canal 1 y un comentario para saber identificar en qué momento se inicia el servicio en los **logs** del sistema. Finalmente guardamos el archivo.

```
rfcomm1 {
    bind yes;
    device 98:D3:32:10:EF:D4;
    channel 1;
    comment "Connection to Bluetooth serial module";
}
```

- Iniciamos el **rfcomm** y emparejamos con **bind all**. Con estos pasos ya tenemos conexión **serial** entre **arduino uno** del acuario y el servidor **raspberry pi.**

```
pi@Termostato ~ $ sudo rfcomm bind all
```

4.4.1.4.1-. Script de recibir datos mediante bluetooth

En este paso vamos a recibir las temperaturas del acuario y las vamos a guardar en la base de datos de nuestra raspberry pi. Ilustración 1: 4.4.1.4.1

- Lo primero, como siempre, es utilizar las librerías adecuadas:
  - **Time**: Librería que nos da acceso a las funciones tiempo: fecha, hora, etc.
  - **MySQLdb**: Librería que nos facilita la comunicación con la **BBDD** de **mysql**.
  - **Serial**: Librería que encapsula el acceso al puerto serie.
  - **Sys**: Librería que proporciona el acceso a variables y funciones con el intérprete.

```
import MySQLdb
import serial
import time
import sys
```

Ilustración 1: 4.4.1.4.1

- Ahora vamos a desarrollar el algoritmo de recogida de datos por **bluetooth** del acuario y vamos a guardarlo en la base de datos para después mostrarlo por web:
  - Lo vamos a integrar todo en una función llamada **blueConexion()**.
  - Vamos a englobarlo en una sentencia **try** para controlar los errores y no se detenga el script.
  - Vamos a guardar en la variable **ser** lo que recibamos por el dispositivo **bluetooth** a **9600 baudios** (velocidad de transmisión) con **serial.Serial("/dev/rfcomm1,9600")**.
  - Vamos a guardar en **read** el serial recogido en bruto, los trece primeros caracteres con **ser.read(13).**

- Vamos a dividir con **read.split(“ ”)** que es una función que divide una secuencia de caracteres, por el carácter que recoge la función (“ ” = espacio) y lo guardaremos en tres variables diferentes; **pecera**, **salidaAgua** y **estadook**. Aquí debemos recordar como se envían los datos del dispositivo **bluetooth** del acuario que está desarrollado en la siguiente sección 4.2.8.2-. Sketch arduino del **bluetooth** de **arduino uno**.
- Con las siguientes líneas de código vamos a subir nuestras variables, ya con la información guardada en ellas a la base de datos. Ilustración 2: 4.4.1.4.1
  - Con **db** guardamos la conexión a la base de datos que ya hemos utilizado en las otras secciones.
  - Creamos un nuevo cursor para ejecutar la inserción en la base de datos.
  - Con **cursor.execute()** insertamos los datos en la base de datos en sus respectivas tablas y campos.
  - Con **db.commit()** escribimos la actualización en la base de datos.
  - Con **db.close()** cerramos la conexión. Con estos pasos ya tenemos la información en la base de datos guardada para poder mostrarla en web.
  - Las líneas de código que se inician con **except** son para controlar un posible error dentro de **try** y que no se detenga el **script** por el error.
  - La línea de **finally** cierra todas las conexiones: las de la base de datos **db.close()** y la del serial **ser.close()**, en caso de error.

```
def blueConexion():
    try:
        ser = serial.Serial("/dev/rfcomm1", 9600)
            read = ser.read(13)
        pecera,salidaAgua,estadook = read.split(" ")
        db = MySQLdb.connect("localhost","root","contraseya","Temperaturas")
        cursor = db.cursor()
        cursor.execute("""INSERT INTO TemperaturaPecera (temperaturaPecera,
        temperaturaSalidaAgua,estado) VALUES (%s,%s,%s) """,
                       (pecera, salidaAgua, estadook))
        db.commit()
        db.close()
        ser.close()
        time.sleep(1)

    except serial.serialutil.SerialException as error2:
        print error2

    except Exception as error:
        print error

    finally:
        db.close()
        ser.close()
```

Ilustración 2: 4.4.1.4.1

- Ahora debemos iniciar la función **blueConexion()** en el **while** principal para que se ejecute cada x tiempo según necesitemos, en este caso cada 3 minutos. Ilustración 3: 4.4.1.4.1

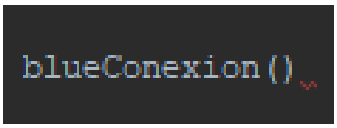


Ilustración 3: 4.4.1.4.1

4.4.3-. Programación del control climatización web y php.

Ahora vamos a desarrollar la página web con **html** y **php** para mostrar los datos de las temperaturas y poder modificar las temperaturas de la climatización del hogar.

En la web podremos modificar las temperaturas del termostato, desconectarlo, ver las temperaturas del acuario y las del hogar (interior y exterior).

Aquí solo vamos a mostrar las temperaturas del acuario, ya que el acuario es autónomo y no se puede modificar nada porque sus parámetros son estrictos y no se deberían alterar para que la vida de sus habitantes sea lo más similar posible al entorno natural.

4.4.3.1-. Mostrar climatización doméstica web y php.

Para mostrar las temperaturas en la web necesitamos hacer las consultas a la base de datos del servidor, para ello utilizaremos el lenguaje php.

Lo primero es conectar con la base de datos y comprobar que estamos conectados a ella:

- Creamos un nuevo objeto con **new mysqli()** e insertamos: la dirección **ip** de la red, el usuario, en este caso **root;** la contraseña y la base de datos a la que nos conectamos, que en nuestro caso y recordamos secciones anteriores es **Temperaturas**. Con esta línea ya tendremos conexión a la base de datos guardada en **$mysqli**. Ilustración 1: 4.4.3.1

```
$mysqli = new mysqli("localhost", "root", "contraseña", "Temperaturas");
```

Ilustración 1: 4.4.3.1

- Ahora comprobamos si la conexión en correcta, si es correcta continuará el script, si no entrara en el condicional, nos mostrará un error. Ilustración 2: 4.4.3.1

```php
if ($mysqli->connect_errno) {
    header("HTTP/1.0 500 Internal Server Error");
    exit();
}
```

Ilustración 2: 4.4.3.1

- **GetUltimaTemperaturaHogar** es la función para ejecutar la consulta de la última temperatura en el interior y exterior del hogar registrada en la base de datos, para poder mostrarla en la web. Ilustración 1: 4.4.2.5
  - La función recoge mediante argumentos un objecto de tipo **$mysqli** que es la conexión a la base de datos explicada arriba.
  - Con **$mysqli->prepare(string query)** preparamos la consulta **SQL** que, en este caso, significa que vamos a descargarnos los campos **tempInterior**, **tempExterior** y **fechaCaptura** de la tabla (**FROM**) **tempsHogar** ordenada por (O**RDER BY**) el campo **fechaCaptura** de modo descendente (**DESC**) y solo una fila (**LIMIT 1**) y esta consulta la guardaremos en la variable **$consulta.**
  - Con **$consulta -> execute():** ejecutamos la consulta**.**
  - Con **$consulta ->bind_result:** Vinculamos los campos o filas devueltos a una variable de almacenamiento que llamamos **$respuesta**.
  - Con **$consulta→fetch():** Obtenemos una fila o campos de resultados como un **array** numérico.
  - Con **return $respuesta**: devolvemos la consulta para poder guardarla en una variable.

```php
function getUltimaTemperaturaHogar($mysqli) {
    $consulta = $mysqli->prepare("SELECT tempInterior, tempExterior,
                                  fechaCaptura FROM tempsHogar
                                  ORDER BY fechaCaptura DESC LIMIT 1");
    $consulta->execute();
    $consulta->bind_result($respuesta['tempInterior'], $respuesta['tempExterior'],
                                  $respuesta['fechaCaptura']);
    $consulta->fetch();
    return $respuesta;
}
```

Ilustración 3: 4.4.3.1

- **GetTemperaturaMediaHogar** es la función para ejecutar la consulta de la media de temperaturas en el interior y exterior del hogar en un día, en la base de datos, para poder mostrarla en la web. Ilustración 4: 4.4.3.1

- La función recoge, mediante argumentos, un objecto de tipo **$mysqli** que es la conexión a la base de datos explicado arriba.
- Con **$mysqli->prepare(string query)** preparamos la consulta **SQL** que, en este caso, significa que vamos a descargarnos de los campos las medias (**AVG**) de **tempInterior**, **tempExterior** de la tabla (**FROM**) **tempsHogar** donde (**WHERE**) el campo **fechaCaptura** será **mayor o igual (>=)** que el resultado de la resta de la fecha actual del sistema y el intervalo (**INTERVAL**) de las fechas de un día, y esta consulta la guardaremos en la variable **$consulta.**
- Con **$consulta -> execute()** ejecutamos la consulta**.**
- Con **$consulta →bind_result, v**inculamos los campos o filas devueltos a una variable de almacenamiento que llamamos **$respuesta**.
- Con **$consulta→fetch()**, obtenemos una fila o campos de resultados como un **array** numérico.
- Con **return $respuesta**, devolvemos la consulta para poder guardarla en una variable.

```
function getTemperaturaMediaHogar($mysqli) {
    $consulta = $mysqli->prepare("SELECT AVG(tempInterior), AVG(tempExterior)
                                  FROM tempsHogar
                                  WHERE fechaCaptura >= SYSDATE() - INTERVAL 1 DAY");
    $consulta->execute();
    $consulta->bind_result($respuesta['tempInterior'], $respuesta['tempExterior']);
    $consulta->fetch();
    return $respuesta;
}
```

Ilustración 4: 4.4.3.1

- En este paso vamos a ejecutar estas dos funciones y las vamos a guardar en dos variables diferentes para mostrarlas en nuestra web. Ilustración 4b: 4.4.3.1
  - **$ultimaTemperatura**: guardará el resultado de la función **getUltimaTemperaturaHogar($mysqli)**.
  - **$temperaturaMedia**: guardará el resultado de la función **getTemperaturaMediaHogar($mysqli)**.

```
$ultimaTemperaturaPecera = getLastTempsPecera($mysqli);
$temperaturaMediaPecera = getAVGTempsPecera($mysqli);
```

Ilustración 4b: 4.4.3.1

- Ahora vamos a desarrollar el código **html** y **php** que mostrará la información de las temperaturas, para explicarlo más claro vamos a partirlo por ámbitos resaltados por rectángulos de colores; este código pertenece solo a la temperatura interior: Ilustración 5: 4.4.3.1

- Rectángulo azul: En este ámbito la primera línea es el título, que en este caso es la climatización doméstica con una fuente de 20, en la siguiente línea ejecutamos la clase de **sensores** con ** ** dentro de un contenedor ‘**div**[8]’ que le va a dar una diseño agradable con **css**[9]. En la última línea vamos a utilizar dos contenedores de la clase **label** que se imprimarán en el sitio adecuado en la web.
- Rectángulo verde: En esta ámbito se ejecutan clases de **css** que dibujan unos anillos de diferentes colores, el primer contenedor **div** se refiere al color del anillo exterior, en este caso es el verde, los siguientes se mezclan para implementar un diseño agradable de varios colores y el último es un anillo blanco.
- Rectángulo rojo: En este ámbito vamos a mostrar las consultas guardadas en las variables explicadas anteriormente. Este código en **php** está dentro del último **div** del ámbito del rectángulo verde.

  La temperatura la tenemos guardada en la variable **tempParts** con el alias de **tempInterior** y está guardada en dos partes: la entera y la decimal, por eso tenemos un punto en la mitad de los códigos que imprimen la temperatura final.
- Rectángulo amarillo: En este ámbito estamos cerrando todos los contenedores abiertos en el ámbito del rectángulo verde.

```
<FONT SIZE = 20>Climatización Doméstica</font></h1></div>
 <div class="sensores">
   <div class="label">Temperatura Interior</div><div class="label">Temperatura Exterior</div>
      <div class="del">
        <div class="den">
          <div class="dene">
            <div class="denem">
              <div class="deneme">
                <?php print tempParts($ultimaTemperatura['tempInterior'], 0); ?>
                <span>.<?php  print tempParts($ultimaTemperatura['tempInterior'], 1); ?>
                </span><strong>&deg;</strong>
              </div>
            </div>
          </div>
        </div>
</div>
```

Ilustración 5: 4.4.3.1

8 Se emplea para definir un bloque de contenido o sección de la página, para poder aplicarle diferentes estilos e incluso para realizar operaciones sobre ese bloque específico.

9 CSS(**C**ascading **S**tyle **S**heets) permite crear páginas web con un diseño agradable, por medio de un código específico.

- Ahora vamos a desarrollar el código **html** y **php** que mostrará la información de las temperaturas, para explicarlo más claro vamos a dividirlo en rectángulos de colores, este código pertenece solo a temperatura: Ilustración 6: 4.4.3.1
  - Rectángulo azul: En esta ámbito vamos a ejecutar clases de **css** que dibujan unos anillos de diferentes colores, el primer contenedor **div** se refiere al color del anillo exterior, en este caso es el verde, los siguientes se mezclan para implementar un diseño agradable de varios colores y el último es un anillo blanco.
  - Rectángulo rojo: En este ámbito vamos a mostrar las consultas guardadas en las variables explicadas anteriormente. Este código en **php** está dentro del último **div** del ámbito del rectángulo azul.

    La temperatura la tenemos guardada en la variable **tempParts** con el alias de **tempExterior** y está guardada en dos partes: la entera y la decimal, por eso tenemos un punto en la mitad de los códigos que imprimen la temperatura final.
  - Rectángulo amarillo: En este ámbito estamos cerrando todos los contenedores abiertos en el ámbito del rectángulo azul.
  - Rectángulo verde: En esta ámbito utilizamos las clase **details (**dentro de contenedores **div**) de **css.** Para mostrar la temperatura completa necesitamos la función de **php number_format()** cuyo cometido es insertar el número entero y los decimales deseados, en nuestro caso uno, así que insertamos dentro de esta función la variable guardada de **$temperaturaMedia[con su campo]** y nos formateará el número entero con un decimal directamente. La última línea es para insertar la última temperatura registrada y le vamos a dar el formato a la fecha, donde el formato es **d** (día), **m** (mes), **Y** (año), **H** (hora), **i** (minuto) y **S** (segundos), con la función de **php strtotime(),** .

Ilustración 6: 4.4.3.1

- El resultado final de todo el código **html** y **php** que hemos explicado en parágrafos anteriores se puede observar en esta Ilustración 7: 4.4.3.1

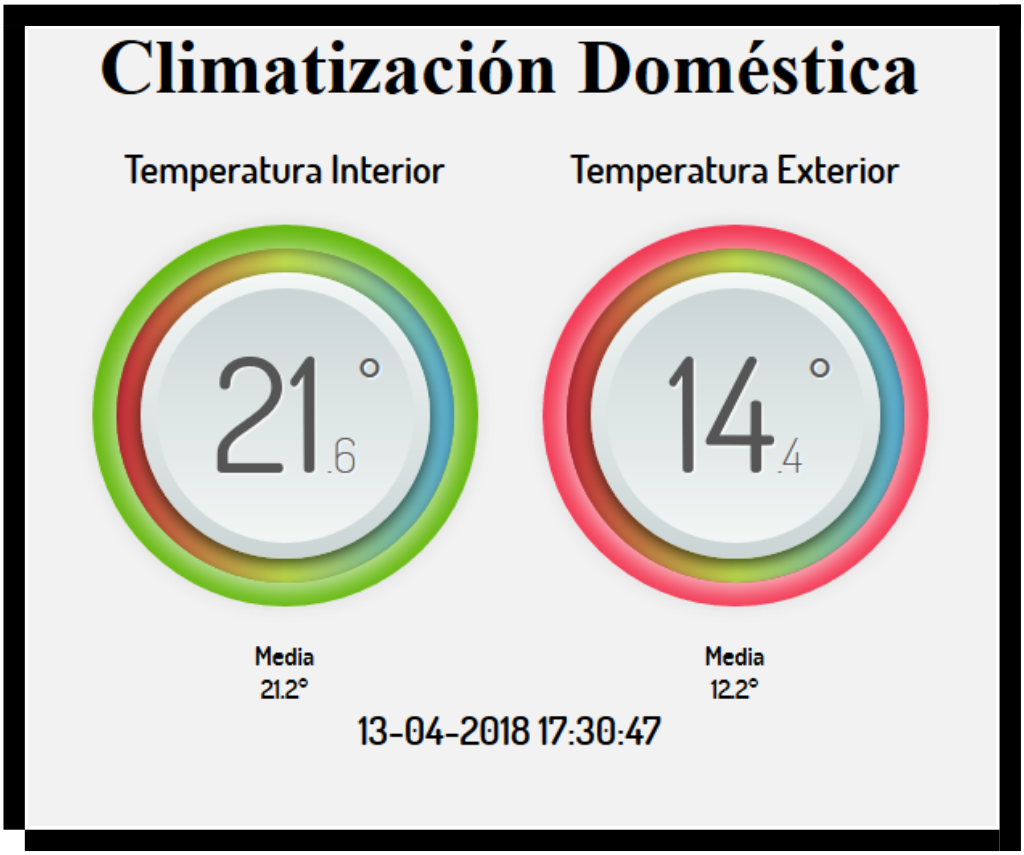


Ilustración 7: 4.4.3.1

- **GetUltimaTemperaturaPecera** es la función para ejecutar la consulta de la última temperatura registrada del agua del acuario y de la salida de la bomba de agua en la base de datos, para poder mostrarla en la web. Ilustración 8: 4.4.3.1
  - La función recoge mediante argumentos un objecto de tipo **$mysqli** que es la conexión a la base de datos explicada arriba.
  - Con **$mysqli->prepare(string query)** preparamos la consulta **SQL** que, en este caso, significa que vamos a descargarnos los campos **temperaturaPecera**, **temperaturaSalidaAgua** y **estado** de la tabla (**FROM**) **TemperaturaPecera** ordenada por (**ORDER BY**) el campo **idTemperaturaPecera** de modo descendente (**DESC**) y solo una fila (**LIMIT 1**); esta consulta la guardaremos en la variable **$consulta.**
  - Con **$consulta -> execute()** ejecutamos la consulta**.**
  - Con **$consulta ->bind_result** vinculamos los campos o filas devueltos a una variable de almacenamiento que llamamos **$respuesta**.

- Con **$consulta→fetch()** obtenemos una fila o campos de resultados como un **array** numérico.
- Con **return $respuesta** devolvemos la consulta para poder guardarla en una variable.

```
function getUltimaTemperaturaPecera($mysqli) {
    $consulta = $mysqli->prepare("SELECT temperaturaPecera, temperaturaSalidaAgua, estado
                        FROM TemperaturaPecera ORDER BY idTemperaturaPecera DESC LIMIT 1");
    $consulta->execute();
    $consulta->bind_result($respuesta['temperaturaPecera'], $respuesta['temperaturaSalidaAgua'],
                        $respuesta['estado']);
    $consulta->fetch();
    return $respuesta;
}
```

Ilustración 8: 4.4.3.1

- Esta función sirve para separar los números guardados en decimal y escoger los apropiados como ya veremos en ejemplos posteriores. Ilustración 9: 4.4.3.1

```
function tempParts($temp, $index) {
    $parts = explode('.', number_format($temp, 1));
    return $parts[$index];
}
```

Ilustración 9: 4.4.3.1

- **GetTemperaturaMediaPecera** es la función para ejecutar la consulta de la media de temperaturas del agua del acuario y de la salida de la bomba de agua en un día en la base de datos, para poder mostrarla en la web. Ilustración 10: 4.4.3.1
  - La función recoge mediante argumentos un objecto de tipo **$mysqli** que es la conexión a la base de datos explicado arriba.
  - Con **$mysqli->prepare(string query)** preparamos la consulta **SQL** que, en este caso, significa que vamos a descargarnos de los campos las medias (**AVG**) de **temperaturaPecera**, **temperaturaSalidaAgua** de la tabla (**FROM**) **TemperaturaPecera;** esta consulta la guardaremos en la variable **$consulta.**
  - Con **$consulta -> execute()** ejecutamos la consulta**.**
  - Con **$consulta →bind_result** vinculamos los campos o filas devueltos a una variable de almacenamiento que llamamos **$respuesta**.
  - Con **$consulta→fetch()** obtenemos una fila o campos de resultados como un **array** numérico.

- Con **return $respuesta** devolvemos la consulta para poder guardarla en una variable.

```
function getTemperaturaMediaPecera($mysqli) {
    $consulta = $mysqli->prepare("SELECT AVG(temperaturaPecera), AVG(temperaturaSalidaAgua)
                                  FROM TemperaturaPecera");
    $consulta->execute();
    $consulta->bind_result($respuesta['temperaturaPecera'], $respuesta['temperaturaSalidaAgua']);
    $consulta->fetch();
    return $respuesta;
}
```

Ilustración 10: 4.4.3.1

- En este paso vamos a ejecutar estas dos funciones y las vamos a guardar en dos variables diferentes para mostrarlas en nuestra web. Ilustración 11: 4.4.3.1
  - **$ultimaTemperaturaPecera** guardará el resultado de la función **getUltimaTemperaturaPecera($mysqli)**.
  - **$temperaturaMediaPecera** guardará el resultado de la función **getTemperaturaMediaPecera($mysqli)**.

```
$ultimaTemperaturaPecera = getUltimaTemperaturaPecera($mysqli);
$temperaturaMediaPecera = getTemperaturaMediaPecera($mysqli);
```

Ilustración 11: 4.4.3.1

- Por último, tenemos una función **SacarEstado($numero)** que nos indicará el estado del agua del acuario, esta función recoge en la variable **$numero** de tipo integer, guardada en la base de datos, un número: 0 si está muy caliente, 1 si está caliente, 2 si es óptima, 3 si está fría y 4 si está muy fría. Ilustración 12: 4.4.3.1

```php
function SacarEstado($numero){
    if($numero == 0){
        return 'Muy Caliente';
    }else if($numero == 1){
        return 'Caliente';
    }else if($numero == 2){
        return 'Óptima';
    }else if($numero == 3){
        return 'Fria';
    }else if($numero == 4){
        return 'Muy Fria';
    }else{
        return 'No existe';
    }
}
```

Ilustración 12: 4.4.3.1

- Ahora vamos a desarrollar el código **html** y **php** que mostrará la información de las temperaturas del acuario, muy parecido al mostrado por las temperaturas del hogar pero cambiando las variables del hogar guardadas por las variables del acuario almacenadas. Solo tenemos una diferencia que en la parte de la climatización del acuario mostramos el estado del agua en vez de la fecha. Ilustración 13: 4.4.3.1
    - Como hemos visto anteriormente, ejecutamos el método **SacarEstado,** le insertamos la variable guardada **$ultimaTemperaturaPecera,** a continuación, elegimos el campo **estado,** que es un número entre 0 y 4 de la consulta, y se nos mostrará el estado del agua.

```php
<div class="last-update"><h1>Estado del agua: <?php print
(SacarEstado($ultimaTemperaturaPecera['estado'])); ?></h1></div>
```

Ilustración 13: 4.4.3.1

- El resultado final de todo el código **html** y **php** que hemos explicado en parágrafos anteriores lo podemos observar en esta Ilustración 14: 4.4.3.1

Ilustración 14: 4.4.3.1

4.4.3.2.- Actualización de la climatización domestica web y php.

Para actualizar las temperaturas en la web necesitamos hacer las actualizaciones a la base de datos del servidor, para ello utilizaremos el lenguaje tipado **php** y actualizaremos los campos por medio de **inputs *de* html** que se activarán al pulsar unos botones.

- Este método hace la consulta a la base de datos y, según lo que tengamos almacenado en la misma, devuelve si el termostato está activado o desactivado. Ilustración 1: 4.4.3.2

```
function verEncendidoTermostato(){

    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver = "SELECT aver FROM tabla_temperatura WHERE idtabla_temperatura='1'";
    $respuesta = mysql_query($aver);
    $row = mysql_fetch_array($respuesta);


    $row['aver'];
    if($row['aver'] == '1'){
        return 'Termostato Activo';
    }else{
        return 'Termostato Desactivado';

    }
```

Ilustración 1: 4.4.3.2

- Este método hace la consulta a la base de datos y, según lo que tengamos almacenado en la base de datos, devuelve 1 si está en modo verano (calor) o 0 si está en modo invierno (frío). Ilustración 2: 4.4.3.2

```
function verCalorFrio(){

    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver ="SELECT estacion FROM `Temperaturas`.`estacion`;";
    $respuesta = mysql_query($aver);
    $row = mysql_fetch_array($respuesta);


    $row['estacion'];
    if($row['estacion'] == '1'){
        return 'Modo Verano';
    }else{
        return 'Modo Invierno';

    }
```

Ilustración 2: 4.4.3.2

- Este método hace la consulta a la base de datos y, según lo que tengamos almacenado en la base de datos, devuelve 1 si el modo manual está activado o 0 si está desactivado. Ilustración 3: 4.4.3.2

```
function verManual(){

    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver ="SELECT maquina_manualcol FROM Temperaturas.maquina_manual;";
    $respuesta = mysql_query($aver);
    $row = mysql_fetch_array($respuesta);


    $row['maquina_manualcol'];
    if($row['maquina_manualcol'] == '1'){
        return 'Modo Manual Activo';
    }else{
        return 'Modo Manual Desactivado';

    }


}
```

Ilustración 3: 4.4.3.2

- Este método hace la consulta a la base de datos y, según lo que tengamos almacenado en la base de datos, devuelve 1 si está en funcionamiento o 0 si está apagada. Ilustración 4: 4.4.3.2

```
function verMaquina(){

    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver ="SELECT maquinaEncendidacol
            FROM Temperaturas.maquinaEncendida;";
    $respuesta = mysql_query($aver);
    $row = mysql_fetch_array($respuesta);

    $row['maquinaEncendidacol'];
    if(($row['maquinaEncendidacol'] == '1' &&
        verEncendidoTermostato() == 'Termostato Activo') ||
        verManual() == 'Modo Manual Activo'){
        return 'En Funcionamiento';
    }else{
        return 'Apagada';

    }


}
```

Ilustración 4: 4.4.3.2

- Estos tres métodos hacen las consultas a la base de datos y según lo que tengamos almacenado en la base de datos devuelven las temperaturas que queramos en verano: mínima, media, máxima. Ilustración 5: 4.4.3.2

```php
function verTemperaturaVeranoMin(){
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $minima ="SELECT minima FROM Temperaturas.temperatura_verano;";
    $respuesta1 = mysql_query($minima);
    $row1 = mysql_fetch_array($respuesta1);
    return $row1['minima'];
}

function verTemperaturaVeranoMed(){
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $media ="SELECT media FROM Temperaturas.temperatura_verano;";
    $respuesta2 = mysql_query($media);
    $row2 = mysql_fetch_array($respuesta2);
    return $row2['media'];
}

function verTemperaturaVeranoMax(){
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $maxima ="SELECT maxima FROM Temperaturas.temperatura_verano;";
    $respuesta3 = mysql_query($maxima);
    $row3 = mysql_fetch_array($respuesta3);
    return $row3['maxima'];
}
```

Ilustración 5: 4.4.3.2

- Estos tres métodos hacen las consultas a la base de datos y según lo que tengamos almacenado en la base de datos devuelven las temperaturas que queramos en invierno: mínima, media, máxima. Ilustración 6: 4.4.3.2

```php
function verTemperaturaInviernoMin(){
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $minima ="SELECT minima FROM Temperaturas.temperatura_invierno;";
    $respuesta1 = mysql_query($minima);
    $row1 = mysql_fetch_array($respuesta1);
    return $row1['minima'];
}

function verTemperaturaInviernoMed(){
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $media ="SELECT media FROM Temperaturas.temperatura_invierno;";
    $respuesta2 = mysql_query($media);
    $row2 = mysql_fetch_array($respuesta2);
    return $row2['media'];
}

function verTemperaturaInviernoMax(){
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $maxima ="SELECT maxima FROM Temperaturas.temperatura_invierno;";
    $respuesta3 = mysql_query($maxima);
    $row3 = mysql_fetch_array($respuesta3);
    return $row3['maxima'];
}
```

Ilustración 6: 4.4.3.2

- Con este código, al pulsar **on/off** se ejecutará este condicional que recibe los datos del formulario con el nombre de **termostatoon** y **termostatooff**, entonces cambiaremos de estado. Si nos fijamos, con la función de php **exec(“script a ejecutar”),** ejecutamos un **ShellScript** en **linux**. Las líneas siguientes son para actualizar el estado en la base de datos. Ilustración 7: 4.4.3.2

```php
if(isset($_POST['termostatoon'])){
    exec("/home/pi/termostato_enciende.sh");
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver = "UPDATE `Temperaturas`.`tabla_temperatura` SET `aver`='1' WHERE `idtabla_temperatura`='1'";
    mysql_query($aver);
}

if(isset($_POST['termostatooff'])){
    exec("/home/pi/termostato_apagar.sh");
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver = "UPDATE `Temperaturas`.`tabla_temperatura` SET `aver`='0' WHERE `idtabla_temperatura`='1'";
    mysql_query($aver);
}
```

Ilustración 7: 4.4.3.2

- Con este código, cuando pulsemos el botón de **manual on/off** se ejecutará este condicional que recibe los datos del formulario con el nombre **termostatomanualon** y **termostatomanualoff**, entonces cambiaremos de estado. Si nos fijamos, con la función de php **exec("script a ejecutar")** ejecutamos un **ShellScript** en **linux**. Las líneas siguientes son para actualizar el estado en la base de datos. Ilustración 8: 4.4.3.2

```php
if(isset($_POST['termostatomanualon'])){
    exec("/home/pi/termostato_enciende_manual.sh");
     mysql_connect("localhost","root","contraseña");
     mysql_select_db("Temperaturas");
     $aver = "UPDATE `Temperaturas`.`maquina_manual` SET `maquina_manualcol`='1' WHERE `idmaquina_manual`='1';";
     mysql_query($aver);
}

if(isset($_POST['termostatomannualoff'])){
    exec("/home/pi/termostato_apagar_manual.sh");
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver = "UPDATE `Temperaturas`.`maquina_manual` SET `maquina_manualcol`='0' WHERE `idmaquina_manual`='1';";
    mysql_query($aver);

}
```

Ilustración 8: 4.4.3.2

- Con este código, cuando pulsemos el botón de **calor** y **frío** se ejecutará este condicional que recibe los datos del formulario con el nombre **termostatocalor** y **termostatofrio**, entonces cambiaremos de estado. Si nos fijamos, con la función de php **exec("script a ejecutar")** ejecutamos un **ShellScript** en **linux**. Las líneas siguientes son para actualizar el estado en la base de datos. Ilustración 9: 4.4.3.2

```php
if(isset($_POST['termostatocalor'])){
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver = "UPDATE `Temperaturas`.`estacion` SET `estacion`='1' WHERE `idestacion`='1';";
    mysql_query($aver);

}
if(isset($_POST['termostatofrio'])){
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver = "UPDATE `Temperaturas`.`estacion` SET `estacion`='0' WHERE `idestacion`='1';";
    mysql_query($aver);

}
```

Ilustración 9: 4.4.3.2

- Con este código, cuando pulsemos el botón de **actualizar calor** o **frío** se ejecutará este condicional que recibe los datos del formulario con el nombre **actualizartemperaturasverano** y **actualizartemperaturasinvierno**, almacenaremos los datos del formulario, entonces actualizaremos los datos. Las líneas siguientes son para actualizar las temperaturas en la base de datos. Ilustración 10: 4.4.3.2

```php
if(isset($_POST['actualizartemperaturaverano'])){
    $minima=$_POST['tvmin'];
    $media=$_POST['tvmed'];
    $maxima=$_POST['tvmax'];
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver ="UPDATE `Temperaturas`.`temperatura_verano`
    SET `minima`='$minima', `media`='$media', `maxima`='$maxima'
    WHERE `idtemperatura_verano`='1'";
    mysql_query($aver);
}

if(isset($_POST['actualizartemperaturainvierno'])){
    $minima=$_POST['timin'];
    $media=$_POST['timed'];
    $maxima=$_POST['timax'];
    mysql_connect("localhost","root","contraseña");
    mysql_select_db("Temperaturas");
    $aver ="UPDATE `Temperaturas`.`temperatura_invierno`
    SET `minima`='$minima', `media`='$media', `maxima`='$maxima'
    WHERE `idtemperatura_invierno`='1'";
    mysql_query($aver);
}
```

Ilustración 10: 4.4.3.2

Para poder explicar mejor esta parte hemos dividido el código en ámbitos de colores: Ilustración 11: 4.4.3.2

- Rectángulo azul: Estamos mostrando el apartado correspondiente a Calefacción/Refrigeración. En la siguiente línea de **html** con **php** mostramos el estado de la máquina, por eso llamamos a la función **verMaquina()** que nos devolverá el estado de la máquina (encendida o apagada).
- Rectángulo rojo: Estamos mostrando si el termostato está activado, por eso llamamos con código **php** a la función **verEncendidoTermostato()** que nos devolverá si está activado o desactivado.
- Rectángulo verde: Estamos mostrando los botones de activar y desactivar el termostato, si pulsamos **On** desencadenará un evento recogido por una variable de tipo **$_POST** con la etiqueta **termostatoon** y, entonces, ejecutará el **script** que activará el termostato y pasará a **off** pero con **termostatooff.**
- Rectángulo lila: La primera línea mostrará con la función **verManual()** si el manual está activado o desactivado, en la línea siguiente estamos mostrando los botones de activar y desactivar del funcionamiento manual, si pulsamos **On** desencadenará un evento que será recogido por una variable de tipo **$_POST** con la etiqueta **termostatomanualon** y, entonces, ejecutará el **script** que activará el manual y pasará a **off** pero con **termostatomanualoff.**
- Rectángulo amarillo: La primera línea mostrará con la función **verCalorFrio()** si la máquina está en modo verano (calor) o en modo invierno (frío), en la línea siguiente estamos mostrando los botones de activar y desactivar el funcionamiento de verano o invierno, si pulsamos **Calor** desencadenará un evento que será recogido por una variable de tipo **$_POST** con la etiqueta **termostatocalor** y, entonces, ejecutará el **script** que activará el modo verano; pasará lo mismo con **off** pero con **termostatomanualoff** y pasará a modo invierno**.**

Ilustración 11: 4.4.3.2

- Esta imagen nos muestra el resultado del código desarrollado anteriormente. Ilustración 12: 4.4.3.2

Ilustración 12: 4.4.3.2

Para poder explicar mejor esta parte hemos dividido el código de la asignación de temperaturas de verano por ámbitos de colores : Ilustración 13: 4.4.3.2

- Rectángulo rojo: En la primera línea llamamos a la función **verTemperaturaVeranoMin()** para ver la temperatura mínima a la que está asignada, en la segunda línea tenemos un **input** de tipo **text** que es un campo de texto, que nos servirá para actualizar la temperatura mínima de verano y, para obtener más información, nos muestra la temperatura con el método **verTemperaturaVeranoMin().** Para activar el evento de actualización, hemos asignado al **input** el nombre de **tvmin** para que cuando pulsemos el botón de actualizar recoja la temperatura y la actualice en la base de datos.
- Rectángulo lila: En la primera línea llamamos a la función **verTemperaturaVeranoMed()** para ver la temperatura media a la que está asignada, en la segunda línea tenemos un **input** de tipo **text** que es un campo de texto, que nos servirá para actualizar la temperatura media de verano y, para obtener más información, nos muestra la temperatura con el método **verTemperaturaVeranoMed().** Para activar el evento de actualización le hemos

asignado al **input** el nombre de **tvmed** para que cuando pulsemos el botón de actualizar recoja la temperatura y la actualice en la base de datos.

- Rectángulo amarillo: En la primera línea llamamos a la función **verTemperaturaVeranoMax()** para ver la temperatura máxima que está asignada, en la segunda línea tenemos un **input** de tipo **text** que es un campo de texto, que nos servirá para actualizar la temperatura máxima de verano y, para obtener más información, nos muestra la temperatura con el método **verTemperaturaVeranoMax().** Para activar el evento de actualización le hemos asignado al **input** el nombre de **tvmax** para que cuando pulsemos el botón de actualizar recoja la temperatura y la actualice en la base de datos.
- Rectángulo verde: Aquí tenemos el botón que lanzará el evento de actualización de todos los **inputs** con campos de asignación de temperaturas de verano que se asigna con **name** y el nombre del evento que es **actulizartemperaturaverano**.

```
<h3>Temperaturas Verano</h3>
  
<h4>Temperatura Mínima Actual: <?php echo verTemperaturaVeranoMin() ?> C</h4>
Mín:<input type="text" name="tvmin" style="border:1px solid #000; font-size:20px;" value="<?php echo verTemperaturaVeranoMin() ?>">
<h4>Temperatura Media Actual: <?php echo verTemperaturaVeranoMed() ?> C</h4>
Med:<input type="text" name="tvmed" style="border:1px solid #000; font-size:20px;" value="<?php echo verTemperaturaVeranoMed() ?>">
<h4>Temperatura Máxima Actual: <?php echo verTemperaturaVeranoMax() ?> C</h4>
Máx:<input type="text" name="tvmax" style="border:1px solid #000; font-size:20px;" value="<?php echo verTemperaturaVeranoMax() ?>">
<input type="submit" style="border:1px solid #000; font-size:20px;" value="Actualizar" name="actualizartemperaturaverano">
<br>  
</form>
```

Ilustración 13: 4.4.3.2

Para poder explicar mejor esta parte hemos dividido el código de la asignación de temperatura de invierno por ámbitos de colores: Ilustración 14: 4.4.3.2

- Rectángulo rojo: En la primera linea llamamos a la función **verTemperaturaInviernoMin()** para ver la temperatura mínima que está asignada, en la segunda línea tenemos un **input** de tipo **text** que es un campo de texto, que nos servirá para actualizar la temperatura mínima de invierno y, para obtener más información, nos muestra la temperatura con el método **verTemperaturaVeranoMin().** Para activar el evento de actualización le hemos asignado al input el nombre de **timin** para que cuando pulsemos el botón de actualizar recoja la temperatura y la actualice en la base de datos.
- Rectángulo lila: En la primera línea llamamos a la función **verTemperaturaInviernoMed()** para ver la temperatura media que está asignada, en la segunda línea tenemos un **input** de tipo **text** que es un campo de texto, que nos servirá para actualizar la temperatura media de invierno y, para que tengamos más información también, se nos muestra la temperatura con el método **verTemperaturaInviernoMed().** Para activar el evento de actualización le hemos

asignado al **input** el nombre de **timed** para que cuando pulsemos el botón de actualizar recoja la temperatura y la actualice en la base de datos.

- Rectángulo amarillo: En la primera línea llamamos a la función **verTemperaturaInviernoMax()** para ver la temperatura máxima que está asignada, en la segunda línea tenemos un **input** de tipo **text** que es un campo de texto, que nos servirá para actualizar la temperatura máxima de invierno y, para que tengamos más información, se nos muestra la temperatura con el método **verTemperaturaInviernoMax().** Para activar el evento de actualización le hemos asignado al input el nombre de **timax** para que cuando pulsemos el botón de actualizar recoja la temperatura y la actualice en la base de datos.
- Rectángulo verde: Aquí tenemos el botón que lanzará el evento de actualización de todos con campos de asignación de temperaturas de verano que se asigna con **name** y el nombre del evento que es **actulizartemperaturainvierno**.

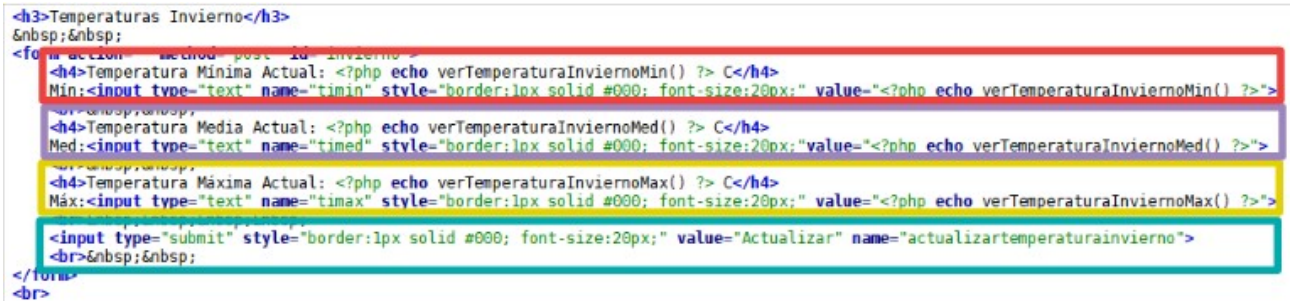


Ilustración 14: 4.4.3.2

- Esta imagen nos muestra el resultado del código desarrollado anteriormente. Ilustración 15: 4.4.3.2

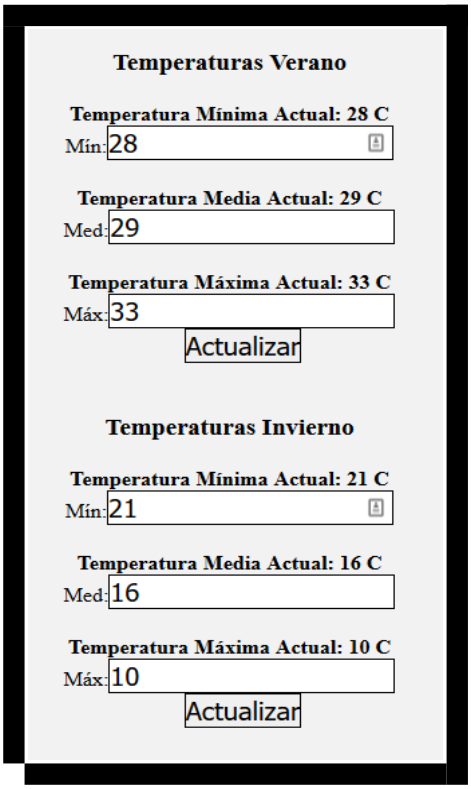


Ilustración 15: 4.4.3.2

4.4.3.4.- ShellScripts utilizados en climatización web y php.

Para apagar y encender la máquina de la climatización en web y php, tenemos cuatro **shellscripts** que ejecutamos en secciones anteriores, vamos a exponerlos en esta sección.

- **termostato_enciende_manual.sh**: cambiamos el **gpio18** en modo salida (**out**) y después con **write** lo cambiamos a 0 y se encenderá la máquina. Ilustración 1: 4.4.3.4

```
#!/bin/bash
gpio -g mode 18 out
gpio -g write 18 0
```

Ilustración 1: 4.4.3.4

- **termostato_apagar_manual.sh**: cambiamos el **gpio18** con **write** a 1 y después lo cambiamos en modo entrada (**in**) y se apaga la máquina. Ilustración 2: 4.4.3.4

```
#!/bin/bash
gpio -g write 18 1
gpio -g mode 18 in
```

Ilustración 2: 4.4.3.4

- **termostato_enciende.sh**: cambiamos el pin a modo salida(**out**) y se activa el termostato. Ilustración 3: 4.4.3.4

```
#!/bin/bash
gpio -g mode 18 out
```

Ilustración 3: 4.4.3.4

- **termostato_apagar.sh**: cambiamos el pin a modo entrada (**in**) y se desactiva el termostato. Ilustración 4: 4.4.3.4

```
#!/bin/bash
gpio -g mode 18 in
```

Ilustración 4: 4.4.3.4

4.4.3.5-. Diseño web con hojas de estilo (CSS).

Vamos a comentar esta sección a través de las imágenes y el código **html,** hemos incluido estilos, pero no vamos a profundizar en ellos porque no afectan en nada en el funcionamiento de la web y es meramente estético.

Te permite aplicar diseños de manera selectiva por medio de etiquetas a elementos escritos en **html**. Aquí mostramos cómo programamos el diseño seleccionado.

- Si nos fijamos en el rectángulo azul estamos asignando el estilo a nuestro contenedor **div** por medio de **class=”de2”**. Ilustración 1: 4.4.2.5

```
<div class="de2">
    <div class="dene">
      <div class="denem">
        <div class="deneme">
          <?php print tempParts($ultimaTemperatura['tempExterior'], 0); ?>
          <span>.<?php print tempParts($ultimaTemperatura['tempExterior'], 1); ?>
          </span><strong>&deg;</strong>
```

Ilustración 1: 4.4.2.5

- La clase .de2 (Ilustración 2: 4.4.3.5) está dibujando un anillo rojo que es el que vemos en la imagen de la Ilustración 3: 4.4.3.5.

```
.de2 {
    position: relative;
    float: left;
    margin: 20px;
    width: 240px;
    height: 240px;
    border-radius: 100%;
    box-shadow: 0 0 10px rgba(0, 0, 0, .1);
    background-color: #F2304D;
}
```

Ilustración 2: 4.4.3.5

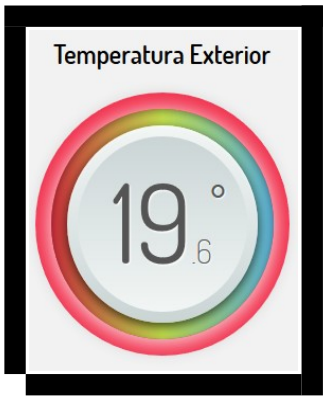
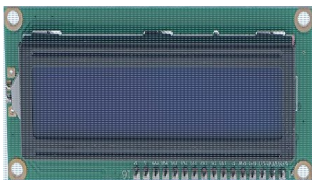
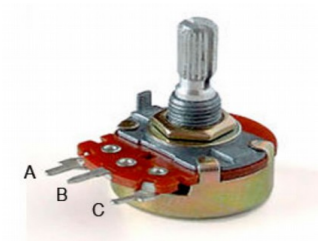
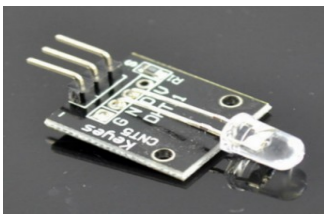
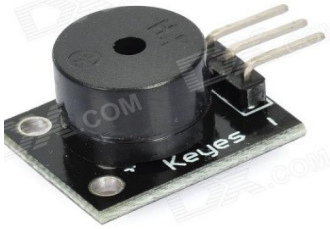
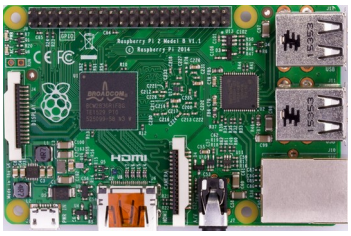


Ilustración 3: 4.4.3.5

5-. Presupuesto

Artículo	Imagen	Modelo	Precio
Sensor de Agua		2706625	2€
Arduino Uno		CH340G	6€
4 Sensores de temperatura		DS18B20	8€
Pantalla LCD Módulo de Carácter Azul		IIC I2C Serie 1602 16x2	4€
Sensor de humo		MQ-2 Metano, butano, GLP, humo	1€
Potenciómetro		Potenciómetro Giratorio B10K 10K ohm Lineal Simple	1,30€
Módulos de Relés de 4 Canales a DC 5V con Optoacoplador para Arduino UNO		ES-EL-SM-006	5€

Arduino KY-034 Automatic flashing colorful LED module		YB-3120B4PnYG-PM	1€
Arduino KY-006 Small passive buzzer module		KY-006	1€
Pulsador switch 12mm		CPM-0039	0,70€
Módulo esclavo bluetooth HC-06		HC-06	20€
RaspBerry Pi 2		Modelo B	40€
Cargador de alimentación		Raspberry Pi 2.5A 5 V 3	4€

Micro SD 4Gb			5€
Fotorresistencia LDR			2€
Conexión de 10 cables			10€
Bluetooth USB 2.0			2,50€
Calentador Refrigerador 12v.			25€

5m de tubo transparente de diámetro 3.2mm.			5€
4 disipadores agua CPU			8€

En la hoja de cálculo que está a continuación presentamos un hipotético presupuesto (850,21€) con el coste real del proyecto si se comercializara.

Como vemos en la imagen lo que más encarecería el producto final son las horas de trabajo que ascenderían a 600€ (20 horas x 30€/hora) ya que el hardware no tiene un coste muy alto y es fácilmente asumible.

Materiales	Unidades	Precio Unitario	Importe	IVA %	Total con IVA
Sensor de agua	2	2 €	4 €	21,00 %	**4,84 €**
Arduino Uno	1	6 €	6 €	21,00 %	**7,26 €**
Sensor de temperatura	4	2 €	8 €	21,00 %	**9,68 €**
Pantalla LCD	1	4 €	4 €	21,00 %	**4,84 €**
Sensor Humo	1	1 €	1 €	21,00 %	**1,21 €**
Potenciometro	1	1 €	1 €	21,00 %	**1,21 €**
Relé de 6 canal	1	5 €	5 €	21,00 %	**6,05 €**
Led KY-034 y KY-0006	2	1 €	2 €	21,00 %	**2,42 €**
Disipadores CPU	4	2 €	8 €	21,00 %	**9,68 €**
Pulsador	1	1 €	1 €	21,00 %	**0,85 €**
Bluetooth HC-06	1	20 €	20 €	21,00 %	**24,20 €**
RaspBerry Pi 2	1	40 €	40 €	21,00 %	**48,40 €**
Alimentacion	2	4 €	8 €	21,00 %	**9,68 €**
Micro SD	1	5 €	5 €	21,00 %	**6,05 €**
Fotoresistencia	1	2 €	2 €	21,00 %	**2,42 €**
Cables conectores	10	1 €	10 €	21,00 %	**12,10 €**
Bluetooth USB	1	3 €	3 €	21,00 %	**3,03 €**
Calentador- Refrigerador	1	25 €	25 €	21,00 %	**30,25 €**
Tubo transparente de 3.1mm y	5	1 €	5 €	21,00 %	**6,05 €**
Total			157,20 €	0,00 %	**190,21 €**
Personal - Mano de Obra			600,00 €	10,00 %	**660,00 €**
			0,00 €		**0,00 €**

Total Presupuesto				
Total sin IVA			**757,20 €**	
Base imponible			**757,20 €**	
I.V.A.	4,00 %	10,00 %	21,00 %	
Total I.V.A.	0,00 €	60,00 €	33,01 €	**93,01 €**
			Total Presupuesto	**850,21 €**

6-. Diseño de placas shield.

En caso de querer suprimir el cableado de las conexiones a los periféricos y facilitar la reparación de cualquier sensor hemos diseñado las placas de **shield** de conexión de **raspberry pi** y **arduido uno.**

El diseño lo hemos realizado con **Kicad**, la aplicación de código abierto para la creación de esquemas electrónicos y diseño de placas de circuito impreso.

6.1-. Placa Raspberry Pi

Como podemos ver en la Ilustración 1: 6.1 hemos colocado la conexión de 40 pines que que se conectarán a la **GPIO** de la **Raspberry pi**.

Tenemos la conexión de los sensores de temperatura y la del sensor de luz.

También están la resistencia necesaria para los sensores de temperatura y el condensador para el sensor de luz **LDR**.

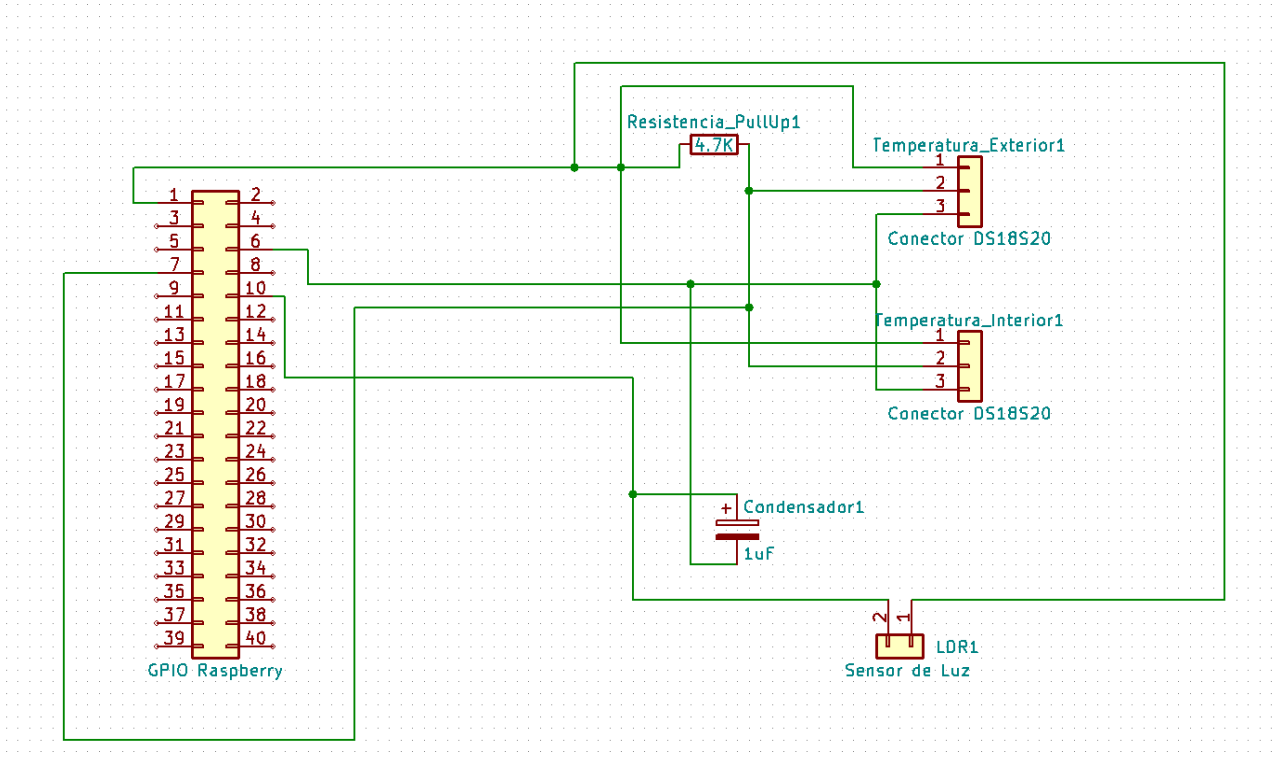


Ilustración 1: 6.1

6.2-. Placa Arduino Uno.

Como podemos ver en la Ilustración 1: 6.2 en esta placa ya tenemos todos los conectores de los sensores agregados. Esta placa es más complicada ya que desarrollar el enrutado de las redes es más complicado.

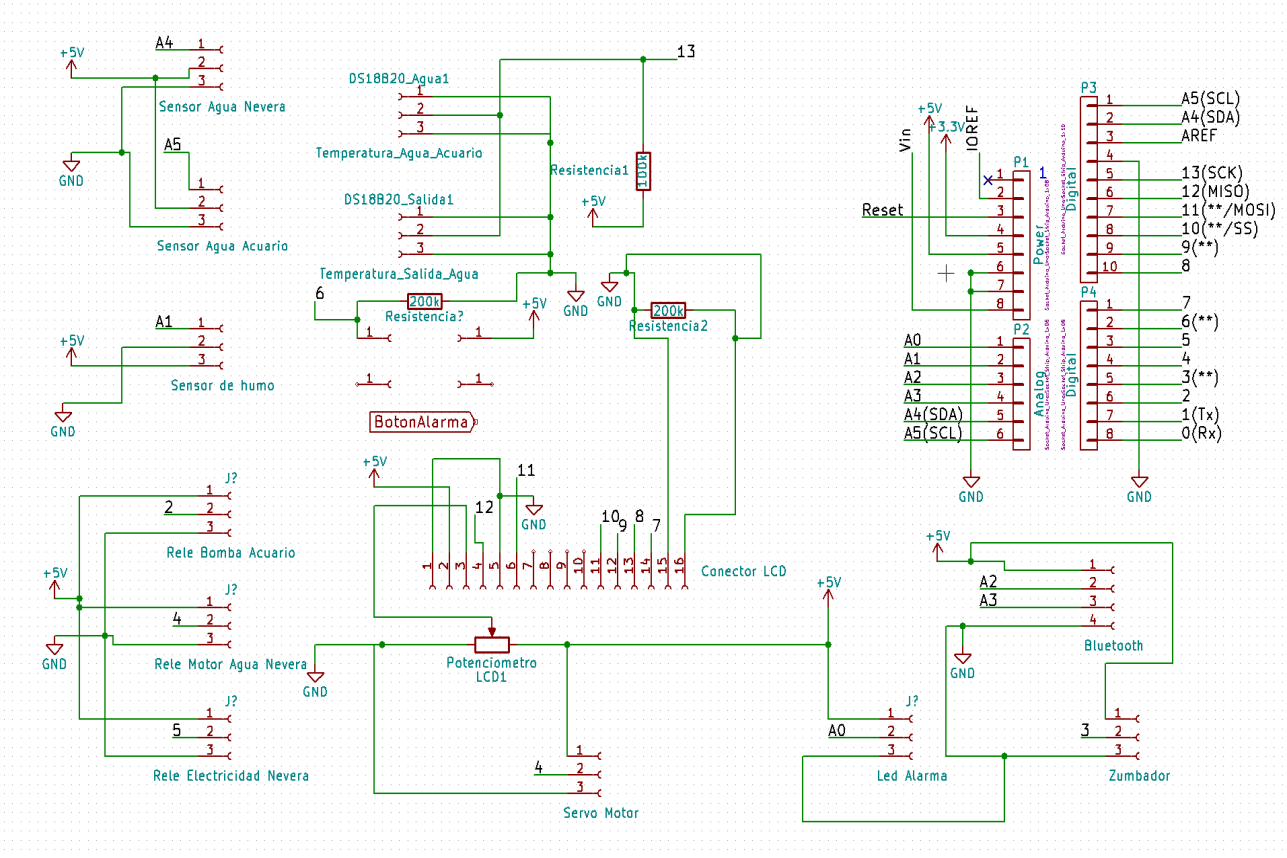


Ilustración 1: 6.2

7-. Referencias bibliográficas y web-grafía

- http://cetroniconline.blogspot.com.es/2014/07/tutorial-arduino-iv-sensor-de.html
- https://programarfacil.com/tutoriales/fragmentos/arduino/texto-en-movimiento-en-un-lcd-con-arduino/#LCD_Liquid_Crystal_Display
- https://www.luisllamas.es/arduino-detector-gas-mq/
- https://www.prometec.net/sensor-agua/
- http://www.picuino.com/es/uno-doc/index.html
- https://forum.arduino.cc/index.php?board=81.0
- https://programarfacil.com/blog/arduino-blog/rele-y-arduino-mkr1000/
- http://manueldelgadocrespo.blogspot.com.es
- https://store.arduino.cc/arduino-uno-rev3
- https://tkkrlab.nl/wiki/Arduino_KY-034_Automatic_flashing_colorful_LED_module
- https://tkkrlab.nl/wiki/Arduino_KY-006_Small_passive_buzzer_module
- https://aprendiendoarduino.wordpress.com/tag/pulsador/
- https://www.arduino.cc/en/Reference/SoftwareSerial
- http://www.uugear.com/portfolio/bluetooth-communication-between-raspberry-pi-and-arduino/
- https://www.raspberrypi.org/downloads/raspbian/
- http://www.bujarra.com/raspberry-pi-termometro-y-control-web-de-la-casa/
- https://www.atareao.es/tutorial/raspberry-pi-primeros-pasos/lamp-raspberry-pi/
- http://blascarr.com/raspberry-gpio-with-python/
- https://es.wikipedia.org/wiki/Raspberry_Pi
- https://www.raspberrypi.org/documentation/usage/gpio/README.md
- http://pyspanishdoc.sourceforge.net/
- http://untitled.es/ldr-raspberry-pi2-python/
- https://www.atareao.es/tutorial/raspberry-pi-primeros-pasos/lamp-raspberry-pi/
- http://php.net/manual/es/
- https://www.w3schools.com/html/

8-. Herramientas de desarrollo

La herramientas utilizadas para la programación de este proyecto son todas de código abierto.

- Microsoft Visual Studio 2017 Community Edition. (Python)
- Netbeans IDE 8.2. (Php, Html)
- Fritzing Beta Versión 0.9.3. (Creación de Esquema de montajes arduino)
- Mysql Server 5.1. (Bases de datos)
- Mysql WorkBench 6.3 CE. (Administración de Bases de datos mysql)
- KiCad 4.01. (Creación Placa Electrónicas)
- ShellScriptLinux.(Script Linux)
- LlibreOffice 6. (Creación de documentación)
- Arduino IDE. (Creación de programas para arduino en C)
- Servidor LAMP.(Servidor Web y php)
- PyCharm Community 2018.1.1 (Python)