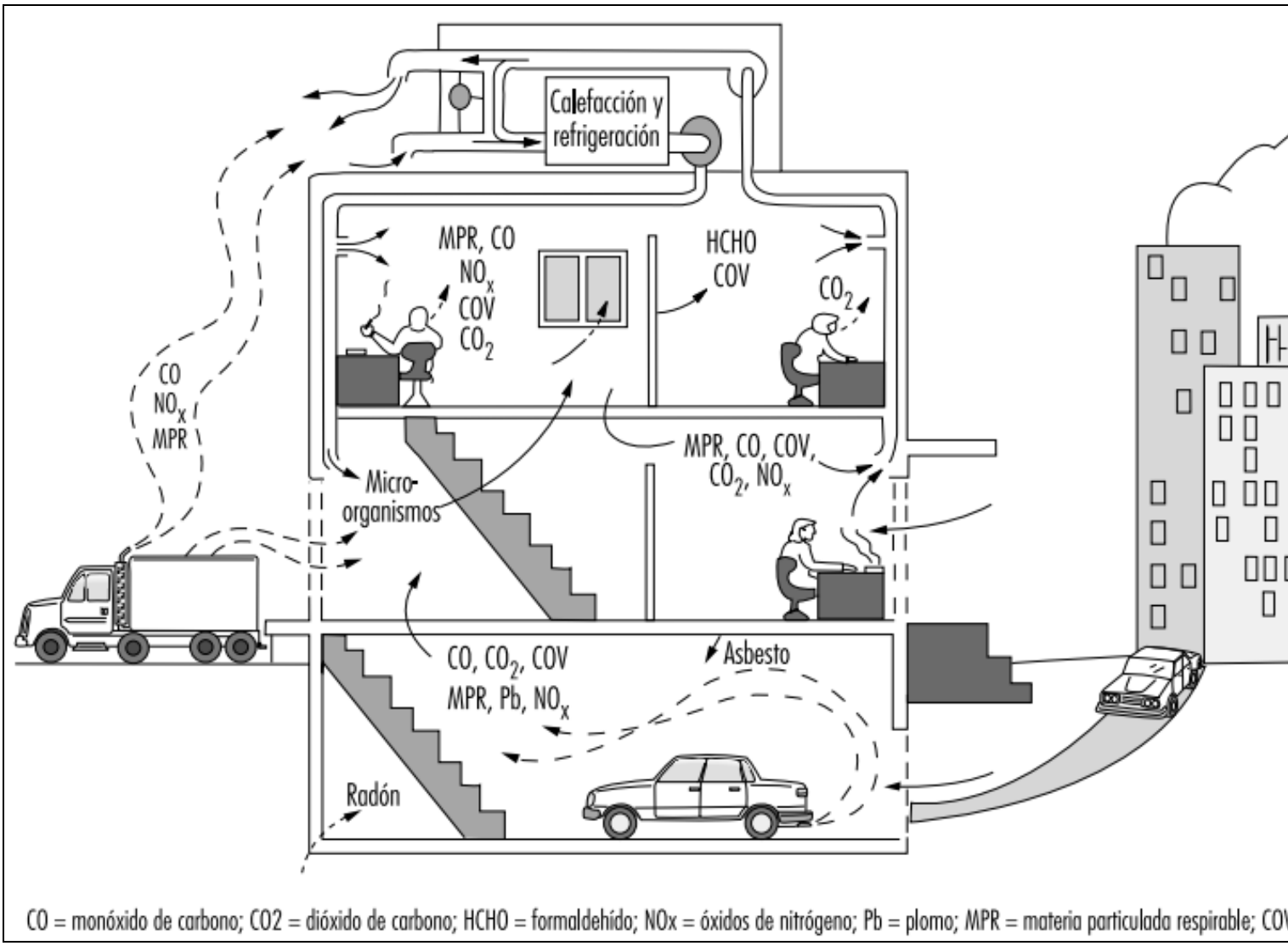
Python - Raspberry Pi

Sensoralia

- Crearemos algo parecido... pero web - Este casi 500?
- Calidad del aire interior



- Air pollution
- Árboles inteligentes:
    - ◊ Artículo.
    - ◊ Web.
- Para el diseño de los circuitos instalar el software: fritzing
- Comandos de utilidad GPIO

Sumario

Descargar e instalar Raspbian

- Descargamos e instalamos la versión RASPBIAN STRETCH LITE. Para la instalación desde Windows utilizamos el software **Etcher**.
- Cuando se encuentre todo configurado crearemos una imagen, para clonar en el resto de los sensores, utilizando el siguiente manual.

Configuración automática sistema

Configuración básica

Configurar los siguientes parámetros:

- Usuarios + password :

El password por defecto de pi es "raspberry" Tanto al usuario **pi** como a **root** les configuramos el password **abc123..**

```
#Le cambiamos el password a pi
$ passwd
#Nos pasamos a root
$ sudo su
#Le cambiamos el password a root
$ passwd

* Actualizamos firmware de la Raspberry
<source lang="bash">
$ rpi-update
```

1. Configuramos la hora

dpkg-reconfigure tzdata

1. Configuramos los locales y dejamos solamente marcado es_ES.utf8

dpkg-reconfigure locales

</source>

- Actualizamos el equipo

```
$ apt update
$ apt upgrade
$ reboot
```

- Cambiamos el teclado a español:

```
$ dpkg-reconfigure keyboard-configuration
```

- Activamos acceso por SSH:

```
$ raspi-config
# 5 Interfacing Options
## P2 SSH
```

- Configuraciones varias:

```
#Cambiar permisos de iwlist para que www-data pueda escanear wifis
$ chmod 4755 /sbin/iwlist
#Cambiar permisos de i2cdetect para www-data pueda escanear i2c
$ chmod 4755 /usr/sbin/i2cdetect
```

- Hacemos copias de seguridad de algunos archivos que luego se van a cambiar:

```
$ cp /etc/hosts /etc/hosts.bak
$ cp /etc/hostname /etc/hostname.bak
$ cp /etc/network/interfaces /etc/network/interfaces.bak
$ cp /etc/wpa_supplicant/wpa_supplicant.conf /etc/wpa_supplicant/wpa_supplicant.conf.bak
```

- Instalamos pip y pip3 para la instalación posterior de módulos Python:

```
$ apt install python-pip python3-pip
# Instalamos mkpasswd para cifrar contraseñas
$ apt install whois
```

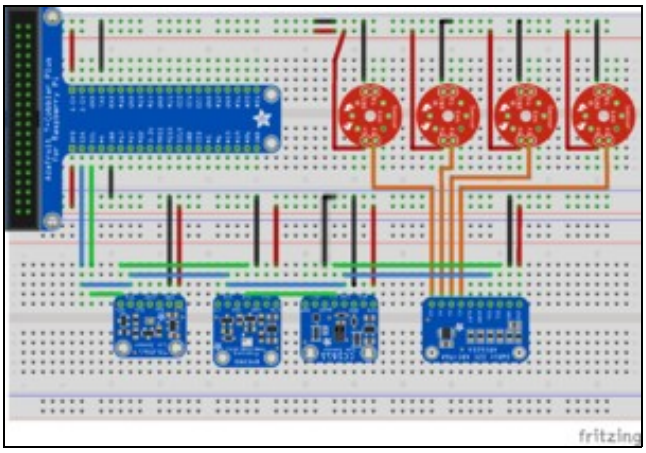
- Instalación de módulos Python que vamos a necesitar:

```
#netifaces para descubrir configuración de red
$pip3 install netifaces
##
#MySQLdb para trabajar con las bases de datos desde Python
$ apt install python3-mysqldb

#Para realizar envíos al servidor por PHP
$pip3 install requests
```

- Instalación de apache2 php7 mariadb y phpmyadmin:

```
# Instalar como root

apt update
apt upgrade

apt install apache2 mariadb-server php7.0 libapache2-mod-php7.0 php7.0-mysql php7.0-gd php7.0-opcache phpmyadmin

mysql_secure_installation
```

Configuración de los sensores



Montaje Sensoralia en Protoboard

- Activamos el i2c:

```
$apt update
$apt upgrade
$apt install python-smbus -y
$apt install i2c-tools -y
$apt install build-essential python-dev git -y

$raspi-config
#5
##P5 i2C

$reboot
```

Creamos una función llamada **i2cocupados()** que guardamos en la librería **funciones3.py** que nos permite leer los i2c ocupados en ese momento:

```
def i2cocupados():

    from subprocess import call
    import re

    call ('/usr/sbin/i2cdetect -y 1 > ./i2cocupados.txt', shell=True)

    salida = list()

    patron_bus = re.compile('\s[0-9a-f]{2}\s')

    with open('./i2cocupados.txt', 'r') as fi2c:
```

```
            for linea in fi2c:
                l = patron_bus.findall(linea.lower())
                if len(l) != 0:
                    for bus in l:
                        salida.append(str(bus.strip()))

        call('/bin/rm ./i2cocupados.txt >> /dev/null', shell=True)

        return salida
```

Sensor CCS811

El sensor CCS811 necesita bajar la velocidad del puerto i2c para un buen funcionamiento:

```
$sudo nano /boot/config.txt
dtparam=i2c_baudrate=10000

$reboot
```

El sensor CCS811 trabaja en la dirección i2c : 5a

Instalamos su librería:

```
$ pip install Adafruit_CCS811
```

Este sensor tiene de problema que no lee bien hasta después de unas 50 lecturas, cada una de estas se pueden hacer cada 2 segundos como muy poco. Al llegar a la lectura 150 este sensor se ?desmadra? y hay que volver a empezar. Tenemos dos opciones:

- Realizar continuamente esta medición guardándola en un archivo que se refresque cada X tiempo. Cuando se pide una medición simplemente se leerá este archivo con el CO2 y los VOCs.
- No dar esta medida hasta pasados poco más de un minuto desde que se pida.

Sensor BME280

El sensor trabaja en la dirección I2C : 77

Instalamos la librería:

```
$ pip install RPi.bme280
```

--- Se lee perfectamenta con la librería de funciones.py ---

Sensor TSL2561

- Comprar unos 6?

El sensor trabaja en la dirección I2C : 39

Instalamos la librería:

```
$ pip install tsl2561
```

--- Se lee perfectamenta con la librería de funciones.py ---

Conversor Analógico - Digital ADS1115

El sensor trabaja en la dirección I2C : 48

Instalamos la librería:

```
$ pip install Adafruit_ADS1x15
```

--- Se lee perfectamenta con la librería de funciones.py pero, en este caso, como hay cuatro posibles entradas, hay que configurar menús para indicar qué sensor tenemos colocado en cada una de las entradas. También debemos de poder configurar las unidades que mide el sensor y el factor de corrección (si este fuese necesario) ---

Configuración del servidor web

```
mv /var/www/html /var/www/public

# Editar el fichero siguiente:
nano /etc/apache2/sites-enabled/000-default.conf

# Modificar este parámetro

        DocumentRoot /var/www/public

        <Directory "/var/www/public">
                Options -Indexes +FollowSymLinks
                AllowOverride None
                Require all Granted
        </Directory>



DocumentRoot /var/www/public

service apache2 restart

chmod 757 /var/www

# Enviar los archivos al /var/www
```

- Nombre equipo : **sensor**
- Dominio : **ies.local**
- NIC Ethernet:
    - ◊ IP : **10.24.220.1**
    - ◊ Máscara de Subred : **255.255.0.0**
    - ◊ Puerta de Enlace : **10.24.0.254**
    - ◊ Servidores DNS : **10.0.4.1 10.0.4.2**
- NIC Wifi:
    - ◊ IP : **10.0.220.2**
    - ◊ Máscara de Subred : **255.255.0.0**
    - ◊ Puerta de Enlace : **10.24.0.254**
    - ◊ Servidores DNS : **10.0.4.1 10.0.4.2**

manual cambio IP

/etc/dhcpcd.conf

Realizamos una copia de seguridad del archivo:

```
$ cp /etc/dhcpcd.conf /etc/dhcpcd.conf.bak
```

----No modificamos este archivo----

/etc/network/interfaces

Realizamos una copia de seguridad del archivo:

```
$ cp /etc/network/interfaces /etc/network/interfaces.bak
```

Luego modificamos el archivo **/etc/network/interfaces**:

```
# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

# Interfaz eth0
auto eth0
#allow-hotplug eth0
iface eth0 inet static
    address 10.24.220.1
    netmask 255.255.0.0
    gateway 10.24.0.254
    dns-nameservers 10.0.4.1 10.0.4.2

# Interfaz wlan0
auto wlan0
#allow-hotplug wlan0
iface wlan0 inet dhcp
#iface wlan0 inet static
    pre-up wpa_supplicant -B Dwext -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.conf
    post-down killall -q wpa_supplicant
#    address 10.231.0.231
#    netmask 255.255.0.0
#    gateway 10.231.0.254

#    dns-nameservers 8.8.8.8
```

/etc/wpa_supplicant/wpa_supplicant.conf

Hacemos copia de seguridad del archivo **/etc/wpa_supplicant/wpa_supplicant.conf**:

```
$ cp /etc/wpa_supplicant/wpa_supplicant.conf /etc/wpa_supplicant/wpa_supplicant.conf.bak
```

Luego modificamos el archivo **/etc/wpa_supplicant/wpa_supplicant.conf** añadiendo un bloque como este para cada Wifi:

```
country=ES
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="SCT_Profes"
    proto=RSN
    key_mgmt=WPA-EAP
    pairwise=CCMP TKIP
    group=CCMP TKIP
    identity="USUARIO"
    password="AQUIvaLAclave"
    phase1="peaplabel=0"
    phase2="auth=MSCHAPV2"
}
```

/etc/hosts

```
$ cp /etc/hosts /etc/hosts.bak
$ nano /etc/hosts
127.0.0.1       localhost.localdomain localhost
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters

127.0.1.1       sensor1.sanclemente.local sensor1
```

/etc/hostname

```
$ mv /etc/hostname /etc/hostname.bak
$ nano /etc/hostname
sensor1
```

Configuración hostAP

```
https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md

Para que muestre hasta el canal 13
apt-get install crda

Editamos el fichero /etc/default/crda y añadimos la línea:
REGDOMAIN=ES
```

Scripts Python

datosbase.conf

```
nome      : Nome do sensor : s : sensor
dominio   : Dominio : s : IES.local
eth0dhcp  : IP por DHCP en la conexión de cable [S/N] : b : False
eth0ip    : IP de la conexión de cable : s : 192.168.0.220
eth0ms    : Máscara de subred de la conexión de cable : s : 24
eth0pe    : Puerta de enlace de la conexión de cable : s : 192.168.0.1
eth0dns   : Servidores DNS de la conexión de cable : s : 8.8.8.8 8.8.4.4
wlan0dhcp : IP por DHCP en la conexión Wifi [S/N] : b : True
wlan0ip    : IP de la conexión Wifi : s :
wlan0ms    : Máscara de subred de la conexión Wifi : s :
wlan0pe    : Puerta de enlace de la conexión Wifi : s :
wlan0dns   : Servidores DNS de la conexión Wifi : s :
```

datos.conf

```
nome      : sensor
dominio   : ies.local
eth0dhcp  : False
eth0ip    : 192.168.0.220
eth0ms    : 24
eth0pe    : 192.168.0.1
eth0dns   : 8.8.8.8 8.8.4.4
wlan0dhcp : True
wlan0ip    :
wlan0ms    :
wlan0pe    :
wlan0dns   :
```

funcions.py

```
### Función que cifra el password si se le pasa como parámetro
#### si no se le pasa ninguno como parámetro te pide uno por pantalla
#### devuelve el password cifrado
def cifrarpassword(p=''):
    from getpass import getpass
    from subprocess import PIPE, Popen

    # Que la contraseña no tenga espacios
    p = str(p.replace(" ",""))

    # Si la contraseña queda "Vacía"
    if p == "":
        try:
            # Nos pide una contraseña que no aparece por pantalla
            p = getpass(prompt='Contrasinal: ')

        except Exception as erro:
            print('ERRO: ', erro)

        else:
```

```
            # Ciframos la contraseña
            cep = 'mkpasswd {}'.format(p)
            objep = Popen (cep, stdout=PIPE, stderr=PIPE, shell=True)
            erro = objep.stderr.read()
        # La salida de Pope es en binario, hay que pasarla a texto
        ##y quitarle el salto de linea final
            ep = ((objep.stdout.read()).decode('utf-8')).replace('\n','')
            objep.stdout.close()
            if not erro:
                return ep
            else:
                print ('ERRO: ', erro)
#########################################################################

## Función que cambia el password del usuario user
### llama a la función cifrarpassword(p) que cifrará el password "p" pasado
### si no se le pasa ningún password, la función pedirá una por pantalla
def changepassword(user,p=''):
    from subprocess import call

    ep = cifrarpassword()
    # Comando que cambiará el password de user
    ccp = 'echo "{}:{}" | chpasswd -e'.format(user,ep)
    # Ejecución del comando que cambiará el password
    call(ccp, shell=True)
#########################################################################

## Function pedirDatosNomIP
### Le pasamos de parámetro la dirección del archivo datosbase.conf
#### devuelve un diccionario con los datos pedidos por teclado
def pedirDatosNomIP(pathdatosbase):
    dicdatos = dict() #Creamos diccionario para los datos
    with open(pathdatosbase, "r") as archdatosbase: #Leemos los datos a guardar del archivo datosbase.conf
        for line in archdatosbase:
            nome_valor = str(line.split(':')[0].strip()).replace(" ","") #Nombre del valor
            pregunta = '{} [{}]: '.format(str(line.split(':')[1].strip()),str(line.split(':')[3].strip())) #Pregunta de la explicació

            if 'eth0dhcp' in dicdatos: #Si eth0 por DHCP ya no se piden los datos para eth0
                if dicdatos.get('eth0dhcp') == 'True' and str(line.split(':')[0].strip()).replace(" ","").startswith('eth0'):
                    continue

            if 'wlan0dhcp' in dicdatos: #Si wlan0 por DHCP ya no se piden los datos para wlan0
                if dicdatos.get('wlan0dhcp') == 'True' and str(line.split(':')[0].strip()).replace(" ","").startswith('wlan0'):
                    continue

            valor  = str(input(pregunta)).replace(" ","") #Se lee el valor del dato por teclado

            if valor == "" and  str(line.split(':')[2].strip()).replace(" ","") == 's': #Si el dato es un string y no se introduce na
                valor = str(line.split(':')[3].strip()).replace(" ","") ## se rellena con el existente en el archivo datosbase.conf

            if str(line.split(':')[2].strip()).replace(" ","") == 'b': #Si el dato es un booleano, se guarda True o False ...
                if valor == 's' or valor == 'S':
                    valor = 'True'
                elif valor == 'n' or valor == 'N':
                    valor = 'False'
                else:
                    valor = str(line.split(':')[3].strip()).replace(" ","")

            dicdatos.setdefault(nome_valor,valor) #Todos estos valores se van guardando en un diccionario

    #Se devuelve un diccionario con los datos introducidos por el usuario
    return dicdatos
#########################################################################################################################

## Function guardarDatosNomIP
###función que guarda el diccionario de datos en el archivo datos.conf
def guardarDatosNomIP(pathdatos,dicdatos):
    #Recorremos el diccionario
    ##y vamos creando el texto que será guardado en el archivo pathdatos
    datos = ''
    for k,v in dicdatos.items():
        datos = '{}{} : {}\n'.format(datos,k,v)
```

```
    with open(pathdatos, "w") as archdatos:
        archdatos.write(datos)
#########################################################################################################################################
```

Sensores

Más sensores

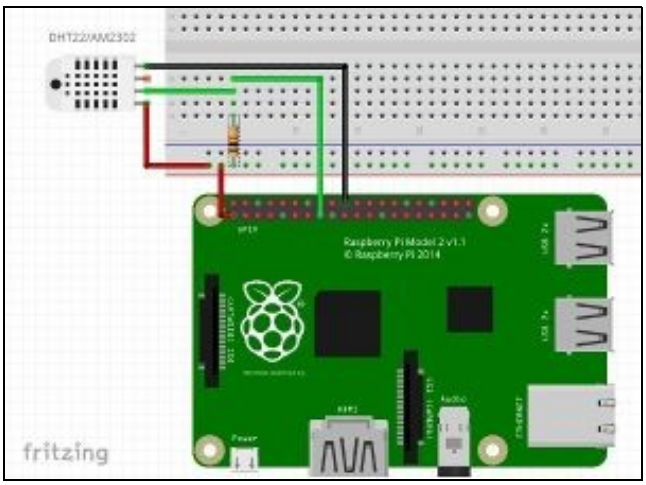
- Radon

    ◊ Información INSHT para pasar de ?Sv a Bq/m3

Sensor de Temperatura y Humedad DHT AM2302

Intentar cambiar este sensor por uno con interface I2C para simplificar la comunicación: AM2311, AM2312, AM2313, AM2315

Lectura de la temperatura y la humedad con el sensor DHT AM2302.
La librería realizada debe estar programada en Python 2.x
El montaje es como el siguiente:



Montaje DHT AM2302

```
#Función que lee la Temperatura con el sensor DHT AM2302
def DHTAM2302_T (pin):

    import sys
    import Adafruit_DHT

    #Parametros
    sensor = Adafruit_DHT.AM2302
    #pin = 4
    pin = int(pin)
    # Esta funcion lee la temperatura y la humedad cada 2 segundos
    humedad, temperatura = Adafruit_DHT.read_retry(sensor, pin)

    # Devolvemos la temperatura si esta se consigue leer
    if humedad is not None and temperatura is not None:
        return temperatura
#         return humedad
    else:
        return -273

#Función que lee la Humedad con el sensor DHT AM2302
def DHTAM2302_H (pin):

    import sys
    import Adafruit_DHT
```

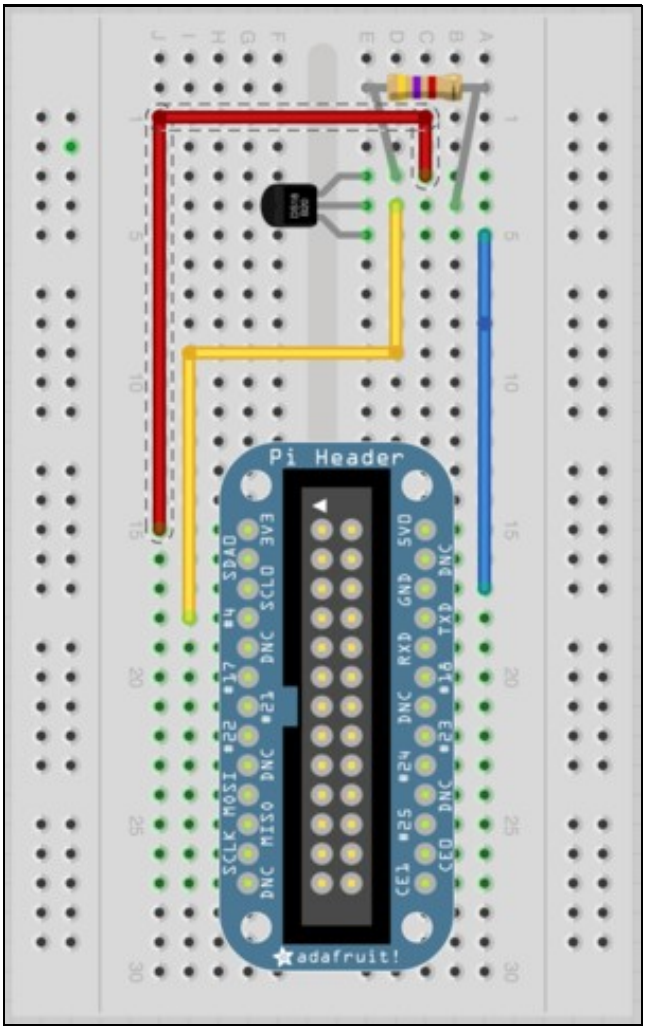
```
    #Parametros
    sensor = Adafruit_DHT.AM2302
    #pin = 4
    pin = int(pin)
    # Esta funcion lee la temperatura y la humedad cada 2 segundos
    humedad, temperatura = Adafruit_DHT.read_retry(sensor, pin)

    # Devolvemos la temperatura si esta se consigue leer
    if humedad is not None and temperatura is not None:
        return humedad
    else:
        return -273



# Probando que todo va bien
# -*- coding: utf-8 -*

print 'Temperatura = {0:0.1f}*C'.format(DHTAM2302_T (4))
print 'Humedad = {0:0.1f}%'.format(DHTAM2302_H (4))
```

- **Notas:**

    ◊ Con los pintes BCM 4 y 22 me funciona, con otros como el 26 NO va.

Sensor de Temperatura DALLAS 8820

Como sensor de temperatura también tenemos el DALLAS 8820.
La comunicación con la Raspberry la realiza por el bus 1-Wire.
El montaje es como el siguiente:



Montaje DS18S20
Preparación del sistema:

Agregamos al archivo /boot/config.txt el parámetro:

dtoverlay=w1-gpio

Ejecutamos los comandos:

modprobe w1-gpio
modprobe w1-therm

En el directorio /sys/bus/w1/devices

accedemos al directorio 28-xxxxx
y leemos el archivo w1_slave allí existente
de ahí extraemos la temperatura

Esta función devuelve un float con la Temperatura en ºC:

```
def D8820_T():
    directorio = '/sys/bus/w1/devices'
    w1file = 'w1_slave'
    from os import listdir
    for d in listdir(directorio):
        if (d.startswith('28-')):
            pathfile = directorio+"/"+d+"/"+w1file
            with open(pathfile) as f:
                i = 0
                for line in f:
                    i = i + 1
                    if (i == 2):
                        temperatura = float(line.split('=')[1].strip())/1000
                        return temperatura
```

Enlaces interesantes:

- Enlace Adafruit

Sensor Temperatura + Humedad + Presión BME280 por I2C

Este sensor tiene interface I2C y SPI.
Lo conectamos a GPIO por I2C:

◊ SDI - SDA
◊ SCK - SCL
◊ GND - GND
◊ VIN - 3V3

Seguimos el [1]

Instalamos la librería : pip install RPi.bme280

Adaptamos el script Python:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import smbus2
import bme280

port = 1
address = 0x77
bus = smbus2.SMBus(port)

bme280.load_calibration_params(bus, address)

# the sample method will take a single reading and return a
# compensated_reading object
data = bme280.sample(bus, address)

# the compensated_reading class has the following attributes
```

```
print("Identificador = %s" %data.id)
print("Fecha y Hora = %s" %data.timestamp)
print("Temperatura = %.2f °C" %data.temperature)
print("Presión = %.2f hPa" %data.pressure) # 1hPa == 1mbar también saber que 1 atm == 1013,25 mbar.
print("Humedad = %.2f %%" %data.humidity)
```

Sensores de gas

Página muy interesante

Prácticamente la misma en castellano

Lista sensores CO2

Sensor MQ-9 Urban air quality sensors Ejemplo interesante de MQ + ADS1115 para dispositivo de Bomberos

Sensor de CO2 NDIR

Diferencias CO CO2

Calidad del aire

Problemas con el MCP3008:

- Adafruit
- Configuración del driver - Velocidad del reloj - Intentar bajarla
- Leer voltaje
- Foro donde parece que se cambia esa velociad

Sensor CCS811 : mide la cantidad de VOC y de eCO2, compartiendo los parámetros por I2C.

- Web Adafruit
- [file:///Users/manuel/Downloads/CCS811_DS000459_4-00.pdf Información fabricante chip CCS811]
- Estupendo video

Sensores MQ + ADC ADS115

Librería actual
Dispositivos interesantes

Sensor utilizado: MG-811.

- Calibrado del sensor MG-811
- Otra web de calibrado

El voltaje de salida decrece al aumentar la cantidad de CO2, Vs es inversamente y linealmente proporcional al logaritmo en base 10 de la concentración de CO2.

Haremos las siguientes lecturas:

- Mo : Lectura de la salida del sensor a 400 PPM medidas con uno de confianza (aire lo más limpio posible).
- Mc : Lectura de la salida del sensor a 1000 PPM medidas con uno de confianza.

La ecuación que debemos implementar sería:

log(PPMs) = log(400) + [log(1000) - log(400)] * [Ms - Mo] / [Mc - Mo]
PPMs = 10^{log(400) + [log(1000) - log(400)] * [Ms - Mo] / [Mc - Mo]}

```
#!/usr/bin/python
# -*- coding: utf-8 -*

import time
from funciones import sensor_ads1115()
from math import pow, log10
```

```
#Tenemos dos lecturas ayudados de un Testo 435:
##Una lectura con la mínima cantidad de CO2 que podamos
### y su correspondiente salida del ADS1115
PPM0 = float(1260)
M0 = float(2565)

##Una lectura con la máxima cantidad de CO2 que podamos
### y su correspondiente salida del ADS1115
PPMc = float(11900)
Mc = float(1543)

co2_raw = float(sensor_ads1115()['MG811'])
co2 = pow(10.0, (log10(PPM0) + (log10(PPMc) - log10(PPM0)) * (co2_raw - M0) / (Mc - M0)))

print "Lectura de CO2 : %s" % co2
```

Sensor CO2 MH-Z16 NDIR

- Web del fabricante

Se trata de un sensor I2C de CO2 por infrarrojos. Esto permite que no tengamos que "calentarlo" para medir, como se hace en los orgánicos y que, al ser I2C, las medidas se hagan con facilidad.

Su dirección I2C es "4d".

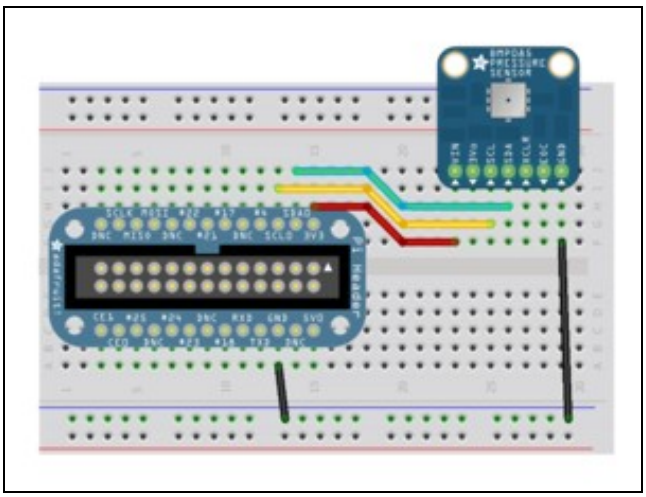
Para leer los datos, utilizaremos un script con base el siguiente código:

```
import smbus2
import time

class Sensor():
    cmd_measure = [0xFF,0x01,0x9C,0x00,0x00,0x00,0x00,0x00,0x63]
    ppm         = 0

    IOCONTROL   = 0X0E << 3
    FCR         = 0X02 << 3
    LCR         = 0X03 << 3
    DLL         = 0x00 << 3
    DLH         = 0X01 << 3
    THR         = 0X00 << 3
    RHR         = 0x00 << 3
    TXLVL       = 0X08 << 3
    RXLVL       = 0X09 << 3

    def __init__(self, i2c_addr):
        self.i2c_addr = i2c_addr
        self.i2c      = smbus2.SMBus(1)

    def begin(self):
        try:
            self.write_register(self.IOCONTROL, 0x08)
        except IOError:
            pass

        trial = 10

        while trial > 0:
            trial = trial - 1

            try:
                self.write_register(self.FCR, 0x07)
                self.write_register(self.LCR, 0x83)
                self.write_register(self.DLL, 0x60)
                self.write_register(self.DLH, 0x00)
                self.write_register(self.LCR, 0x03)
                return True
            except IOError:
                pass

        return False
```

```
    def measure(self):
        try:
            self.write_register(self.FCR, 0x07)
            self.send(self.cmd_measure)
            time.sleep(0.01)
            self.parse(self.receive())
            return True
        except IOError:
            return False

    def parse(self, response):
        checksum = 0

        if len(response) < 9:
            return

        for i in range (0, 9):
            checksum += response[i]

        if response[0] == 0xFF:
            if response[1] == 0x9C:
                if checksum % 256 == 0xFF:
                    self.ppm = (response[2]<<24) + (response[3]<<16) + (response[4]<<8) + response[5]

    def read_register(self, reg_addr):
        time.sleep(0.001)
        return self.i2c.read_byte_data(self.i2c_addr, reg_addr)

    def write_register(self, reg_addr, val):
        time.sleep(0.001)
        self.i2c.write_byte_data(self.i2c_addr, reg_addr, val)

    def send(self, command):
        if self.read_register(self.TXLVL) >= len(command):
            self.i2c.write_i2c_block_data(self.i2c_addr, self.THR, command)

    def receive(self):
        n     = 9
        buf   = []
        start = time.clock()

        while n > 0:
            rx_level = self.read_register(self.RXLVL)

            if rx_level > n:
                rx_level = n

            buf.extend(self.i2c.read_i2c_block_data(self.i2c_addr, self.RHR, rx_level))
            n = n - rx_level

           if time.clock() - start > 0.2:
                break

        return buf


#import NDIR
#import time

sensor = Sensor(0x4D)

if sensor.begin() == False:
    print("Adaptor initialization FAILED!")
    exit()

while True:
    if sensor.measure():
        print("CO2 Concentration: " + str(sensor.ppm) + " ppm")
    else:
       print("Sensor communication ERROR.")

    time.sleep(10)
```

Sensor de luminosidad TSL2561

Conexiones:



Conexión I2C

Configurar el I2C

```
$ sudo apt-get install -y python-smbus
$ sudo apt-get install -y i2c-tools
#Arrancamos raspi-config y habilitamos I2C
$ raspi-config
```

Mejor enlace

```
#Comprobar que se lee el puerto I2C
$ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- 39 -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --

#Instalar con pip la librería
#$ pip install Adafruit_Python_GPIO
$ pip install Adafruit_GPIO
$ pip install tsl2561
#También existe la librería para pip3 pero no consigo que funcione el script
```

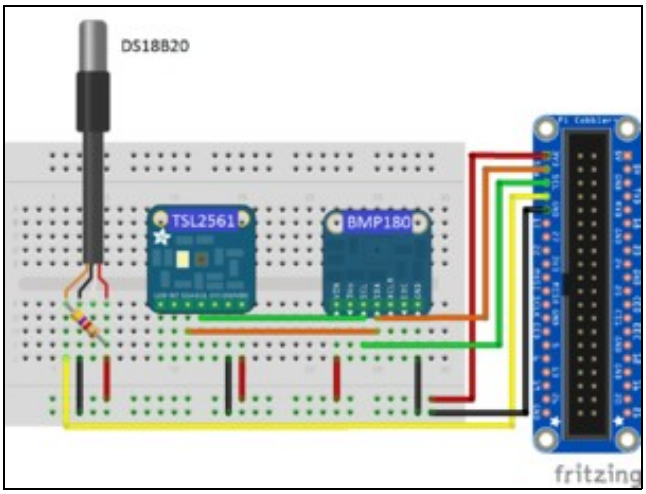
Comprobación de que todo va bien:

```
def TSL2561_L ():
    from tsl2561 import TSL2561
    tsl = TSL2561(debug=1)
    return float(tsl.lux())
```

Enlace

Podemos configurar hasta tres sensores de luminosidad a la vez cambiando la dirección I2C utilizada. Para ello el sensor TSL2561 trae los pines ADDR que, si los dejamos como están utiliza la dirección 0x39, si unimos el del medio con L utiliza la 0x29 y si unimos el del medio con H utiliza la 0x49.

> Luego de hacer la soldadura con L en uno de los sensores conectamos los dos sensores al I2C siguiendo el montaje que podemos ver a continuación:



Varios I2C

Modificamos el archivo python para que pueda leer un sensor en la dirección 0x39 y otro en la dirección 0x29:

```
...
#Sensor de luminosidad TSL2561
def TSL2561_L (canal):
    from tsl2561 import TSL2561
    tsl = TSL2561(address=canal)
    return float(tsl.lux())

# Probando que todo va bien
# -*- coding: utf-8 -*
print 'Lux1 = {0:0.1f} lx'.format(TSL2561_L (0x29))
print 'Lux2 = {0:0.1f} lx'.format(TSL2561_L (0x39))
...
```

Encender LED si luz < 100 lux

Resolvemos aquí el problema de que la librería del sensor TSL2561 sólo va en Python 2.7 y queremos hacer los scripts en Python 3.x. La práctica consiste en comprobar la luminosidad y si esta baja de 100 lux encendemos un LED:

```
# principal.py
#!/usr/bin/python3
# -*- coding: utf-8 -*-

#Función que enciende un LED una cierta cantidad de tiempo
def EncenderLed(LED,tiempo):
    import RPi.GPIO as GPIO
    import time

    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LED, GPIO.OUT)

    GPIO.output(LED,True) # Led encendido
    time.sleep(tiempo) # tiempo encendido
    GPIO.output(LED,False) # Led apagado
    GPIO.cleanup() # Limpieza de los GPIO


from subprocess import call
import time

while True:
    call("python prueba2.py", shell=True)
    time.sleep(5)
    fluz = open('./luz.txt','r')
    luz = float(fluz.read())
    fluz.close()
    if luz < 100.0:
        EncenderLed(17,2)
```

```
    print ("Luz ambiente = %.1f lux" %luz)
    time.sleep(2)
```

El siguiente script es el **prueba2.py** que se ejecuta con Python 2.x y llama al sensor TSL2561:

```
#Sensor de luminosidad TSL2561
def TSL2561_L (canal):
    from tsl2561 import TSL2561
    tsl = TSL2561(address=canal)
    return float(tsl.lux())

# Utilizamos el sensor para ver la luz ambiente
lux = float(TSL2561_L (0x39))

# Escribimos en el archivo luz.txt el valor en lumens
fluz = open("./luz.txt","w")
fluz.write(repr(lux))
fluz.close()
```

Sensor sonido

Para el I2S Microphone:

- Manual a seguir
- Ver los dispositivos de sonido existentes:

```
$ arecord --list-devies
**** List of CAPTURE Hardware Devices ****
card 1: sndrpisimplecar [snd_rpi_simple_card], device 0: simple-card_codec_link snd-soc-dummy-dai-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

- Gravar 5 segundos de sonido wav:

```
$ arecord -D dmic_sv -c2 -r 48000 -f S32_LE -t wav -V mono -v -d 5 file.wav
```

- Función Python 2 que calcula los dBs de un micrófono (en este caso se utilizó un micrófono USB, que parece que va mejor que el I2S). Para ver el **input_device_index** del dispositivo utilizamos el script siguiente:

```
import pyaudio
p = pyaudio.PyAudio()
info = p.get_host_api_info_by_index(0)
numdevices = info.get('deviceCount')
for i in range(0, numdevices):
        if (p.get_device_info_by_host_api_device_index(0, i).get('maxInputChannels')) > 0:
            print "Input Device id ", i, " - ", p.get_device_info_by_host_api_device_index(0, i).get('name')
```

  La función del cálculo de los dBs es la siguiente:

```
import numpy
import pyaudio
import analyse
from time import sleep

# Initialize PyAudio
pyaud = pyaudio.PyAudio()

# Open input stream, 16-bit mono at 44100 Hz
# On my system, device 2 is a USB microphone, your number may differ.
stream = pyaud.open(
    format = pyaudio.paInt16,
    channels = 1,
    rate = 44100,
    input_device_index = 2,
    input = True)

stream.start_stream()
sleep(1)
# Read raw microphone data
```

```
rawsamps = stream.read(10240)
# Convert raw data to NumPy array
samps = numpy.fromstring(rawsamps, dtype=numpy.int16)
# Mostrar volumen
sleep(1)
stream.stop_stream()

db = analyse.loudness(samps)
#print "Volumen = ", db, " dB - tipo de la variable : ", type(db)
fdbs = open("./dbs.txt","w")
fdbs.write(repr(db))
fdbs.close
```

- Con soundmeter debería medir la potencia, pero no está bien configurado el dispositivo de entrada I2S para alsa-utils.

Para trabajar con el .wav desde Python mirar las siguientes librerías:

- PySoundFile
- scipy.io.wavfile (from scipy) - Comandos para scipy.
    - ◊ Instalar scipy
    - ◊ Pasar un archivo wave a un array numpy
    - ◊ Devuelve el volumen de un sonido entre -80dB silencio y 0dB máximo
- wave (to read streams. Included in python 2 and 3)
    - ◊ wave y audioop para calcular max y avg
        - Librería audioop
- scikits.audiolab (that seems unmaintained)
- sounddevice (play and record sounds, good for streams and real-time)
- pyglet
- Calcular la potencia directamente - Problemas con el dispositivo de entrada.

Otros enlaces: web

Otro sensor de sonido

Y otro

- Sensor sonido Sparkfun

    Lo conectamos al conversor analógico digital I2C ADS1115 y leemos la salida ***ENVELOPE***... *Luego tendremos que mirar cómo sacar de ahí una medida de los decibelios.*

Captura de fotos y vídeo

Seguir este tutorial

Comandos raspicam

Información más completa

OpenCV

Detector de caras
Instalar OpenCV

```
import numpy as np
import cv2
import sys

imagePath = sys.argv[1]
cascPath = sys.argv[2]
```

```
faceCascade = cv2.CascadeClassifier(cascPath)

gray = cv2.imread(imagePath,0)

faces = faceCascade.detectMultiScale (
    gray,
    scaleFactor=1.3,
    minNeighbors=5,
    minSize=(30,30),
    flags = cv2.cv.CV_HAAR_SCALE_IMAGE
)

print "Encontradas {0} caras!".format(len(faces))
```

Comando a ejecutar:

```
$ python contarpersonas.py foto.jpg haarcascade_frontalface_default.xml
```

Sensor de vibraciones

Para esta tarea se utiliza un Geophone.

Conversor analógico digital

Enlace

Enlace Amazon 20?

```
# Simple demo of reading each analog input from the ADS1x15 and printing it to
# the screen.
# Author: Tony DiCola
# License: Public Domain
import time

# Import the ADS1x15 module.
import Adafruit_ADS1x15


# Create an ADS1115 ADC (16-bit) instance.
adc = Adafruit_ADS1x15.ADS1115()

# Or create an ADS1015 ADC (12-bit) instance.
#adc = Adafruit_ADS1x15.ADS1015()

# Note you can change the I2C address from its default (0x48), and/or the I2C
# bus by passing in these optional parameters:
#adc = Adafruit_ADS1x15.ADS1015(address=0x49, busnum=1)

# Choose a gain of 1 for reading voltages from 0 to 4.09V.
# Or pick a different gain to change the range of voltages that are read:
#  - 2/3 = +/-6.144V
#  -   1 = +/-4.096V
#  -   2 = +/-2.048V
#  -   4 = +/-1.024V
#  -   8 = +/-0.512V
#  -  16 = +/-0.256V
# See table 3 in the ADS1015/ADS1115 datasheet for more info on gain.
GAIN = 1

print('Reading ADS1x15 values, press Ctrl-C to quit...')
# Print nice channel column headers.
print('| {0:>6} | {1:>6} | {2:>6} | {3:>6} |'.format(*range(4)))
print('-' * 37)
# Main loop.
while True:
    # Read all the ADC channel values in a list.
    values = [0]*4
    for i in range(4):
        # Read the specified ADC channel using the previously set gain value.
```

```
        values[i] = adc.read_adc(i, gain=GAIN)
        # Note you can also pass in an optional data_rate parameter that controls
        # the ADC conversion time (in samples/second). Each chip has a different
        # set of allowed data rate values, see datasheet Table 9 config register
        # DR bit values.
        #values[i] = adc.read_adc(i, gain=GAIN, data_rate=128)
        # Each value will be a 12 or 16 bit signed integer value depending on the
        # ADC (ADS1015 = 12-bit, ADS1115 = 16-bit).
    # Print the ADC values.
    print('| {0:>6} | {1:>6} | {2:>6} | {3:>6} |'.format(*values))
    # Pause for half a second.
    time.sleep(0.5)
```

Script definitivo de captación de datos

Nota : Para que todo funcione correctamente sería interesante alimentar todo con una fuente de alimentación de 10A.

Sensores:

- BME280
- TSL2561
- CCS811
- ADS115:
    - A0 : MQ-2 (Methane, Butane, LPG, smoke)
    - A1 : MQ-9 (Carbon Monoxide, flammable gasses)
    - A2 : MG811 (Carbon Dioxide (CO2))
    - A3 : Sparkfun sound detector

Conectamos todo y miramos las direcciones del puerto I2C:

```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- 39 -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- 5a -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

- **sensoresactivos.conf**

```
temperatura
humedad
#presion
luminosidad
#vocs
#eco2
#sonido
#MQ2
#MQ9
#MG811
```

- **confsensor.conf**

```
#Se van recogiendo medidas de los sensores cada tiempo_entre_medidas (segundos)
tiempo_entre_medidas = 10
#Se recogen así durante tiempo_media y, con todas esas medidas se hará la media
## que es lo que se guarda en la base de datos
tiempo_media = 100
#Se espera tiempo_entre_medias sin guardar ningún tipo de medida
tiempo_entre_medias = 300

DB_HOST = dbserver.sanclemente.local
DB_USER = sensoralia
DB_PASS = sensorpass..
DB_NAME = sensoralia
```

```
DB_TABLA_MEDICIONES = mediciones
```

- **funciones.py**

```
#!/usr/bin/python
# -*- coding: utf-8 -*

#Funciones del sensor

######################################################################
#######   BME280   ###################################
## Temperatura  ##  Humedad  ## Presión atmosférica ##

def sensor_bme280():
    import smbus2
    import bme280

    port = 1
    address = 0x77
    bus = smbus2.SMBus(port)

    bme280.load_calibration_params(bus, address)

    # Devuelve un objeto con todos los parámetros
    data = bme280.sample(bus, address)

    # A nosotros nos van a interesar los atributos:
    ## Temperatura, Humedad y Presión atmosférica
    salida = dict()
    salida.setdefault('temperatura', float(data.temperature)) #Clave : "temperatura"
    salida.setdefault('presion', float(data.pressure))        #Clave : "presion"
    salida.setdefault('humedad', float(data.humidity))        #Clave : "humedad"

    return salida

######################################################################

######################################################################
#######   TSL2561   ###################################
## Luminosidad  ##

def sensor_tsl2561():
    from tsl2561 import TSL2561
    tsl = TSL2561(debug=1)

    salida = dict()
    salida.setdefault('luminosidad', float(tsl.lux()))

    return salida

######################################################################

######################################################################
#######   CCS811   ###################################
## eCO2  ##  VOCs  ##

def sensor_ccs811(): ##Este sensor lo hay que inicializar obligándolo a medir unos 10 - 15 segundos antes de que su medida sea fiable
    from Adafruit_CCS811 import Adafruit_CCS811
    ccs =  Adafruit_CCS811()

    while not ccs.available():
            pass
#    temp = ccs.calculateTemperature()
#    ccs.tempOffset = temp - 25.0

    if ccs.available():
#        temp = ccs.calculateTemperature()
        if not ccs.readData():
            co2 = float(ccs.geteCO2())
            tvoc = float(ccs.getTVOC())
```

```
        else:
            print "ERROR!"
            while(1):
                pass

    salida = dict()
    salida.setdefault('eco2', co2)
    salida.setdefault('vocs', tvoc)

    return salida

######################################################################

######################################################################
#######   ADS1115   ##################################
## A0 - MQ2  ##  A1 - MQ9  ##  A2 - MG811  ##  A3 - Sonido  ##

def sensor_ads1115(): #Devolverá una lista con el valor de las cuatro entradas analógicas
    import Adafruit_ADS1x15

    adc = Adafruit_ADS1x15.ADS1115()

    # Choose a gain of 1 for reading voltages from 0 to 4.09V.
    # Or pick a different gain to change the range of voltages that are read:
    #  - 2/3 = +/-6.144V
    #  -   1 = +/-4.096V
    #  -   2 = +/-2.048V
    #  -   4 = +/-1.024V
    #  -   8 = +/-0.512V
    #  -  16 = +/-0.256V
    # See table 3 in the ADS1015/ADS1115 datasheet for more info on gain.
    GAIN = 1 #Parece que con esta ganancia los valores son buenos?

    sal = [0, 0, 0, 0]
    for i in range(len(sal)):
    # Read the specified ADC channel using the previously set gain value.
        sal[i] = adc.read_adc(i, gain=GAIN)
        # Note you can also pass in an optional data_rate parameter that controls
        # the ADC conversion time (in samples/second). Each chip has a different
        # set of allowed data rate values, see datasheet Table 9 config register
        # DR bit values.
        #salida[i] = adc.read_adc(i, gain=GAIN, data_rate=128)
        # Each value will be a bit signed integer value
    # Devolver la salida

    #Facemos un diccionario para a salida
    salida = dict()
    salida.setdefault('MQ2', sal[0])
    salida.setdefault('MQ9', sal[1])
    salida.setdefault('MG811', sal[2])
    salida.setdefault('sonido', sal[3])

    return salida

######################################################################

#Leemos los datos de los sensores
##Datos de Temperatura, Humedad y Presión atmosférica
bme280_tph = sensor_bme280()
# print(bme280_tph)
##Dato de Luminosidad
tsl2561_l = sensor_tsl2561()
# print(tsl2561_l)
##Datos de eCO2 y VOCs
ccs811_eco2vocs = sensor_ccs811()
# print(ccs811_eco2vocs)
##Datos de Humo, Monóxido, Dióxido y Sonido
ads1115_mqson = sensor_ads1115()
# print(ads1115_mqson)

##Diccionario salida
salida = dict()
salida.update(bme280_tph)
```

```
salida.update(tsl2561_l)
salida.update(ccs811_eco2vocs)
salida.update(ads1115_mqson)
print(salida)

#Lo escribimos en un archivo datosintermedios.txt
with open('/home/pi/datosintermedios.txt', 'a') as f:
    dato_escribir = '{}\n'.format(salida)
    f.write(dato_escribir)

###Un espacio##
print(" ----- ")
```

- **Vemos la salida: general_media.py**

```
#!/usr/bin/python3
# -*- coding: utf-8 -*

import time
import MySQLdb
from subprocess import call

##Función que calcula la media de los datos de una lista
###Descarta el dato de menor valor y el de mayor valor
def media_datos(lista):
    media = 0
    listac = sorted(lista)[1:-1]
    for n in listac:
        media += n
    media /= len(listac)
    return media
##################################


#Borramos el archivo de medidas
call('rm /home/pi/datosintermedios.txt', shell=True)


#Leemos del archivo confsensor.conf los datos
##tiempo_entre_medidas
##tiempo_media
with open('/home/pi/confsensor.conf', 'r') as fconfsensor:
    for datosconf in fconfsensor:
        if datosconf.strip().startswith('tiempo_entre_medidas'):
            tiempo_entre_medidas = int(datosconf.split('=')[1].strip())
        elif datosconf.strip().startswith('tiempo_media'):
            tiempo_media = int(datosconf.split('=')[1].strip())
        else:
            pass
tiempo_media = time.time() + tiempo_media

#Realizamos todas ls medidas intermedias
##de las que haremos la media
while True:
    if time.time() > tiempo_media:
        break
    else:
        call("/usr/bin/python /home/pi/funciones.py", shell=True)
        time.sleep(tiempo_entre_medidas)

##Objeto Sensor
class Sensor:
    def __init__(self, nombre):
        self.nombre = nombre
        self.valores = list()

    def agregar_valor(self, valor):
        self.valores.append(valor)
####

#Leemos el archivo de sensores que nos interesan y creamos un objeto de cada uno
##Para luego poder un introduciendo los valores medidos intermedios en una lista
###y poder hacer la media
```

```
lista_sensores = list()
with open('/home/pi/sensoresactivos.conf', 'r') as fsensores:
   for nsensor in fsensores:
       if not nsensor.strip().startswith('#') and not nsensor.strip() == '':
           lista_sensores.append(Sensor(nsensor.strip()))

for s in lista_sensores:
    print(s.nombre)



with open('/home/pi/datosintermedios.txt', newline='') as f:
    for linea in f:
        for campo in linea.split(','):
            for s in lista_sensores:
                if (campo.strip().split(':')[0].strip("{}'") == s.nombre):
                            s.agregar_valor(float(campo.split(':')[1].strip('{}\n')))


#Miramos fecha y hora actuales
fecha = time.strftime("%Y-%m-%d")
hora = time.strftime("%H:%M:%S")

texto_web = ""
texto_web += "fecha : {} <br/>".format(fecha)
texto_web += "hora : {} <br/>".format(hora)
for s in lista_sensores:
    texto_web += "{} : ".format(s.nombre)
#    print(s.valores)
    texto_web += '{} <br/>'.format(round(media_datos(s.valores), 2))
#    print('----')

print(texto_web)

with open('/var/www/html/sensor.html', 'w') as f:
    f.write(texto_web)


#Introducimos datos en la BD
#DB_HOST = 'localhost'
#DB_USER = 'dbsensor'
#DB_PASS = 'abc123..'
#DB_NAME = 'dbsensor'
#DB_TABLA_MEDICIONES = 'mediciones'
#Leemos los datos de conexión del archivo confsensor.conf
with open('/home/pi/confsensor.conf', 'r') as fconfsensor:
    for datosconf in fconfsensor:
        if datosconf.strip().startswith('DB_HOST'):
            DB_HOST = str(datosconf.split('=')[1].strip())
        elif datosconf.strip().startswith('DB_USER'):
            DB_USER = str(datosconf.split('=')[1].strip())
        elif datosconf.strip().startswith('DB_PASS'):
            DB_PASS = str(datosconf.split('=')[1].strip())
        elif datosconf.strip().startswith('DB_NAME'):
            DB_NAME = str(datosconf.split('=')[1].strip())
        elif datosconf.strip().startswith('DB_TABLA_MEDICIONES'):
            DB_TABLA_MEDICIONES = str(datosconf.split('=')[1].strip())
        else:
            pass

def run_query(query=''):
    datosdb = [DB_HOST, DB_USER, DB_PASS, DB_NAME]
    conn = MySQLdb.connect(*datosdb) # Conectar a la base de datos
    cursor = conn.cursor()         # Crear un cursor
    cursor.execute(query)          # Ejecutar una consulta

    if query.upper().startswith('SELECT'):
        data = cursor.fetchall()   # Traer los resultados de un select
    else:
        conn.commit()              # Hacer efectiva la escritura de datos
        data = None

    cursor.close()                 # Cerrar el cursor
```

```
    conn.close()                    # Cerrar la conexion

    return data

#Generamos la consulta utilizando los valores de los sensores
nombre_estacion = "sensorUNO"
query = "INSERT INTO {} (idestacion, idsensor, valor) VALUES ".format(DB_TABLA_MEDICIONES)

for s in lista_sensores:
    query += "('{}', '{}', '{}'),".format(nombre_estacion, s.nombre, round(media_datos(s.valores), 2))

query = query.strip(',')
print(query)

run_query(query)
```

- **Vemos la salida simple: general.py**

```
#!/usr/bin/python3
# -*- coding: utf-8 -*

from subprocess import call
import time

while True:
    call("python funciones.py", shell=True)
    time.sleep(10)
```

- **sinparar.py** --- Que no pare de medir

```
#!/usr/bin/python3
# -*- coding: utf-8 -*

from subprocess import call
import time

tiempo_entre_medias = 60

#Realizamos medidas sin parar cada tiempo_entre_medias
while True:
    call("python3 general_media.py", shell=True)
    time.sleep(tiempo_entre_medias)
```

Crear servicio

```
# Ver listado de los servicios activos
$ systemctl list-unit-files | grep enabled

# Ir al directorio system
$ cd /etc/systemd/system

# Crear archivo servicio
$ nano sensor.service
#Contenido del archivo:
[Unit]
Description=Daemon Sensor sin parar
After=multi-user.target

[Service]
Type=idle
User=root
ExecStart=/usr/bin/python3 /home/pi/sinparar.py
Restart=always
TimeoutStartSec=10
RestartSec= 10

[Install]
WantedBy=multi-user.target

# Habilitar servicio
$ systemctl enable sensor.service
```

```
# Arrancarlo en cualquier momento
$ systemctl start sensor.service

# Pararlo en cualquier momento
$ systemctl stop sensor.service

# Ver su estado
$ systemctl status sensor.service
```