

4.2 Haciendo uso del pinout

Llegó el momento de *mancharnos las manos*

Todos los montajes **deben hacerse** con la Raspberry Pi apagada.

Cuestiones previas

Antes de comenzar a trabajar con la raspberry Pi necesitamos tener claras algunas cuestiones como el pinout del modelo con el que estemos trabajando. Así, el pinout del modelo 4 es el siguiente:

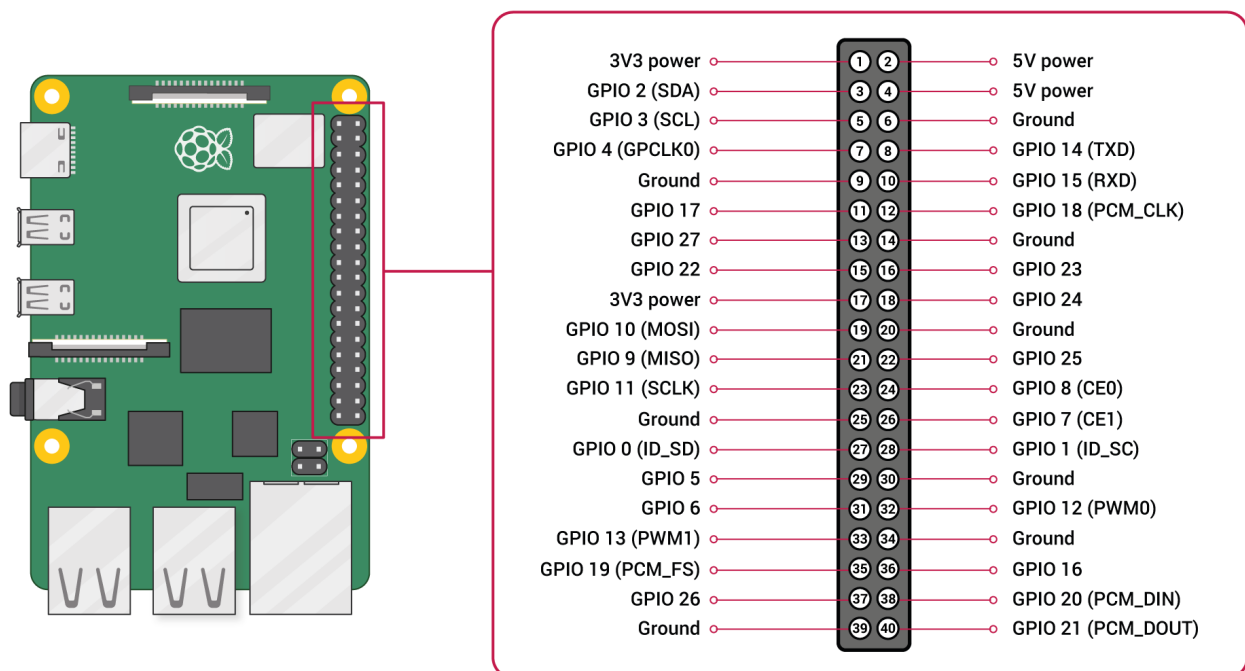


Imagen obtenida de <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>

Una cosa que han hecho muy bien desde Raspberry Pi es mantener la retrocompatibilidad entre modelos así el pinout del modelo 2 B es el mismo que es de este modelo.

Para las prácticas que realizaré a continuación utilizaré una Raspberry Pi modelo 2 B. Las mismas deberían ser totalmente compatibles con modelos actuales.

Vamos a hacer uso de resistencias para algunos montajes por ello te dejo a mano el código de colores de las resistencias:

<p>colores.jpg - Fotos</p> <p>0 1 2 3 4 5 6 7 8 9</p> <p>0 Negro</p> <p>1 Marrón</p> <p>2 Rojo</p> <p>3 Naranja</p> <p>4 Amarillo</p> <p>5 Verde</p> <p>6 Azul</p> <p>7 Púrpura</p> <p>8 Gris</p> <p>9 Blanco</p> <p>±1% Marrón</p> <p>±2% Rojo</p> <p>±5% Dorado</p> <p>±10% Plateado</p>	<p>±1%</p> <p>±2%</p> <p>±5%</p> <p>±10%</p> <p>1.5K</p> <p>0 x1</p> <p>1 1 x10</p> <p>2 2 x100</p> <p>3 3 x1000</p> <p>4 4 x10000</p> <p>5 5 x100000</p> <p>6 6 x1000000</p> <p>7 7 ÷10</p> <p>8 8 ÷100</p> <p>9 9</p>	<p>±1%</p> <p>±2%</p> <p>±5%</p> <p>±10%</p> <p>15K</p> <p>0 0 x1</p> <p>1 1 1 x10</p> <p>2 2 2 x100</p> <p>3 3 3 x1000</p> <p>4 4 4 x10000</p> <p>5 5 5 ÷10</p> <p>6 6 6 ÷100</p> <p>7 7 7</p> <p>8 8 8</p> <p>9 9 9</p>	<p>±1%</p> <p>±2%</p> <p>±5%</p> <p>±10%</p> <p>100</p> <p>25</p> <p>10</p> <p>1</p> <p>620</p> <p>0 0 x1</p> <p>1 1 1 x10</p> <p>2 2 2 x100</p> <p>3 3 3 x1000</p> <p>4 4 4 x10000</p> <p>5 5 5 ÷10</p> <p>6 6 6 ÷100</p> <p>7 7 7</p> <p>8 8 8</p> <p>9 9 9</p>
Código de Colores	Resistencias de 4 Bandas	Resistencias de 5 Bandas	Resistencias de 6 Bandas

Imagen obtenida de <https://i.ytimg.com/vi/ox8Su0lyFXo/maxresdefault.jpg>

También necesitaremos conocer los diferentes segmentos de un display

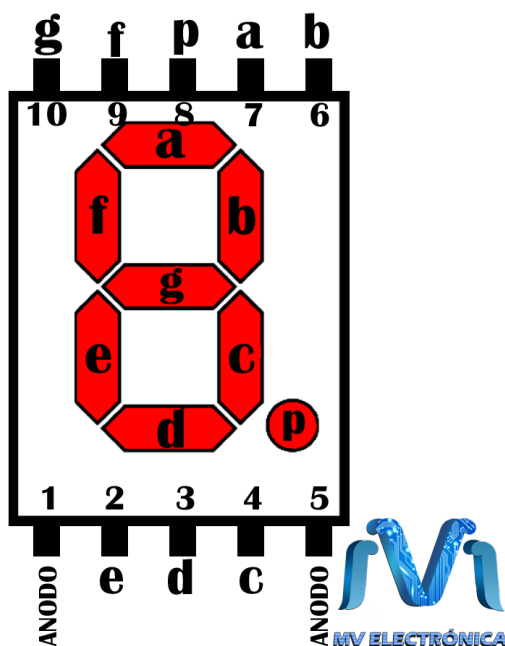


Imagen obtenida de

<https://mvelectronica.com/storage/app/OPTOELECTRONICA/DISPLAY/D7S3A/2.png>

Lo último que tenemos que tener claro es el patillaje de un diodo LED

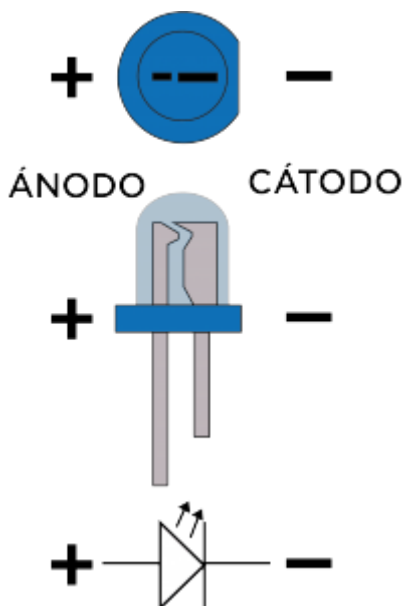


Imagen obtenida de <https://ebotics.com/es/actividad/proyecto-no-2-controlar-el-brillo-del-led/>

También recuerda:

- Ley de Ohm: $V = R \cdot I$ (Tensión es igual a Resistencia por Intensidad) en sus respectivas unidades:
 - Tensión se mide en Voltios (V)
 - Resistencia se mide en Ohmios (Ω)
 - Intensidad se mide en amperios (A)
- Un diodo sólo deja pasar la corriente en un determinado sentido por ello verifica que si no se enciende está conectado en la posición correcta (si no luce también puede ser que esté quemado)

Ejemplo de cálculo

Si queremos actuar sobre algún dispositivo lo haremos a través de los pines que podemos encender y apagar a través del software. Si recurrimos a la [documentación de Raspberry Pi](#) indican "A GPIO pin designated as an output pin can be set to high (3.3V) or low (0V)." Es decir, cuando establezcamos que un pin está encendido (a HIGH, a 1, son todo sinónimos) significa que estaremos estableciendo que en ese pin hay 3,3V mientras que si establecemos que un pin está apagado (a LOW, a 0, son todo sinónimos) significa que estamos estableciendo que en ese pin hay 0V. Además la corriente máxima con la que pueden funcionar un GPIO en la Raspberry es 16mA.

Así, si queremos utilizar un diodo LED rojo que funciona con 1,7 V y 20mA tendremos un exceso de 1,6V (los 3,3V que genera la salida de la Raspberry menos los 1,7V que requiere el LED rojo) ese exceso de tensión tendremos que *dárselos* a otro componente para que no se queme el LED por ello en nuestro diseño meteremos una resistencia, que no tendrá mas objetivo que ser ella la que consuma ese exceso de 1,6V. ¿Y de cuánto tendrá que ser esa resistencia? Sabemos que en esa resistencia tiene que haber 1,6V y que por ella van a circular 16mA = 0,016A (lo ideal sería 20mA pero la Raspberry nos lo limita a 16mA), usamos la Ley de Ohm $R = V/I = 1,6/0,016 = 100 \Omega$. Por lo que junto a nuestro led rojo deberemos usar una resistencia de 100 Ω que coincide con un valor standard comercial. Si el valor calculado no coincide con un valor comercial, hay que elegir el inmediato superior (el inferior proporcionaría más corriente y tenemos la limitación de los 16mA de la Raspberry que no queremos quemarlo).

La potencia que "sufrirá" esa resistencia será $P = VI = 1.6V \times 16mA = 27mW$ por lo tanto cualquier resistencia comercial es válida pues lo mínimo es 1/8W la más común son las 1/4W.

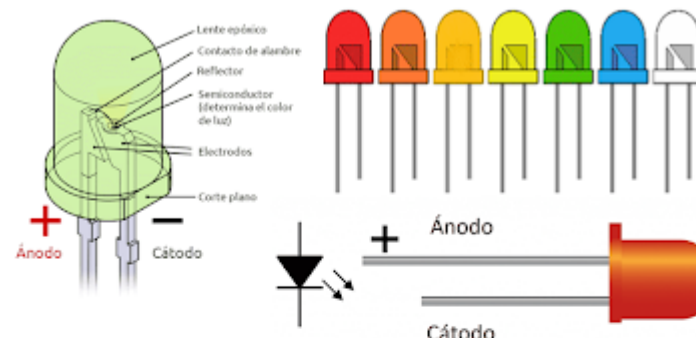
De la misma forma podemos hacer los cálculos para otros colores de Leds, ([aquí para descargar la hoja de cálculo](#)) pero como podemos ver, algunos colores no funcionarían con la Raspberry pues necesitan más tensión de la que proporciona la Raspberry

Color del diodo	Vd	Vcc de la raspberry	mA fabricante	mA Raspberry	$R = (V_{cc} - V_d) / I$ en Ohm	Valor comercial próximo que lo supere
Rojo	1,7	3,3	20	16	100	100
Rojo alta eficiencia	1,9	3,3	20	16	87,5	100
Anaranjado y amarillo	2	3,3	20	16	81,25	82
Verde	2,1	3,3	20	16	75	82
Blanco, verde brillante y azul	3,4	no funciona	12			
Azul brillante y especiales	4,6	no funciona	10			
Display	2	3,3	20	16	81,25	82

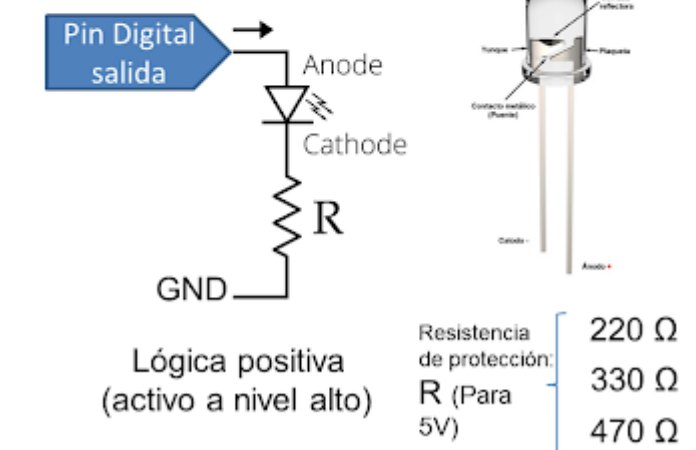
Fuente de datos Wikipedia https://es.wikipedia.org/wiki/Circuito_de_LED

Normalmente en placas como Arduino trabaja a 5V por lo que las resistencias son mayores :

DIODO LED



todotecnologia-eso.blogspot.com



Lógica positiva
(activo a nivel alto)

Resistencia de protección:
R (Para 5V)

220 Ω
330 Ω
470 Ω


Características diodo LED											
Tipo Receptor	Receptor luminoso Diodo emisor de luz (Light Emitting Diode)										
Tipo actuador Arduino	Salida Digital										
Diámetro:	3 – 5- 10 mm										
Símbolo											
Polarizado	Sí										
Intensidad:	10 mA ~ 20 mA										
Tensión de trabajo típica: (Caída de tensión en conducción-polarización directa)	<table border="1"> <tr> <td>Rojo</td> <td>1,7-1,9V / 20 mA</td> </tr> <tr> <td>Ámbar</td> <td>2 V / 20 mA</td> </tr> <tr> <td>Verde</td> <td>2,1 V / 20 mA</td> </tr> <tr> <td>Blanco</td> <td>3,4 V / 12 mA</td> </tr> <tr> <td>Azul</td> <td>4,6 V / 10 mA</td> </tr> </table>	Rojo	1,7-1,9V / 20 mA	Ámbar	2 V / 20 mA	Verde	2,1 V / 20 mA	Blanco	3,4 V / 12 mA	Azul	4,6 V / 10 mA
Rojo	1,7-1,9V / 20 mA										
Ámbar	2 V / 20 mA										
Verde	2,1 V / 20 mA										
Blanco	3,4 V / 12 mA										
Azul	4,6 V / 10 mA										
Potencia	34 mW ~ 46 mW										
Tensión de ruptura (inversa máx.)	2 V ~ 5 V										
Vida útil	30.000 ~ 100.000 h										

Imagen de <https://todotecnologia-eso.blogspot.com/> Licencia CC-BY-SA-NC

Encender y apagar un led

Materiales

- 1 led
- 1 resistencia de 100Ω / 1/4W
- 2 cables
- placa de conexiones

Conexiones

1. Resistencia al cátodo del led
2. ánodo del led a pin 7 (pin, no GPIO)
3. Resistencia a pin 25 (tierra)

Programa

Nos dirigiremos a la terminal y allí escribiremos

```
nano led.py
```

con ello se abrirá el editor nano y estaremos trabajando sobre el fichero led.py que si no existe se creará y si existe trabajaremos sobre el existente. A continuación escribiremos el código python que aparece a continuación:

```
# Importamos la biblioteca que nos permite pausar la ejecución del programa
import time

# Importamos la biblioteca que nos permite conectar a los GPIO pins
# otros dispositivos. GPIO -> General Purpose Input Output
import RPi.GPIO as GPIO

# A partir de aquí configuramos la placa
# 1ero: borramos cualquier configuración previa
GPIO.cleanup()

# 2ndo: establecemos el modo de trabajo de la raspi
```

```
GPIO.setmode(GPIO.BOARD)

# 3ero: Indicamos que el pin 7 actuará como salida
GPIO.setup(7,GPIO.OUT)

# Bucle infinito
while True:

    # Escribo en el terminal el texto enciendo
    print("enciendo")

    # Pongo tensión en el pin 7
    GPIO.output(7,GPIO.HIGH)

    # Detengo la ejecución del programa 2 segundos
    time.sleep(2)

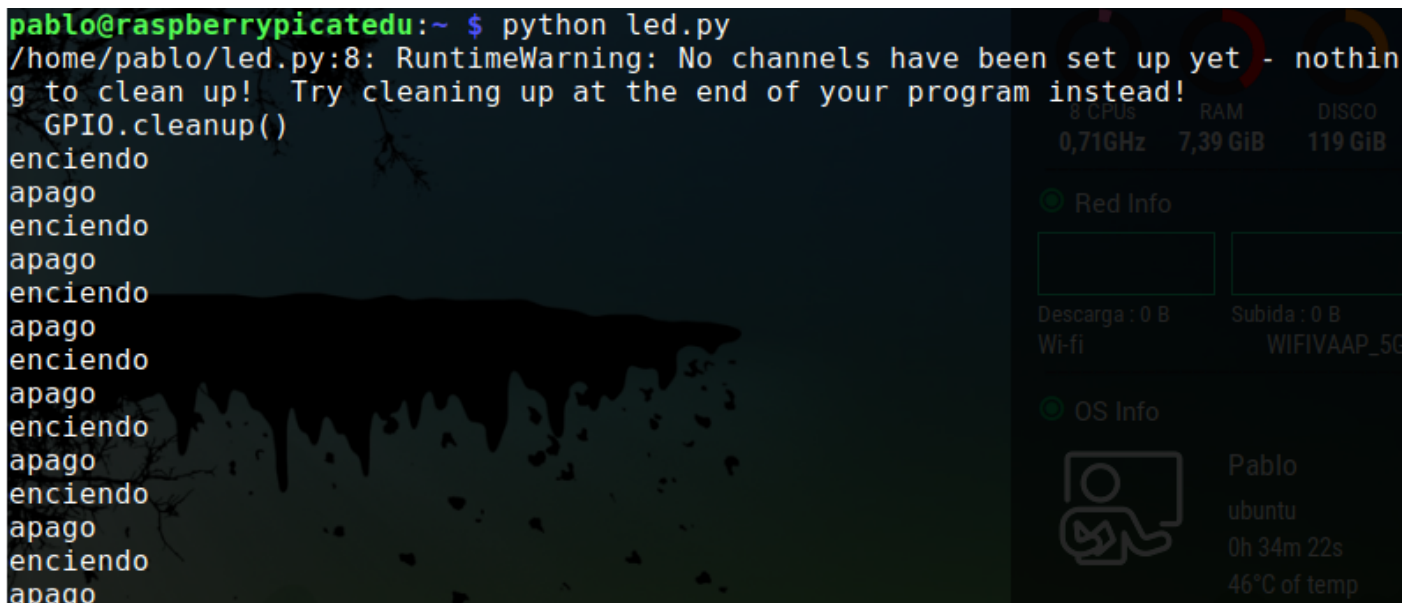
    # Escribo en el terminal
    print("apago")

    # Quito la tensión del pin 7
    GPIO.output(7,GPIO.LOW)

    # Pauso la ejecución del programa 2 segundos
    time.sleep(2)
```

para salir del editor pulsaremos `control + x` y aceptaremos los cambios. Tras ello, si ejecutamos `ls -l` veremos listado el directorio en el que nos encontramos y ahí veremos nuestro fichero. Lo ejecutaremos escribiendo `python led.py`

En el terminal veremos algo similar a



```
pablo@raspberrypicatedu:~ $ python led.py
/home/pablo/led.py:8: RuntimeWarning: No channels have been set up yet - nothing to clean up! Try cleaning up at the end of your program instead!
  GPIO.cleanup()
enciendo
apago
enciendo
apago
enciendo
apago
enciendo
apago
enciendo
apago
enciendo
apago
```

System status overlay:

8 CPUs	RAM	DISCO
0,71GHz	7,39 GiB	119 GiB

Red Info

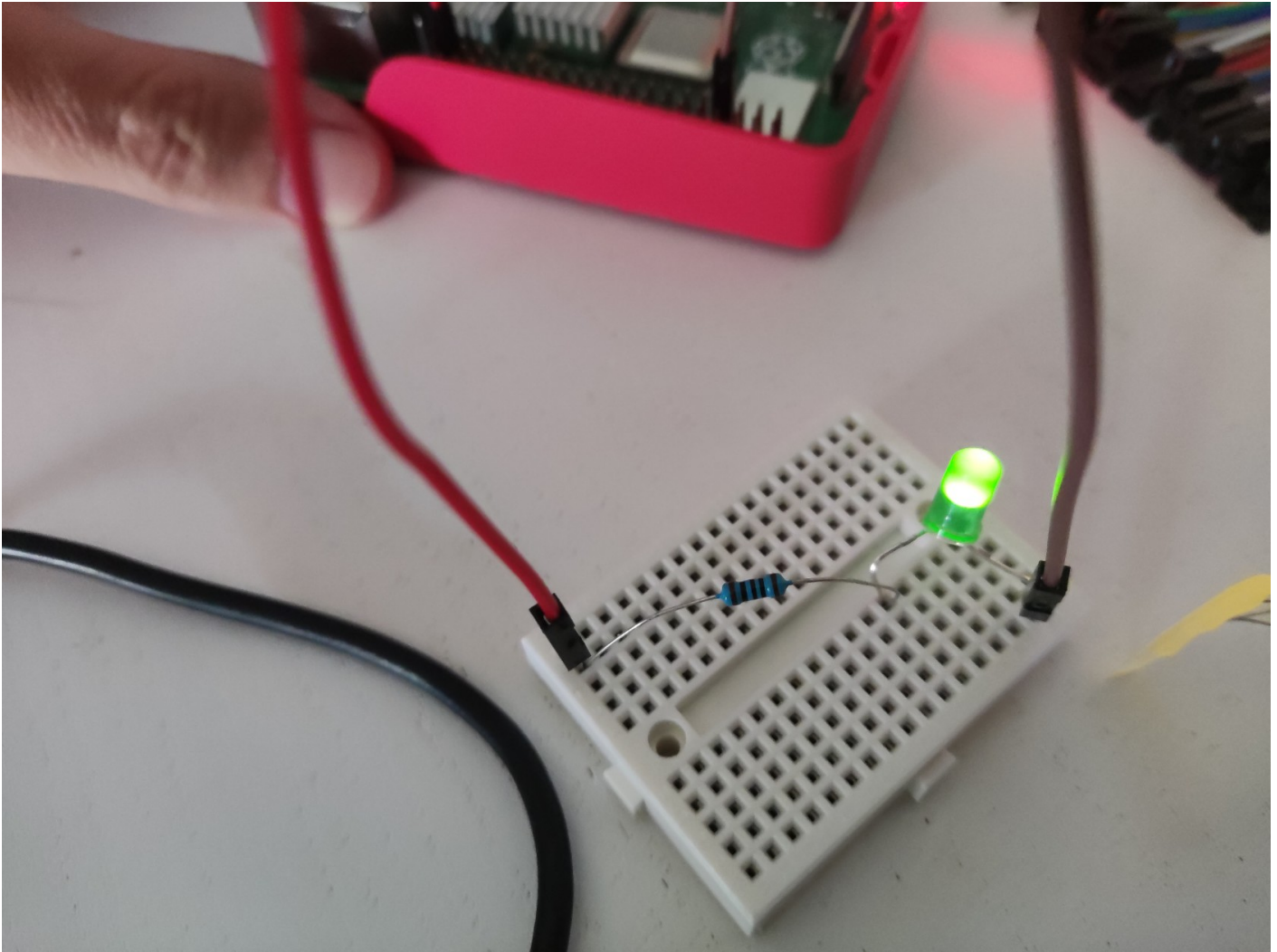
Descarga : 0 B	Subida : 0 B
Wi-fi	WIFIWAAP_5C

OS Info

Pablo
ubuntu
0h 34m 22s
46°C of temp

Elaboración propia

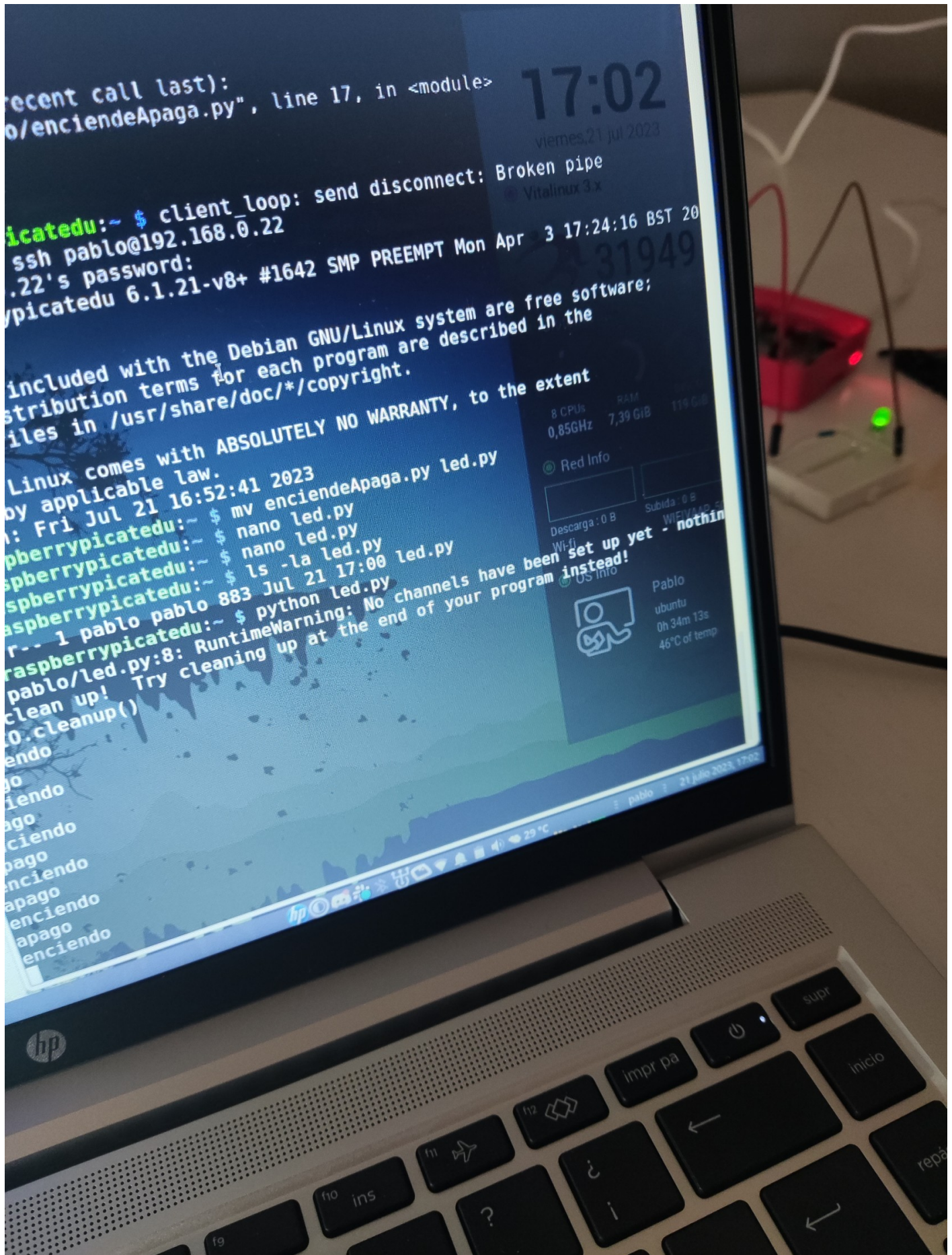
Si las conexiones que hemos realizado son correctas veremos como el led va encendiéndose y apagándose. Para detener la ejecución del programa pulsaremos `control + c`. Dejo a continuación alguna foto del programa funcionando:



Conexiones en la protoboard. Elaboración propia

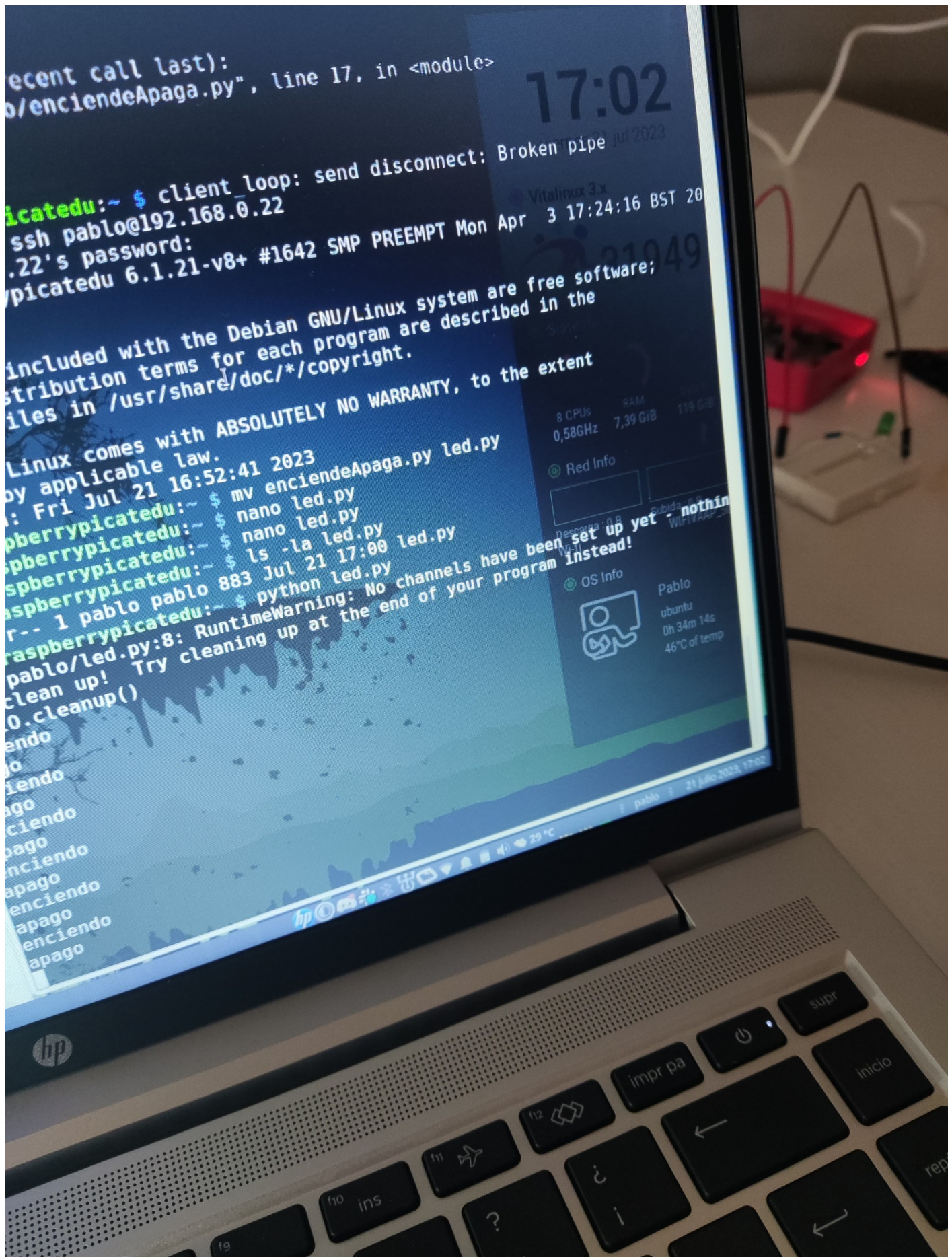


Conexiones en la Raspberry Pi. Elaboración propia





Mensaje de led encendido y led encendido. Elaboración propia



Mensaje de led apagado y led apagado. Elaboración propia

Semáforo

Materiales

- 1 led rojo
- 1 led amarillo
- 1 led verde
- 1 resistencia de 100 Ω y dos de 82 Ω pero si se utilizan todos de 100 Ω no pasa nada.
- 4 cables
- placa de conexiones

Conexiones

1. Una resistencia al cátodo de cada led
2. ánodo del led rojo a pin 3 (pin, no GPIO)
3. ánodo del led amarillo a pin 5 (pin, no GPIO)
4. ánodo del led verde a pin 7 (pin, no GPIO)
5. Resistencias a pin 25 (tierra)

Programa

Al igual que en el caso anterior crearemos el fichero con nano y lo ejecutaremos con python. Esa parte no varía por lo que no me repito. El nombre que le pongáis al programa no es importante.



```

1  # Importamos la libreria que nos permite pausar la ejecucion del programa
2  import time
3  # Importamos la libreria que nos permite conectar la raspberry con otros
4  # dispositivos a traves de los GPIO (General Purpose Input Output) pins
5  import RPi.GPIO as GPIO
6  # Ahora vamos a configurar los GPIO pins
7  # lero Borramos cualquier configuracion que hubiese
8  GPIO.cleanup()
9  # Establecemos el modo de trabajo de los pines
10 GPIO.setmode(GPIO.BOARD)
11 # Establecemos el pin 3 para actuar como salida sera nuestro rojo
12 GPIO.setup(3,GPIO.OUT)
13 # Establecemos el pin 5 para actuar como salida sera nuestro amarillo
14 GPIO.setup(5,GPIO.OUT)
15 # Establecemos el pin 7 para actuar como salida sera nuestro verde
16 GPIO.setup(7,GPIO.OUT)
17 # Bucle infinito
18 while True:
19     # verde
20     GPIO.output(7,GPIO.HIGH)
21     time.sleep(4)
22     GPIO.output(7,GPIO.LOW)
23     # amarillo
24     GPIO.output(5,GPIO.HIGH)
25     time.sleep(2)
26     GPIO.output(5,GPIO.LOW)
27     # rojo
28     GPIO.output(3,GPIO.HIGH)
29     time.sleep(3)
30     GPIO.output(3,GPIO.LOW)
31 # ejecutaremos el programa del siguiente modo
32 # sudo python semaforo.py

```

Elaboración propia

Display de 7 segmentos

Esta práctica variará en función de si usamos un display de cátodo común o de ánodo común, tenlo en cuenta.

Materiales



- Display de 7 segmentos
- 4 resistencias de $82\ \Omega$ o $100\ \Omega$
- 5 cables
- placa de conexiones

Conexiones

Si utilizas un display de cátodo común serán las siguientes:

- Cátodo del display a pin 25 (tierra)
- Pin 3 a resistencia y esta a segmento f.
- Pin 5 a resistencia y esta a segmento e.
- Pin 7 a resistencia y esta a segmento b.
- Pin 11 a resistencia y esta a segmento c.

Si utilizas un display de ánodo común serán las siguientes:

- Ánodo del display a pin 1 (3,3V)
- Pin 3 a resistencia y esta a segmento f.
- Pin 5 a resistencia y esta a segmento e.
- Pin 7 a resistencia y esta a segmento b.
- Pin 11 a resistencia y esta a segmento c.

Programa

Al igual que en el caso anterior crearemos el fichero con nano y lo ejecutaremos con python. Esa parte no varía por lo que no me repito. El nombre que le pongáis al programa no es importante.

El programa encenderá un lateral del *display* y luego otro de modo infinito. Te propongo que modifiques el programa y las conexiones para convertir esta práctica en una cuenta atrás.



```
1  # Importamos la libreria que nos permite pausar la ejecucion del programa
2  import time
3  # Importamos la libreria que nos permite conectar la raspberry con otros
4  # dispositivos a traves de los GPIO (General Purpose Input Output) pins
5  import RPi.GPIO as GPIO
6  # Ahora vamos a configurar los GPIO pins
7  # lero Borramos cualquier configuracion que hubiese
8  GPIO.cleanup()
9  # Establecemos el modo de trabajo de los pines
10 GPIO.setmode(GPIO.BOARD)
11 # Establecemos los pines a usar como salida
12 GPIO.setup(3,GPIO.OUT)
13 GPIO.setup(5,GPIO.OUT)
14 GPIO.setup(7,GPIO.OUT)
15 GPIO.setup(11,GPIO.OUT)
16 # Bucle infinito
17 while True:
18     # encendemos un lateral
19     GPIO.output(3,GPIO.HIGH)
20     GPIO.output(5,GPIO.HIGH)
21     GPIO.output(7,GPIO.LOW)
22     GPIO.output(11,GPIO.LOW)
23     # espera 2 segundos
24     time.sleep(3)
25     # encendemos el otro lateral
26     GPIO.output(3,GPIO.LOW)
27     GPIO.output(5,GPIO.LOW)
28     GPIO.output(7,GPIO.HIGH)
29     GPIO.output(11,GPIO.HIGH)
30     # espera 3 segundos
31     time.sleep(3)
32 # ejecutaremos el programa del siguiente modo
33 # sudo python contador.py
```

Elaboración propia

Otras prácticas

Con mi alumnado de FP básica otras prácticas que he llevado a cabo han sido:

- encender el led cuando no hay luz (haciendo uso del LDR)
- un detector de humedad (que suene un sonido y/o se enciende una luz cuando el circuito se cierra con la humedad)



En internet existen multitud de prácticas pa

Revision #15

Created 4 February 2023 10:59:24 by Pablo Ruiz

Updated 21 July 2023 17:23:15 by Pablo Ruiz