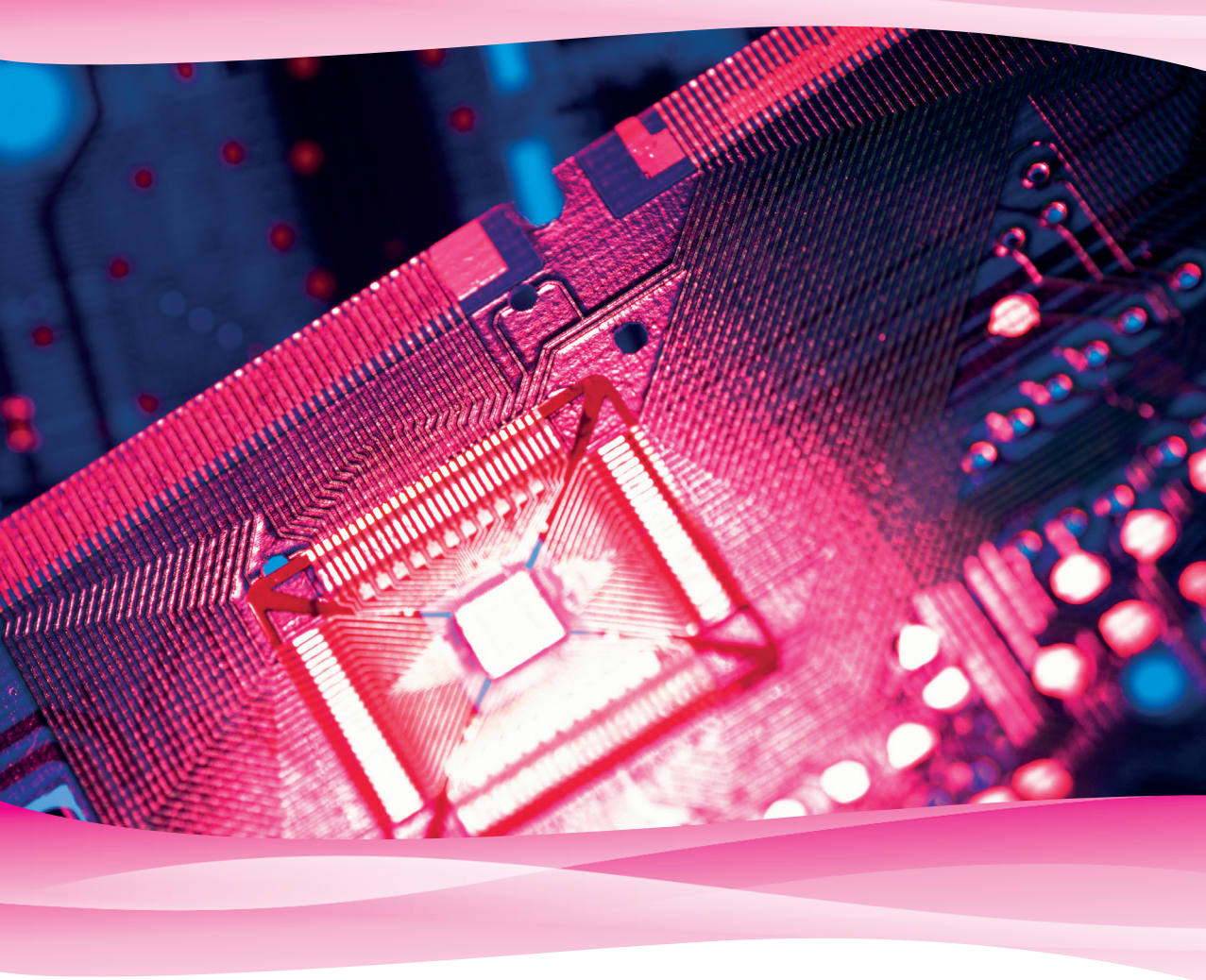


Raspberry Pi para electrónicos

Germán Tojeiro Calaza

Marcombo

RASPBERRY Pi PARA ELECTRÓNICOS

Germán Tojeiro Calaza

Marcombo

Raspberry Pi para electrónicos

Primera edición, 2016

Primera reimpresión, 2017

© 2016, GERMÁN TOJEIRO CALAZA

© MARCOMBO S.A., 2016
Gran Via de les Corts Catalanes, 594
08007 Barcelona
www.marcombo.com

«Cualquier forma de reproducción, distribución, comunicación pública o transformación de esta obra solo puede ser realizada con la autorización de sus titulares, salvo excepción prevista por la ley. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, www.cedro.org) si necesita fotocopiar o escanear algún fragmento de esta obra.»

ISBN: 978-84-267-2259-1
DL: B-20747-2015

Impreso en Gráficas Díaz Tuduri, S.L

Printed in Spain

A mi madre Enma

Índice

Prólogo

La Raspberry Pi llegó a mis manos un fin de semana de lluvia. Aún estaba enfrascado en el planteamiento inicial de mi anterior libro sobre Arduino y tan siquiera había escrito una sola palabra. Fue el comentario de un alumno avezado el que me animó a adquirir esta pequeña plaquita electrónica. Al principio, empecé a buscarle defectos en comparación con las prestaciones de Arduino. Y claro que los encontré: niveles de voltaje no compatibles con TTL, dificultades para una configuración inicial si no poseemos un monitor CRT o de plasma, documentación retorcida en lo referente a la utilización de sus pines de entrada y salida GPIO, etc.

Lo que más me desesperanzaba era el inconveniente de cómo plantear esta nueva tecnología en mis clases de electrónica. Por otra parte, todos los indicios publicados en Internet apuntaban a que en el futuro inmediato iba a ser una seria competidora de Arduino. El lunes siguiente seguía lloviendo y me fui al instituto con la impresión de haber perdido el fin de semana. Abandoné la Pi en un rincón de la mesa por unos meses.

No fue hasta un tiempo más tarde cuando tuve que instalar una distribución de Ubuntu en una máquina virtual en mi portátil y me acordé de que la Raspberry trabajaba con Linux. Se me ocurrió la feliz idea de volver a bucear en Internet y buscar nueva información sobre ella. Encontré, de repente, una riada de manuales y documentación en foros y blogs. Animado por este descubrimiento, me puse manos a la obra e instalé y configuré sin mayores problemas su sistema operativo Raspbian utilizando una aplicación tipo asistente llamado NOOBS.

Me compré un adaptador USB wifi y, en un suspiro, ya tenía la Raspberry conectada a Internet sin necesidad de una shield. Empecé a atisbar su potencia de procesamiento y cómo solventar los problemas de adaptación de voltajes a 5 voltios. Conector para cámara web, puerto USB, salida de vídeo y de audio; estas prestaciones comenzaron a tomar consistencia y sentido en mi cabeza para aplicar a infinidad de aplicaciones.

Un poco más tarde surgió en el mercado el modelo B+ con mejores características y con un número mayor de adeptos en la red. Paralelamente, aparecieron las denominadas HATS (sombreros) que expandían la potencia de la Raspberry a niveles más amplios que las famosas shields propias del mundo Arduino.

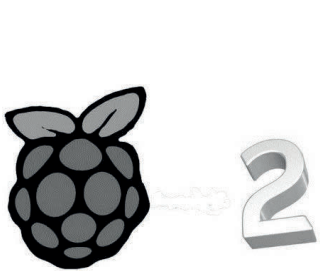
Meses después, cuando ya estaba haciendo mis pinitos con motores paso a paso, esta empresa del Reino Unido comercializó, con unas primeras estrecheces, la joya de la corona: la Raspberry Pi2, que revolucionaría el ámbito de la electrónica.

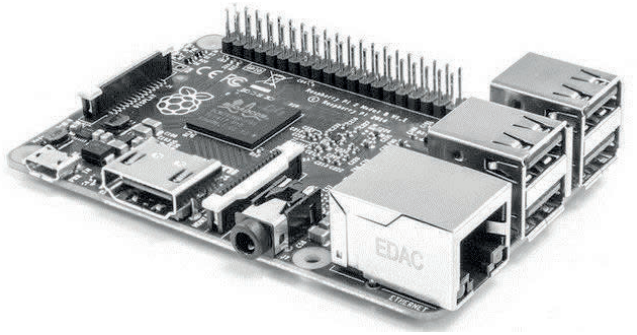
La verdad es que me fui alejando de Arduino poco a poco, como de una antigua novia a la sigues queriendo pero a la que ya no amas.

El libro que tienes en tus manos se centra en la Raspberry Pi2, aunque debido a su compatibilidad con el modelo B+ todos los proyectos electrónicos que se exponen funcionarán igualmente. Pretende ser una guía de autoaprendizaje que te permita conocer básicamente lo que es la Rasp y cómo se programa en Python. Además, aprenderás a manejar algunos dispositivos como LEDS,

sensores de diversos tipos, motores de continua, servos, motores paso a paso y, cómo no, adentrarte en el novedoso mundo del Internet de las Cosas (**IoT**).

Y no nos olvidemos de que Linux es su medio natural, por lo que si aún no lo conoces, este puede ser un buen pretexto para adentrarte en el manejo de este fascinante sistema operativo.





GERMÁN TOJEIRO CALAZA,

profesor de electrónica en el Instituto Monte Neme (Carballo, La Coruña)

Agradecimientos

En primer lugar agradecer a mi amigo Rubén Lema Rodríguez sus consejos y apoyo durante la redacción de este libro.

Agradecer igualmente los consejos de Ramón Abelenda García.

También debo reconocer la ayuda de los incontables entusiastas en la red que me han despejado cuestiones y dudas.

Finalmente dar las gracias a mi hijo Germán Tojeiro Graiño por las fotos e imágenes que contiene el presente texto.

Marcas registradas

El nombre y el logo de Raspberry Pi son marcas registradas en todos los países.

CAPÍTULO

1 COMENZANDO CON LA RASPBERRY Pi2

La Raspberry Pi es una placa de pequeño tamaño (8,5 × 5,4 cm) que incluye todo un ordenador completo. A estos dispositivos se los conoce por las siglas SBC (**Single Board Computer**). En su diseño se utiliza un SoC (**System on a Chip**), que incluye en un solo chip el procesador o CPU (ARMv6/ARMv7), la memoria RAM (512 MB/1 GB) y la tarjeta gráfica o GPU (VideoCore IV).

Los primeros modelos que salieron al mercado consistían en dos tipos de placas: el modelo A, que estaba más bien destinado a desarrolladores y poseía menos prestaciones, y el modelo B, que es el más utilizado por los usuarios domésticos. Este modelo incluye 512 MB o 1 GB de RAM, dos o cuatro puertos USB, un conector de red Ethernet 10/100, salida de vídeo HDMI y otra salida analógica de audio y vídeo. Como unidad de almacenamiento utiliza una tarjeta SD y se alimenta a través de un conector mini USB, similar al de los cargadores para teléfonos móviles. Como sistema operativo puede usar una amplia variedad de distribuciones de Linux (entre ellas, Debian, Ubuntu, Fedora o Arch Linux) u otros sistemas como FreeBSD o RISC OS.

Los primeros diseños se realizaron en el Reino Unido en el año 2006, pero no se empezó a comercializar hasta febrero de 2012 por parte de la Fundación Raspberry Pi, que es la que se encarga de su desarrollo.

En julio de 2014 se presenta una nueva placa, bajo el nombre de modelo B+, que es una actualización del anterior modelo B. Aparte del rediseño de algunos componentes electrónicos de la placa, las diferencias fundamentales entre ambas son la inclusión de 2 puertos USB más (lo que suma un total de 4), la sustitución de la tarjeta SD por una microSD y la fusión de los dos conectores analógicos (audio y vídeo) en un solo conector. Pero la parte fundamental, que es el SoC (CPU, GPU y RAM), es idéntica en los dos modelos de la clase B.

El 2 de febrer1o de 2015 salió al mercado una nueva placa: la **Raspberry Pi2** modelo B. Visualmente es idéntica al modelo B+ anterior, pero añade dos importantes novedades: una CPU de cuatro núcleos (**quad core**) ARMv7 y 1 GB de memoria RAM. Todo lo demás se mantiene exactamente igual y, por ello, es totalmente compatible con el modelo B+.

Esta placa está destinada a desarrolladores y a personas a las que les gusta aprender, experimentar y trastear con electrónica. Con ella pueden hacerse proyectos de todo tipo (robótica, domótica, etc.), pero la mayoría de los usuarios la utilizan para cosas sencillas y prácticas como las siguientes:

- ✓ Centro multimedia y servidor DLNA: se conecta a la TV por medio de la salida HDMI y permite reproducir vídeo y audio en alta definición; o servir contenido multimedia en streaming por DLNA a través de la red local.
- ✓ Servidor de almacenamiento e intercambio de ficheros (NAS y FTP), para guardar y/o compartir ficheros dentro de la red local o a través de internet.
- ✓ Nube personal: se puede instalar ownCloud y tener un sistema de nube privada y personal (tipo Dropbox) con todo el espacio que el usuario desee. Cliente de descargas Torrent.
- ✓ Servidor web: basta con instalar Apache y, si se desea, también PHP y MySQL para conseguir un sistema completo para webs y blogs, sin necesidad de contratar un hosting.

Debido a su bajo consumo (2,5 W), puede estar conectada las 24 horas del día, y como la CPU no se calienta apenas, no necesita ventiladores, por lo que es totalmente silenciosa. Los distintos modelos de toda la familia se muestran en la figura 1.1, remarcando el modelo más actual en que está centrado este libro.

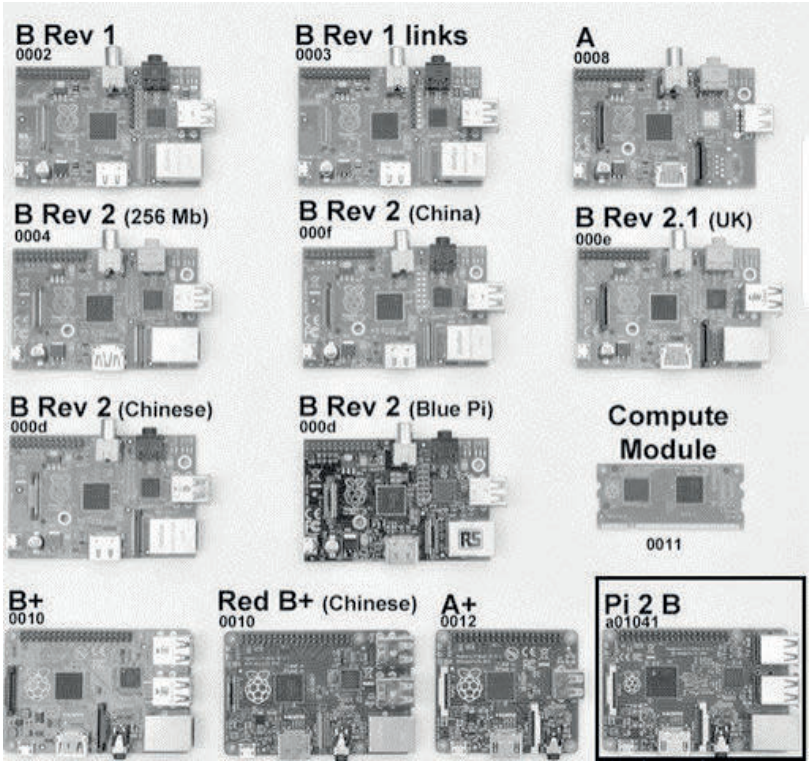


Figura 1.1

1.1 LOS ANTERIORES MODELOS DE LA RASPBERRY Pi2

Cada evolución de la Raspberry Pi es una mejora respecto a la versión anterior. El modelo B consiguió actualizar la memoria RAM del modelo A para aumentar su velocidad; mientras que el B+ era una

revisión de la totalidad, con puertos adicionales USB, un tipo diferente de tarjeta SD y más detalles. El modelo B+ presenta una diferente configuración y es mucho más grande que los otros modelos. La Pi2 mantiene el diseño de la B+ salvo que cambia el procesador y aumenta la memoria RAM. Sin embargo, cada uno de los modelos es aproximadamente del tamaño de una tarjeta de crédito. Definitivamente, con los modelos B+ y, sobre todo, la Pi2 notaremos importantes mejoras de velocidad con los programas de procesamiento de vídeo y gráficos intensivos.

1.1.1 El modelo A+

A diferencia de sus hermanos mayores, el modelo Raspberry Pi A + (figura 1.2) únicamente tiene un puerto USB, no ofreciendo puerto Ethernet. Solo dispone de 256 MB de RAM. Consume alrededor de un tercio de la potencia del modelo B, así que es genial para los proyectos de baja potencia.

La Raspberry Pi modelo A está diseñada específicamente para funcionar como equipo pequeño y portátil que no necesite acceso a internet. Es físicamente más pequeño que sus hermanos mayores y más barato. He aquí una visión general de lo que está en la pizarra:

- ✓ **256 MB de RAM**: La pequeña cantidad de memoria RAM significa que este modelo no es tan potente como los otros, pero también significa que consume menos energía.
- ✓ **Broadcom BCM2835 SoC a 700 MHz**: El procesador fue diseñado originalmente para los teléfonos móviles, pero es lo suficientemente potente como para ejecutar un completo software de escritorio.
- ✓ Doble núcleo multimedia **VideoCore IV coprocesador**: Este procesador gráfico es el mismo para los cuatro modelos. Está integrado en el SoC, pero es una pieza separada del hardware y permite a la Pi ejecutar muchos tipos de juegos y aplicaciones.
- ✓ **Ranura microSD**: Esto es para la tarjeta microSD. El Raspberry Pi no tiene un disco duro. En su lugar, se instalan los sistemas operativos en una tarjeta SD.
- ✓ **5 voltios** con ranura **microUSB**: Aquí es por donde se alimenta la Pi.
- ✓ **Conector USB**: Solo se puede conectar un dispositivo USB a la Raspberry Pi modelo A+. Esto significa que si deseamos utilizar un teclado y un ratón a la vez, necesitaremos un concentrador USB externo.
- ✓ **Puerto HDMI**: Salida de vídeo a través del puerto HDMI. La salida HDMI le da una imagen más clara de alta definición y funciona mejor con modernos monitores o televisores.
- ✓ **3,5 mm conector de audio**: Este conector de audio-plus-vídeo hace una doble función. Para el audio, se necesita un cable de audio de 3,5 mm, pero si se desea enviar vídeo compuesto, también necesitaremos un cable adaptador de 3.5 mm a RCA-3.
- ✓ **GPIO** (entrada de uso general/salida): Conector GPIO de 40 pines que le permite ser compatible con los otros modelos.
- ✓ **Conector de la cámara**: Esto le permite conectar la cámara oficial de Raspberry Pi.
- ✓ **Conector de pantalla**: El conector de la pantalla hace posible utilizar una pantalla en vez de usar el HDMI.

El modelo A+ no tiene un conector Ethernet propio, lo que significa que no puede conectarse a Internet o redes de forma nativa. Podemos solventar el problema utilizando un adaptador wifi a través del conector USB.

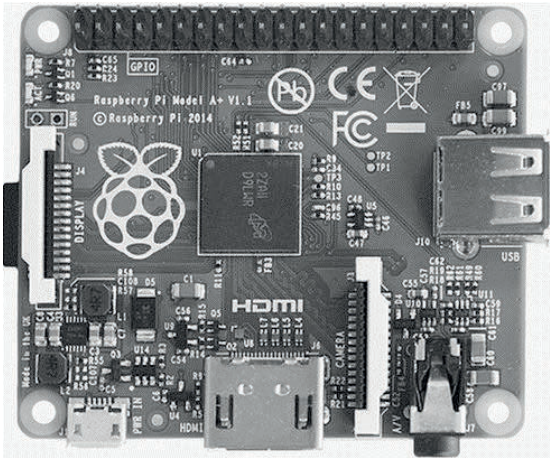


Figura 1.2

1.1.2 El modelo B

El Raspberry Pi modelo B (figura 1.3) gasta un poco más de energía que el modelo A+ ya que proporciona más memoria RAM y más puertos.

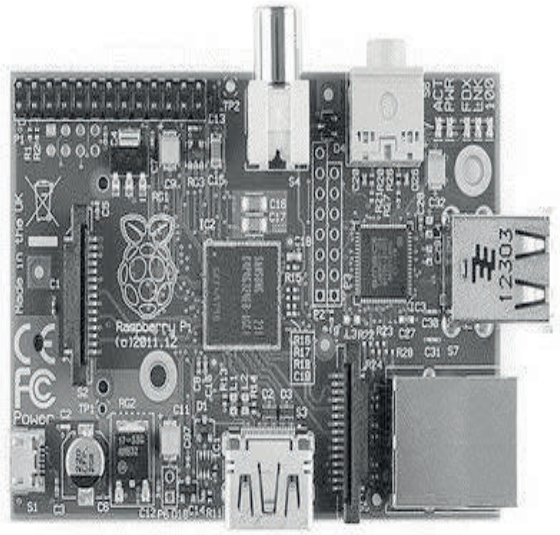


Figura 1.3

- ✓ **512 MB de RAM**: El modelo B tiene el doble de la memoria RAM que el modelo A+, por lo que puede ejecutar software más pesado. También significa que es mucho mejor ejecutando vídeo, incluyendo la codificación y emisión en HD.

- ✓ **Broadcom BCM2835 SoC a 700 MHz**.
- ✓ Doble núcleo multimedia **VideoCore IV coprocesador**.
- ✓ **SD, MMC, ranura para tarjeta SDIO**: El modelo B es el único modelo con una ranura para tarjetas SD de tamaño estándar.
- ✓ **5 voltios** con ranura microUSB: Aquí es por donde se alimenta la Pi.
- ✓ **Dos conectores USB**: Podemos conectar dos dispositivos USB diferentes.
- ✓ **10/100 Ethernet RJ45 jack**: Este conector le permite conectarse a Ethernet.
- ✓ **HDMI y salida RCA**: Son las mismas que en modelo A+.
- ✓ **3,5 mm conector de audio**: El modelo B tiene una salida de audio tipo jack de 3,5 mm.
- ✓ **GPIO** (conector de entradas/salidas): El modelo B tiene un conector de 26-pines.
- ✓ **Conector de la cámara**: Este es el mismo para los cuatro modelos.
- ✓ **Conector de la pantalla**: Este es el mismo para los cuatro modelos.

1.1.3 El modelo B+

La Raspberry Pi modelo B+ (figura 1.4) marca el primer gran avance en la línea de hardware. A diferencia del salto de las primera Raspberry al modelo B, el modelo B + cambia la arquitectura general y añade importantes mejoras. Con la excepción de una nueva ranura para tarjeta microSD, todas las mejoras en el modelo B+ se presentan en forma de nuevos puertos. Las especificaciones son básicamente las mismas que el modelo B.

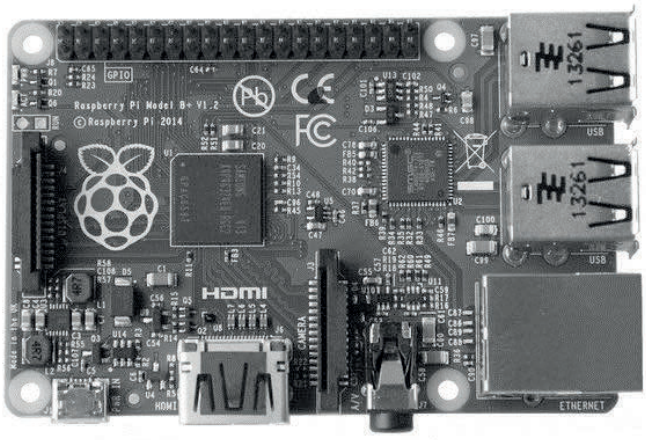


Figura 1.4

- ✓ **512 MB de RAM**: El modelo B+ tiene la misma capacidad de memoria que el modelo B.
- ✓ **Broadcom BCM2835 SoC con 700MHz**: El mismo procesador que los modelos anteriores.
- ✓ Doble núcleo multimedia **VideoCore IV coprocesador**: El mismo que los modelos anteriores.
- ✓ **Ranura de tarjeta SDIO**: Ranura para la microSD.
- ✓ **5 voltios** con ranura microUSB: Aquí es por donde se alimenta la Pi.

El modelo B+ difiere de los modelos B y A+ en que tiene más conectores USB. Tanto el A+ y el B+ tienen más pines GPIO.

- ✓ **Cuatro conectores USB**: El modelo B+ duplica los conectores USB del modelo B; tiene un total de cuatro. Esto hace que sea mucho más fácil de usar como un ordenador con un teclado, ratón, adaptador wifi y cualquier otra cosa que necesitemos conectar.
- ✓ **10/100 Ethernet RJ45**: Es el mismo que el del modelo B.
- ✓ **HDMI**: El puerto es el mismo que el del modelo B.
- ✓ **3,5 mm conector de audio**: El modelo B tiene una salida de audio idéntica al modelo A+.
- ✓ **GPIO**: El modelo B+ y el modelo A+ tienen un conector de 40 pines.
- ✓ **Conector de la cámara**: Este es el mismo para los cuatro modelos.
- ✓ **Conector de la pantalla**: Este es el mismo para los cuatro modelos.

En la figura 1.5 podemos observar comparativamente las diferencias entre los modelos B y B+, y darnos cuenta de la evolución evidente de arquitectura. La diferencia del número de pines GPIO evidencia una incompatibilidad del hardware externo.

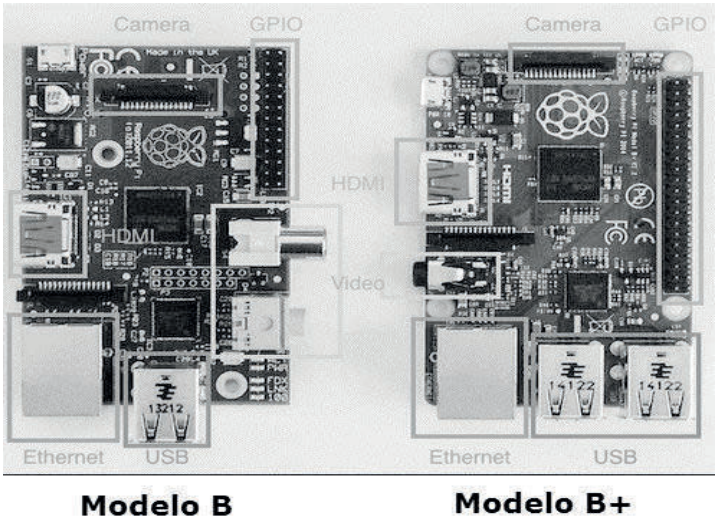


Figura 1.5

1.2 EL HARDWARE DE LA RASPBERRY Pi2

La Raspberry Pi2 llega en un momento donde otras placas de desarrollo se estaban aprovechando del tirón de la Raspberry Pi, ofreciendo en algunos casos placas más interesantes. Sinceramente estaban adelantando a la Raspberry Pi por la derecha, ofreciendo más potencia y funciones, y lo único que ha permitido a la Raspberry Pi continuar en el reinado ha sido la ingente comunidad de seguidores que tiene, en el sector educativo y en el gran conjunto de personas que han buscado un ordenador barato capaz de reproducir vídeo 1080p sin despeinarse.

La nueva Raspberry es aproximadamente 6 veces más rápida que los modelos anteriores en las tareas más frecuentes. Atrás quedan también los modelos con 256 y 512 MB, ya que ahora la memoria RAM es de 1 GB. Como detalle curioso apuntar que es la caja más grande de todas las Raspberry Pi, ya que ahora contamos con un manual de inicio rápido y uso seguro en muchos idiomas, aunque la parte destinada a cada idioma ocupa unas 8 hojas. Además del manual, como siempre nos encontramos a la Raspberry Pi2 con una bolsa electrostática de protección. La placa base Raspberry Pi2 modelo B, del tamaño de una tarjeta de crédito, cuenta con un chip Broadcom BCM2836 con cuatro núcleos a una frecuencia de 900 MHz con arquitectura de 32 bits, en lugar del chip de un solo núcleo a 700 MHz de las anteriores versiones. Sus características más relevantes en comparación con su modelo anterior se muestran en la tabla 1.1:

	Raspberry Pi modelo B+	**Raspberry Pi 2 modelo B**
CPU	ARM11 ARMv6 700 MHz	ARM11 **ARMv7** ARM Cortex-A7 **4 núcleos** @ 900 MHz
SoC	Broadcom BCM2835	Broadcom **BCM2836**
Overclocking	Sí, Hasta 1000 MHz	Sí Hasta 1000 MHz
GPU	Broadcom VideoCore IV 250 MHz. OpenGL ES 2.0	Broadcom VideoCore IV 250 MHz. OpenGL ES 2.0
RAM	512 MB LPDDR SDRAM 400 MHz	**1 GB** LPDDR2 SDRAM 450 MHz
USB	4	4
Salidas de vídeo	HDMI 1.4 @ 1920 x 1200 píxeles	HDMI 1.4 @ 1920 x 1200 píxeles
Almacenamiento	microSD	microSD
Ethernet	Sí 10/100 Mbps	Sí 10/100 Mbps
Consumo	5 V, 600 mA	5 V, **900 mA**, aunque depende de la carga de trabajo de los 4 cores
Precio estimativo	33,60 € (sin IVA)	39,50 € (sin IVA)

Tabla 1.1

Como se puede comprobar en la tabla anterior y salvo algunas características tales como la GPU, conectividad Ethernet, tamaño, peso y precio en las que empatan, la Raspberry Pi2 mejora bastante a su predecesora en cuanto a CPU, velocidad y cantidad de memoria. El nuevo procesador no es que sea ya más potente gracias a los cuatro núcleos, sino que abre las puertas a otro tipo de distribuciones y sistemas operativos (Ubuntu Mate, Windows 10) e incluso a extensiones virtualizadas que basan su programación en las instrucciones más potentes del ARMv7. La memoria, mayor en

cantidad y en velocidad, ayuda a dar ese empujón que necesitaba la Pi2 para proyectos más complejos. La diferencia de precio entre los dos modelos es insignificante si atendemos a las mejoras de la Pi2, por lo que es realmente aconsejable adquirir esta última si somos nuevos en este terreno.

La nueva Raspberry Pi2 ofrece mejores prestaciones, sí, pero no ha renunciado a ser una placa de desarrollo barata y orientada a la educación. Por todo ello, tanto el sector educativo como los aficionados a la electrónica, podemos estar contentos.

Antes de conectar nada, es recomendable inspeccionar visualmente la placa. También debemos familiarizarnos con las diferentes conexiones. En la figura 1.6 se muestra el hardware de la Pi2.

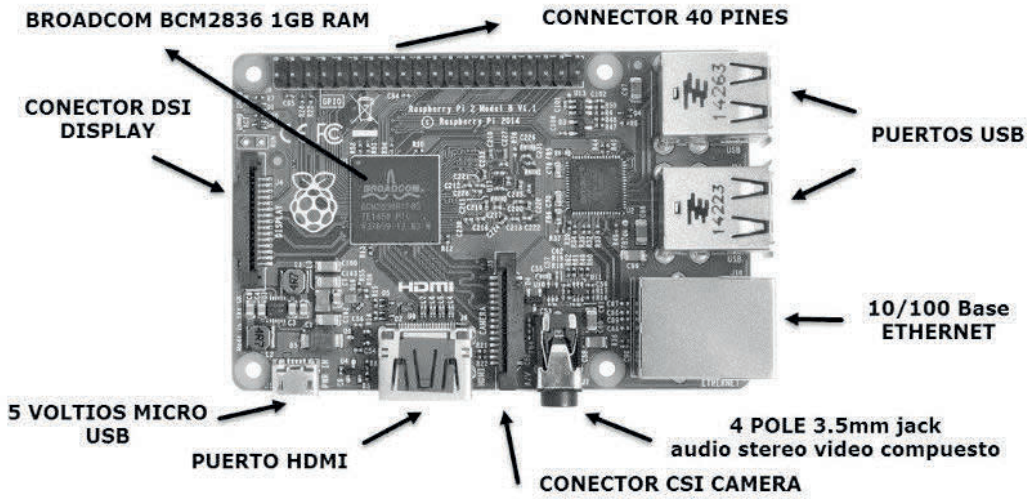


Figura 1.6

En primer lugar, alimentamos la placa mediante el conector microUSB a 5 voltios. Necesitamos una fuente que proporcione 2 amperios para que funcione apropiadamente. El LED de color rojo debe encenderse. El LED verde parpadeará en cuanto instalemos la tarjeta de memoria microSD, indicando que arranca el sistema operativo. **Esta acción la abordaremos más adelante**. De momento, nos conformamos con que el LED rojo se encienda indicando que la placa recibe alimentación.

1.3 INSTALANDO EL SISTEMA OPERATIVO EN LA RASPBERRY Pi2

Es necesario instalar un sistema operativo en la tarjeta microUSB. La pregunta que nos hacemos es cuál de ellos. Raspberry nos ofrece varios tipos que van de las diversas distribuciones o sabores Linux a la reciente Windows 10 IOT. Todo depende de lo que queramos hacer con ella. Para comenzar a movernos en este mundo, el sistema operativo más recomendable es **Raspbian**. Si lo que queremos es

utilizar la Raspberry Pi como Media Center, disponemos de varias distribuciones como **OpenELEC** o **RaspBMC**. También podríamos instalar **Ubuntu Mate** o incluso la Windows 10 IOT. Ambas posibilidades están solo reservadas, claro está, para la nueva Pi2.

Por otra parte, si somos principiantes, es recomendable utilizar aplicaciones como **NOOBS** que nos facilitan la instalación de varias distribuciones. Tan solo tenemos que introducir la tarjeta de memoria en la ranura de la placa, arrancar e indicar qué sistema operativo queremos instalar, y la aplicación lo hará por nosotros en un proceso totalmente automatizado.

Empecemos instalando Raspbian, con la herramienta NOBBS. Para descargarnos esta aplicación entramos en el siguiente enlace:

https://www.raspberrypi.org/downloads/

Descomprimimos el fichero en el directorio por defecto que contendrá subdirectorios y ficheros. Ahora necesitamos una tarjeta de memoria microUSB de al menos 8 GB, como la que se muestra en la figura 1.7, para volcarle NOOBS.

Figura 1.7

Ahora preparamos la tarjeta microUSB formateándola con la aplicación gratuita y recomendada: **SD Formatter 4.0,** que descargamos del siguiente enlace:

https://www.sdcard.org/downloads/formatter_4/

Desde el propio entorno (figura 1.8) tendremos que configurar la opción **FORMAT SIZE ADJUSTMENT** en **ON** para asegurarnos de que todo el volumen de la SD será formateado y no solo una única partición.

Los usuarios de Linux pueden acudir al siguiente enlace para completar una serie de instrucciones para el formateo de la tarjeta de memoria:

http://qdosmsq.dunbar-it.co.uk/blog/2013/06/noobs-for-raspberry-pi/

Finalmente, una vez formateada, simplemente copiamos todos los subdirectorios y ficheros que contiene la imagen en la SD. Seguidamente, la expulsamos del ordenador y ya la podemos insertar en nuestra Raspberry.

Para arrancar NOOBS necesitamos, como opción más sencilla, un televisor o un monitor con entrada HDMI y un teclado y un ratón USB. En la figura 1.9 se observa el conexionado de estos elementos.

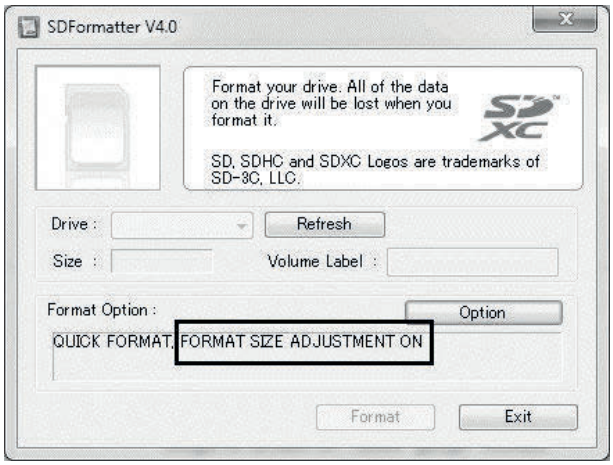


Figura 1.8

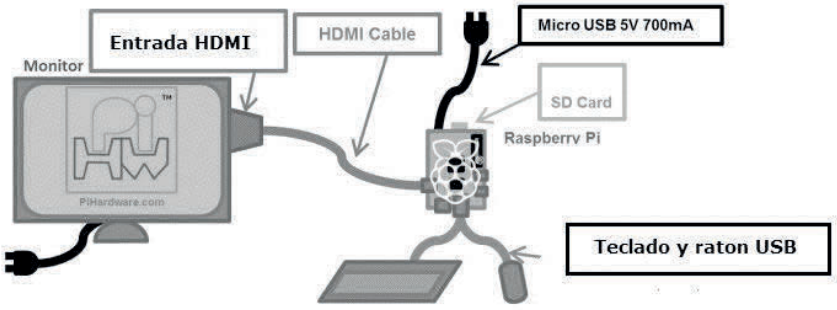


Figura 1.9

Al alimentar la Pi2 nos va aparecer la pantalla de instalación de NOOBS en el canal HDMI del monitor o televisor tal como se muestra en la figura 1.10. Debemos seleccionar con el ratón la casilla marcada como Raspbian. Desde ese momento, el proceso es automático mostrándonos una barra de progreso porcentual del total de la instalación. Cuando finalice esta, la Pi2 se debe reiniciar para completar el proceso.

Al iniciarse de nuevo entraremos en la configuración particular de nuestra Raspberry tal y como se observa en la figura 1.11. A partir de ahí nos moveremos utilizando el teclado. En primer lugar, ampliaremos el sistema de archivos para ocupar la totalidad de la tarjeta. Aceptaremos la primera opción denominada: **Expand filesystem**.

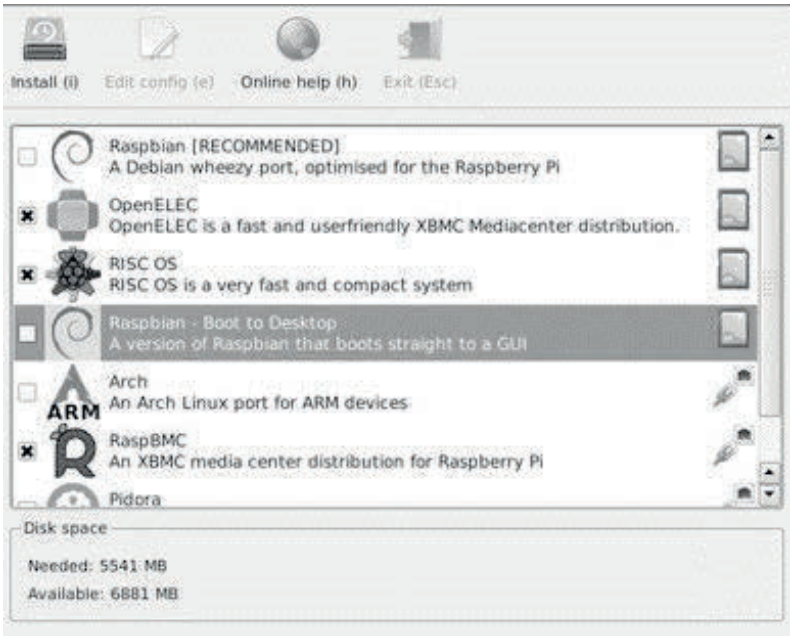


Figura 1.10

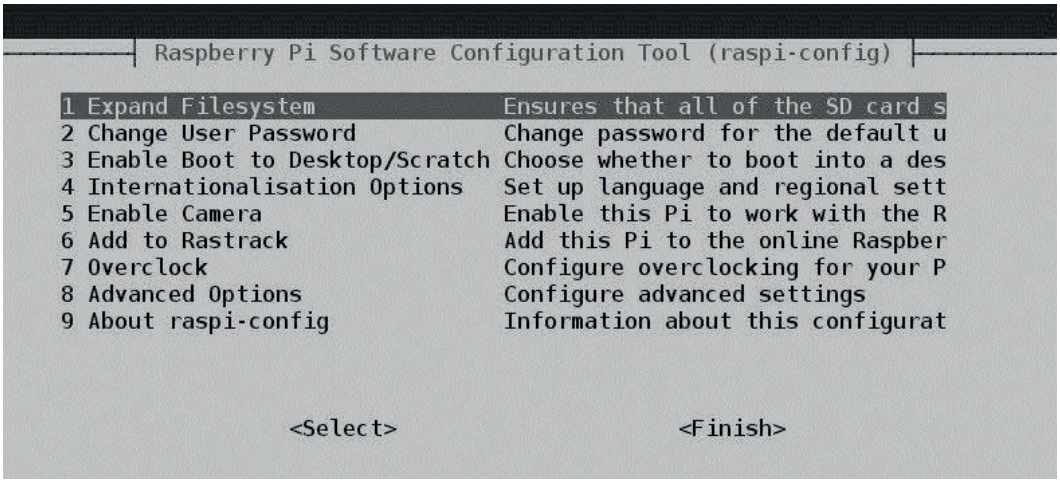


Figura 1.11

A continuación tenemos la posibilidad de cambiar el nombre de usuario y contraseña que por defecto son **pi** y **Raspberry**. Esto lo acometemos tal y como se observa en la figura 1.12. Después es necesario cambiar la configuración del teclado (ES) y la zona horaria (figuras 1.13 y 1.14). La siguiente configuración es muy importante ya que ahí estableceremos el tipo de Raspberry, que en nuestro caso es la Pi2 (figuras 1.15 y 1.16).

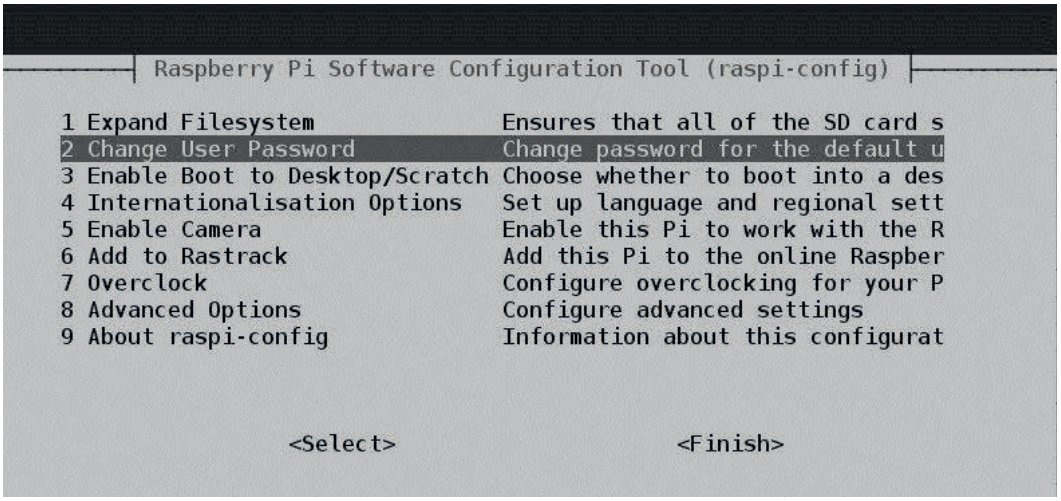


Figura 1.12

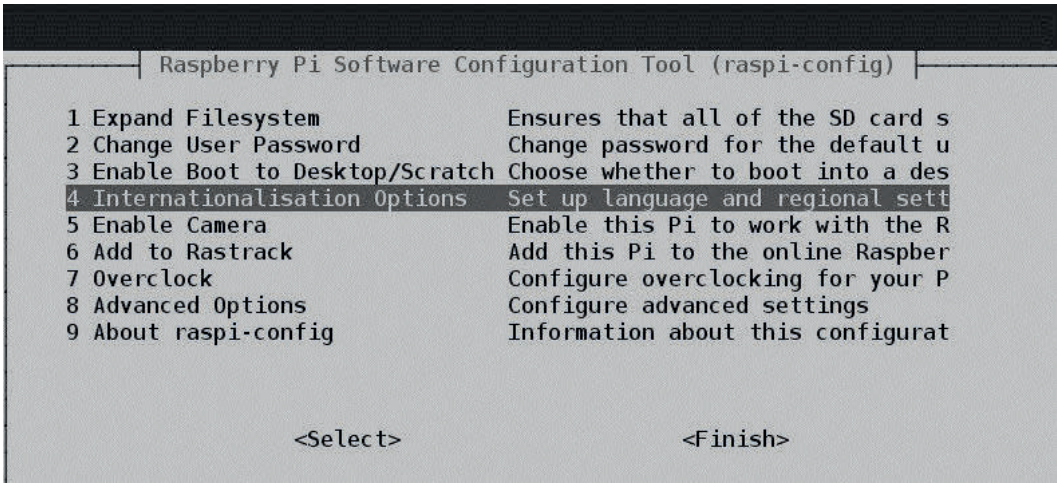


Figura 1.13

Existen otras opciones avanzadas en este menú que abordaremos más adelante cuando sea necesario. Al finalizar la configuración, la Pi2 se reiniciará y, cuando vuelva a arrancar, nos mostrará una consola en la que se nos solicitará el nombre de usuario y contraseña. Tras esto, ya estamos en disposición de trabajar con ella. La pregunta que seguro que nos planteamos en este momento es si estamos obligados a permanecer “unidos” o “pegados” al monitor o televisor para siempre. La respuesta es negativa. En el capítulo siguiente aprenderemos cómo independizar nuestra Pi2.

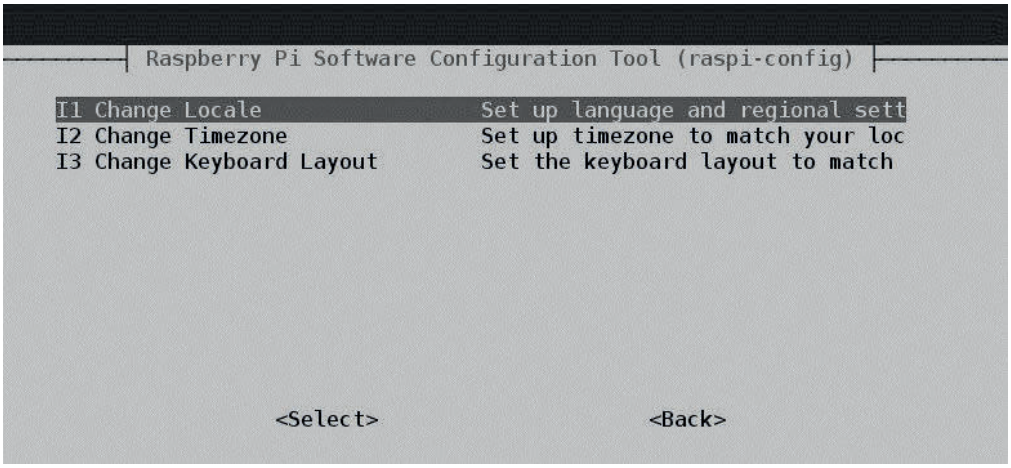


Figura 1.14

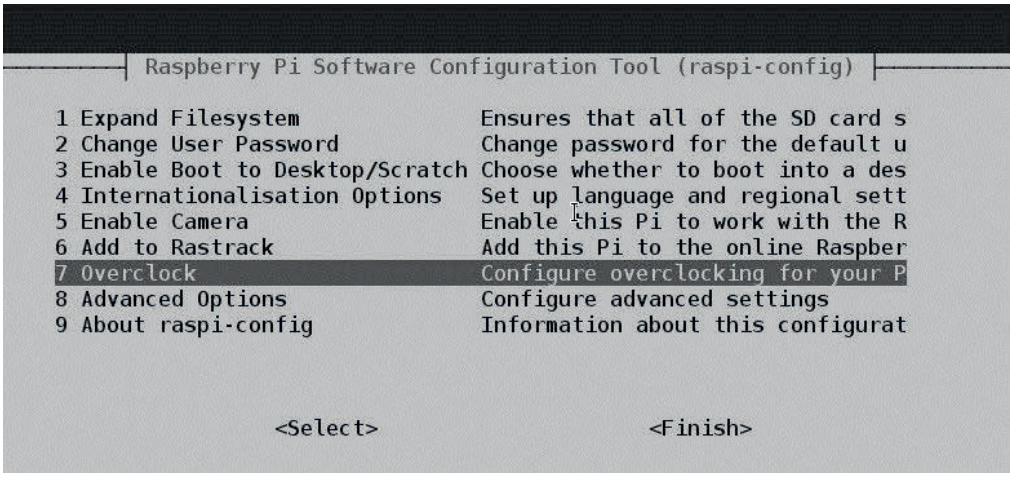


Figura 1.15

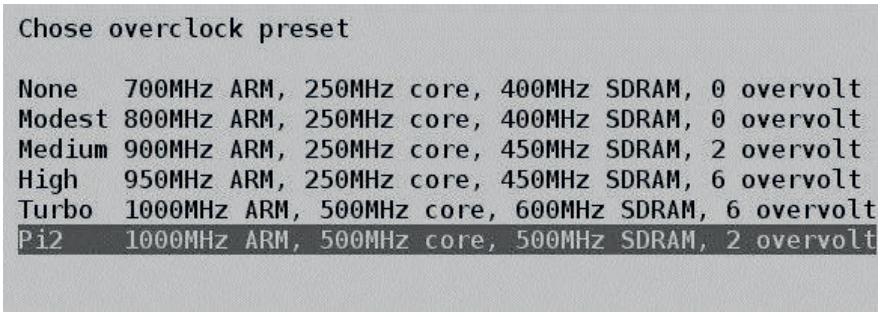


Figura 1.16

CAPÍTULO

2 CONFIGURANDO LA RASPBERRY EN RED

Hasta ahora hemos instalado el sistema operativo y configurado nuestra Raspberry Pi utilizando el monitor o televisor como única posibilidad de visualización. Sin embargo, este método, si bien es necesario en un primer momento, en un futuro inmediato deberíamos "independizar" nuestra Rasp y poder trabajar e interactuar con ella desde un ordenador y sin cables; es decir, usando la tecnología de redes sea cableada o inalámbricamente.

La Rasp está diseñada para ser conectada a redes y en especial a Internet. Su capacidad para comunicarse es una de sus principales características y abre todo tipo de posibilidades de uso, incluyendo domótica, web, monitoreo de datos, etc. La conexión se puede establecer a través de un cable Ethernet o mediante un módulo o "pincho" wifi/USB (muy utilizado hoy en día).

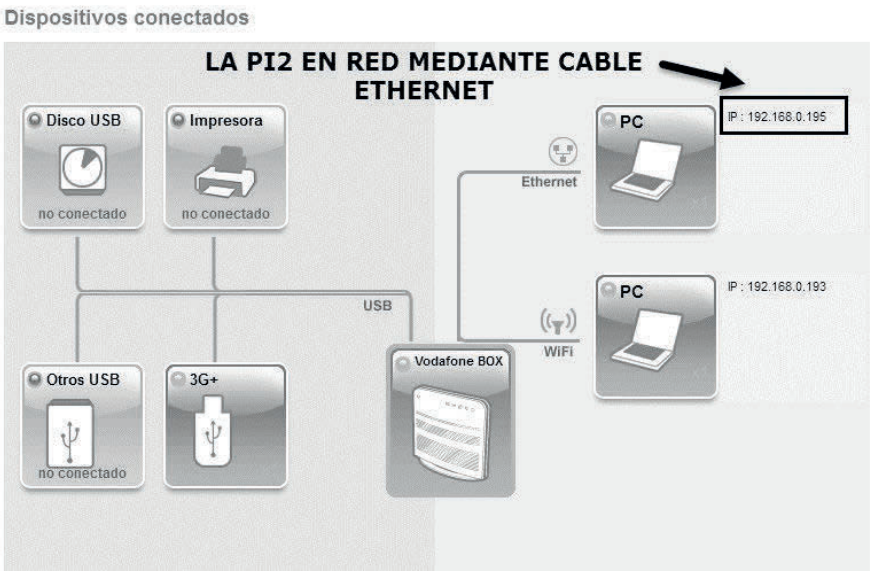


Figura 2.1

Las siguientes dos formas o posibilidades por las cuales vamos a poder acceder a nuestra Rasp desde nuestro ordenador son:

• A través de un protocolo de interfaz de terminal llamado **SSH**.

• El uso de un programa llamado **vncserver**. Esto nos permitirá abrir una interfaz de usuario, que reflejará la interfaz gráfica de usuario en la Raspberry remota.

Así que primero debemos asegurarnos de que nuestro sistema básico está en marcha y funcionando. Conectamos la Raspberry Pi al router de casa mediante un cable Ethernet. Desde el ordenador abrimos el navegador y entramos por web en la configuración de nuestro router. Desde allí podremos observar la dirección IP que le ha sido asignada a nuestra Pi2. En mi caso, la ventana de Vodafone me muestra una pantalla como la figura 2.1.

Si deseamos averiguar la IP de nuestra Raspberry de otra forma, podemos utilizar bajo Windows, el software gratuito denominado **Advanced IP Scanner** (http://www.advanced-ip-scanner.com/es/) tal y como se muestra en la figura 2.2.

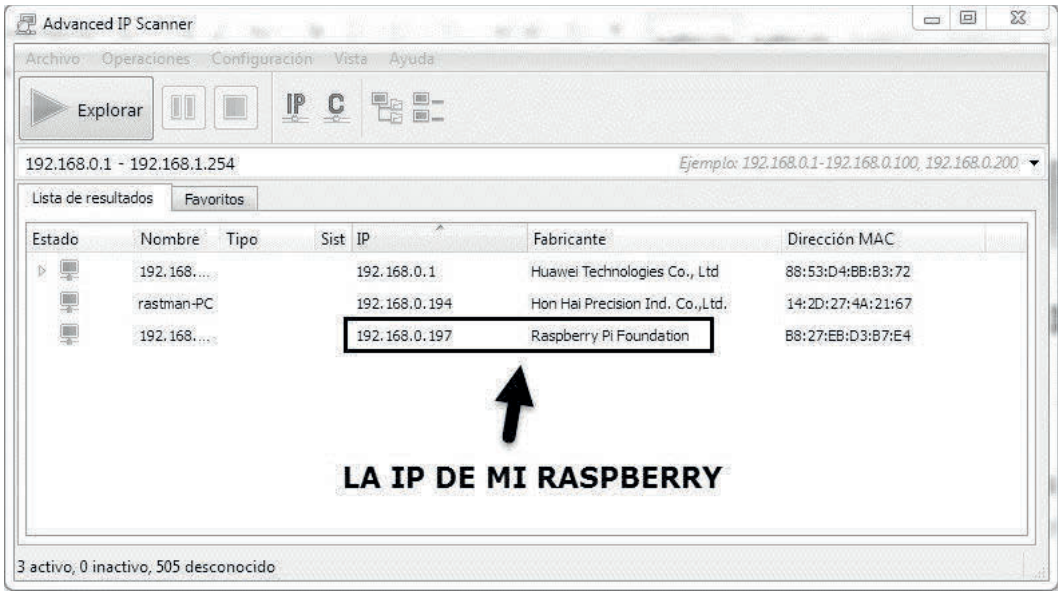


Figura 2.2

Disponemos de una versión instalable y otra portable. Antes de nada es importante seleccionar el campo o margen de exploración. En mi caso: "192.168.0.1-192.168.1.254". Ahora tan solo es necesario presionar la pestaña: "explorar" y automáticamente realizará un testeo de todos los equipos y hardware conectados a la red así como los puertos abiertos de los mismos. Si trabajamos con Linux, simplemente podemos usar **nmap** o **Angry IP Scanner**, éste último presenta muchas similitudes con Advanced IP Scanner.

2.1 CONEXIÓN EN RED CON CABLE ETHERNET

Ahora ya sabemos la dirección IP de nuestra Pi2. En este momento queremos acceder remotamente a la misma desde nuestro ordenador (no hay que olvidar que la Rasp debe estar conectada directamente a nuestro router mediante cable Ethernet). Ello nos permitirá realizar cualquier tipo de operación o programación en ella sin tenerla físicamente cerca ni depender del monitor o televisor, teclado y ratón para trabajar con ella como cuando la configuramos en un principio.

Con la IP a mano, necesitaremos un programa de terminal SSH que ejecute en el ordenador un terminal SSH, que es una conexión **Secure Shell Hyperterminal (SSH)**, y que básicamente sirve para acceder remotamente a la Raspberry Pi. Si estamos bajo Windows, podemos descargar una aplicación de este tipo como es **PuTTY**. Es gratuita, portable (http://ww.putty.org) y muy simple de utilizar tal y como se muestra en la siguiente figura 2.3:

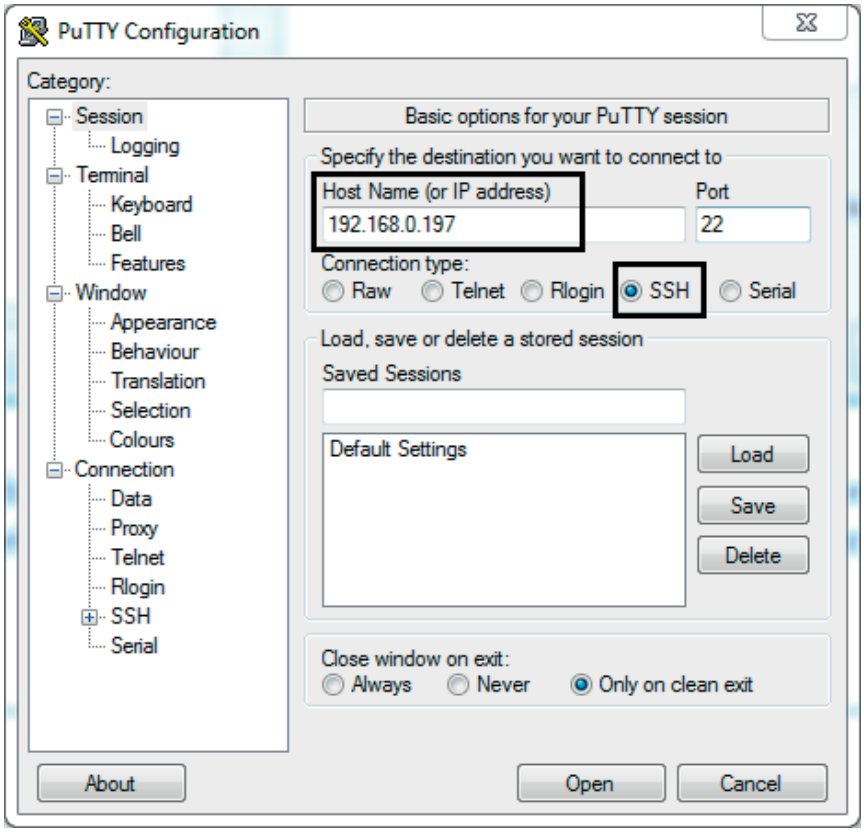


Figura 2.3

En el campo **Host Name** especificamos la dirección IP de la Raspberry que hemos averiguado anteriormente. Nos aseguramos que esté seleccionada la casilla **SSH** y el puerto 22 (vienen por defecto). Presionamos el botón **Open** y se abrirá un terminal en forma de ventana tal y como se muestra en la figura 2.4.

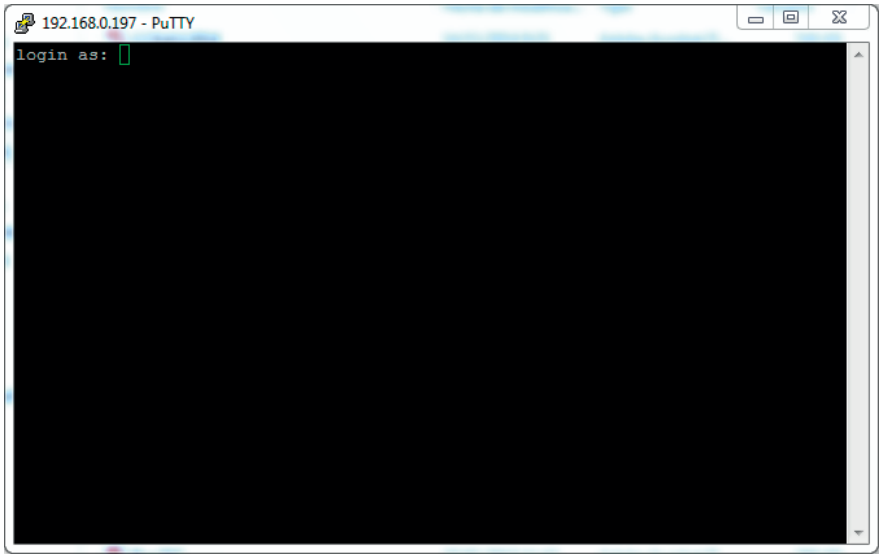


Figura .2.4

A través del terminal se nos solicita el usuario y la contraseña (**pi** y **Raspberry** respectivamente, como vimos antes) que hemos aceptado en la instalación inicial y no hemos cambiado de momento. Tras esto, se no muestra una pantalla de bienvenida como se observa en la figura 2.5.

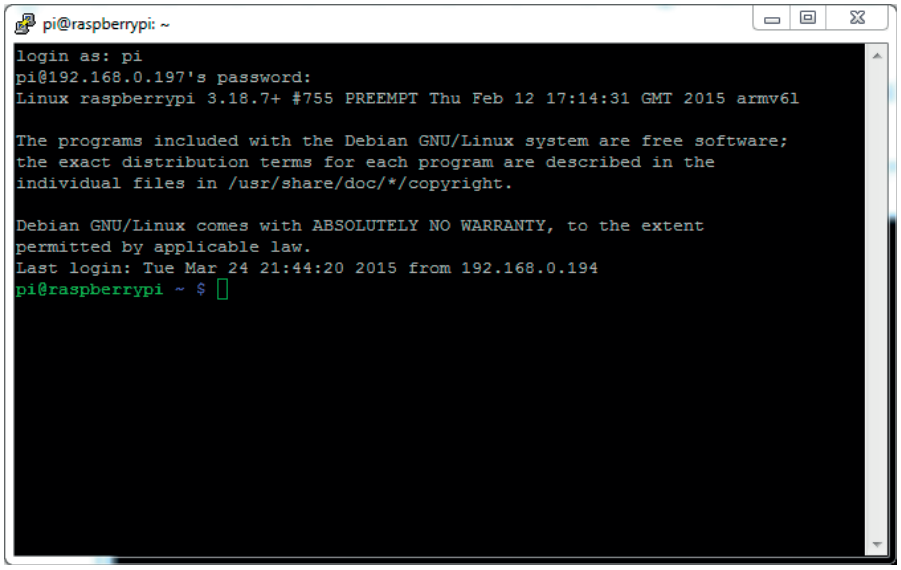


Figura 2.5

Ahora podemos conectarnos y ejecutar comandos en nuestra Pi2. Si deseamos hacer esto desde una máquina Linux, el proceso es aún más simple. Abrimos una ventana de terminal y a continuación escribimos:

```
ssh pi 157.201.194.187 -p 22
```

Este comando nos llevará a la anterior pantalla de bienvenida anterior (figura 2.4).

SSH es una herramienta muy útil para comunicarse con nuestras Pi2. Sin embargo, a veces se necesita un entorno gráfico en el sistema.

Nosotros podemos obtener y manejar el entorno gráfico de la Rasp en el ordenador a través de una aplicación llamada: servidor VNC. Tendremos que instalar una versión en nuestra Raspberry Pi utilizando el comando: *`sudo apt-get install tightvncserver`* en una ventana abierta de terminal. Por cierto, es una oportunidad magnífica para usar SSH. Una vez ha sido instalado, es necesario iniciar este servicio. Esto se hace por medio del siguiente comando:

```
vncserver :1 -geometry 1280x800 -depth 16 -pixelformat rgb565
```

El comando **vncserver** tiene diferentes variantes. El comando anterior es básico y debe tenerse en cuenta que ":1" indica el número del escritorio que está corriendo y necesitará este número cuando vaya a acceder remotamente. Podemos crear diferente cantidad de escritorios si lo deseásemos: ":2 ", ":3 " … El modificador **–geometry** indica el tamaño de la pantalla en pixeles, lo podemos modificar de acuerdo al tamaño del monitor local. El modificador **-depth** funciona para la profundidad del color, en este caso 16 bits, y por último el modificador **-pixelformat** indica la presentación del color (este modificador lo podemos omitir).

La primera vez que ejecutamos **vncserver** nos pedirá una contraseña para dar acceso al escritorio remoto, esta será solicitada cada vez que queramos acceder el escritorio. Es recomendable que usemos la misma contraseña de la Pi2 o alguna otra fácil de recordar. También nos solicitará una contraseña de solo-lectura *(read-only)*; es totalmente opcional.

Existen diferentes herramientas para acceder remotamente. Una aplicación muy sencilla es **Real VNC Viewer** (https://www.realvnc.com/). Es gratuita y tan pronto lo descarguemos y ejecutemos, nos presentará una ventana como la que se muestra en la figura 2.6.

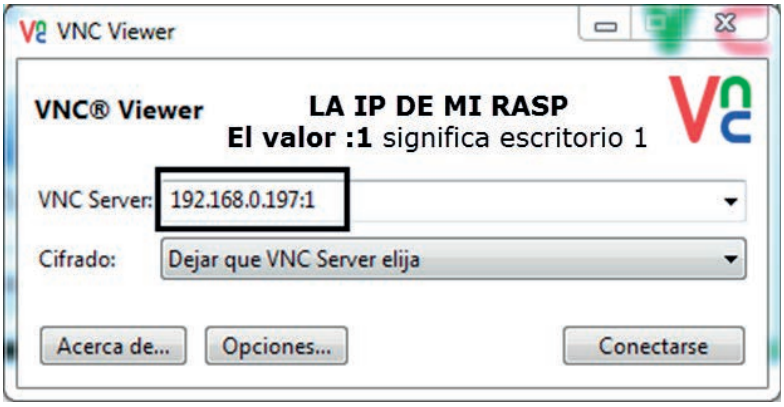


Figura 2.6

Introducimos la dirección del servidor VNC, que es la dirección IP de la Pi2, y conectamos. Recibiremos esta ventana emergente (figura 2.7).

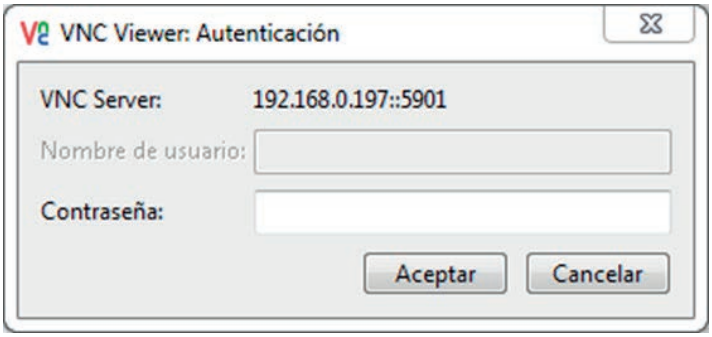


Figura 2.7

Introducimos la contraseña previamente definida y aceptamos. Inmediatamente emerge en nuestro ordenador una gran ventana que contiene el escritorio gráfico de nuestra Raspberry Pi y desde el cual podemos realizar infinidad de cosas (figura 2.8).

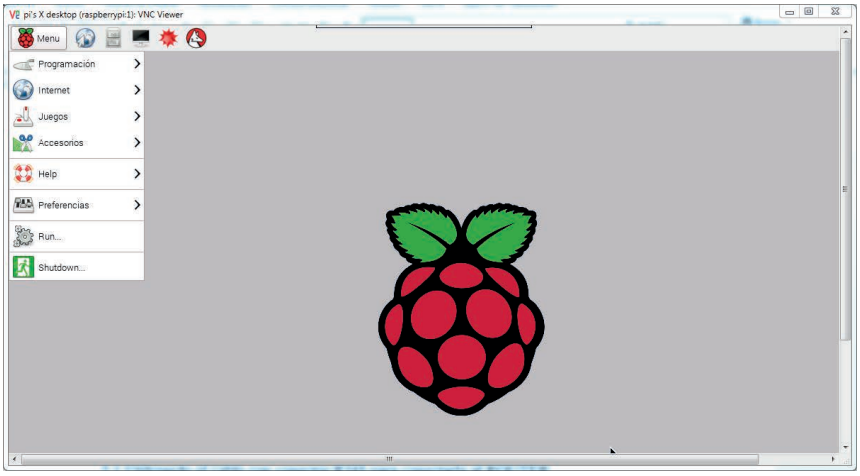


Figura 2.8

Si somos afortunados y nuestro sistema operativo preferido es Linux, tenemos una versión apropiada y descargable de esta estupenda aplicación llamada: **Real VNC Viewer**. Su apariencia y funcionamiento es idéntica a la vista anteriormente para Windows.

2.2 CONEXIÓN EN RED CON WIFI

Ahora vamos a aprender a configurar el pincho wifi en nuestra Raspberry Pi. Con ello nos desharemos del último cable que esclavizaba a nuestra Raspberry con una localización cercana a un router.

Para ello necesitaremos un adaptador wifi (figura 2.9) para conectarlo vía USB. Existe una lista en Internet de varios periféricos que se han probado y que ya se sabe si sirven o no: http://elinux.org/RPi_VerifiedPeripherals.

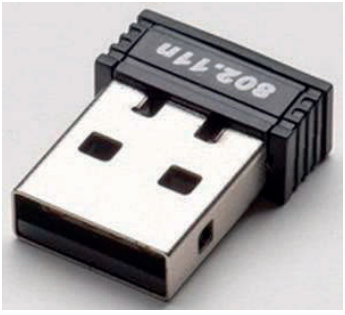


Figura 2.9

Con el wifi conectado a la Rasp, abrimos un terminal por SSH y tecleamos el comando **ifconfig**. Nos debe aparecer una pantalla como la siguiente figura 2.10 en la que observamos la referencia al adaptador **wlan** que antes no teníamos. Esto nos indica que la distribución Raspbian ha reconocido nuestro "pincho" y vamos por buen camino. Ahora vamos a configurar la conexión wifi.

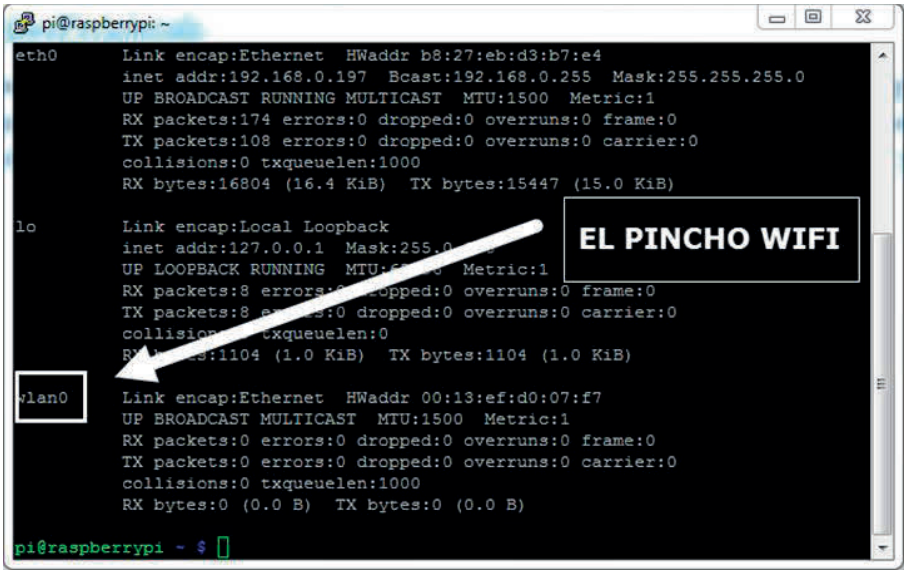


Figura 2.10

Para ello tecleamos desde el terminal:

```
sudo nano /etc/network/interfaces
```

Con ello editamos el fichero *interfaces*, que es donde está toda la configuración básica de redes de la Rasp.

Observamos que no está configurada la conexión wifi a través del adaptador **wlan** (figura 2.11). Para definirla es necesario cambiar algunas cosas en este fichero. En primer lugar establecemos la conexión wlan como **dhcp** y a continuación escribimos la **SSID** del router y su contraseña (para los neófitos: el nombre de tu router que aparece en la lista de redes del Windows y la contraseña para conectarnos a Internet). El resultado lo observamos en la figura 2.12.

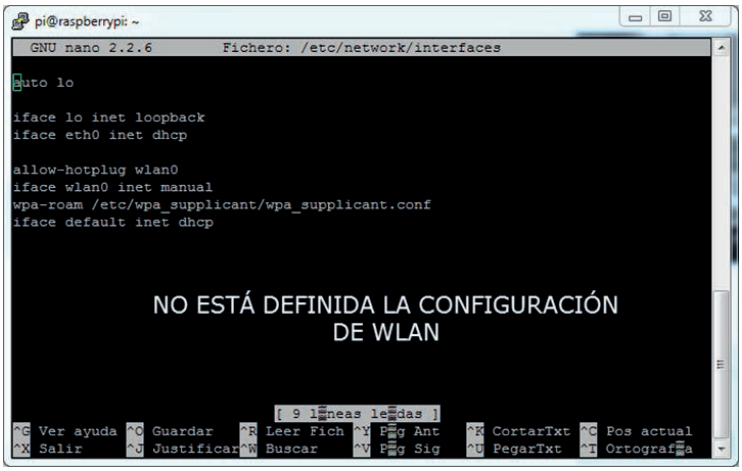


Figura 2.11

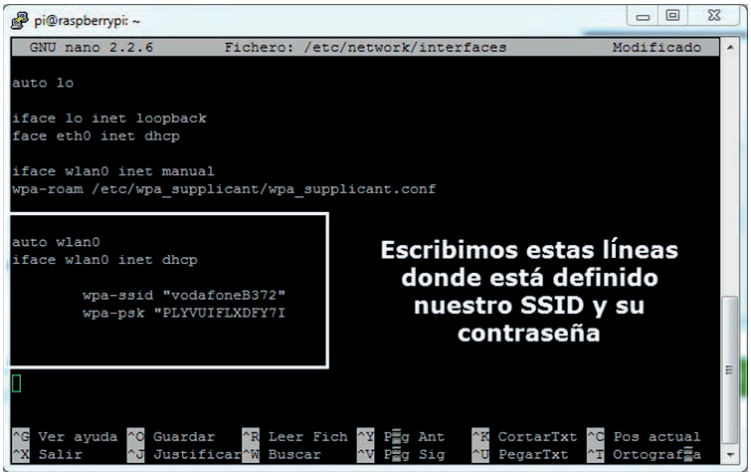


Figura 2.12

Para guardar los cambios en el fichero *interfaces* presionamos CONTROL+O y para salir CONTROL+X. Reiniciamos la Raspberry con:

```
sudo shutdown -r now
```

Volvemos a abrir un terminal SSH pero ahora, al desconectar la conexión Ethernet al router, tenemos otra dirección IP asignada al pincho wifi. Podemos abrir con el navegador web la configuración de nuestro router o utilizar la aplicación **ipscan** para averiguarla.

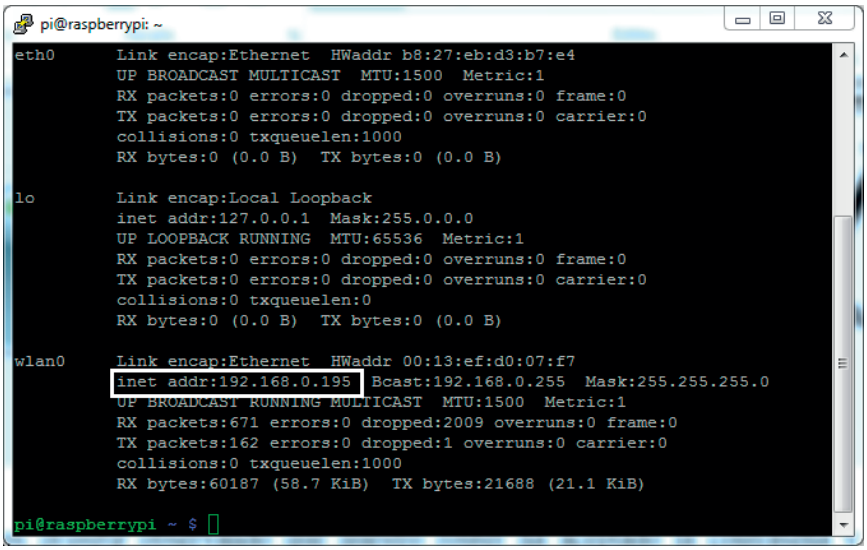


Figura 2.13

Ejecutamos **ifconfig** observando que nuestro router ha aceptado la contraseña y le ha asignado a nuestra Raspberry una dirección IP mediante el pincho wifi (figura 2.13).

Como última prueba ejecutamos desde el terminal el siguiente comando obteniendo respuesta desde el servidor.

```
ping google.com
```

Ahora tenemos conexión a Internet desde nuestra Pi2. Todo funciona perfectamente, así que ¡ya tenemos wifi en nuestra Raspberry Pi! ¡Adiós al cable Ethernet! (figura 2.14).

Por otra parte, si vamos a utilizar el entorno gráfico de la Raspberry, la configuración de la wifi es mucho más fácil. Para ello nos vamos al menú principal y navegamos hasta la opción de configuración wifi tal y como se muestra en la figura 2.15.

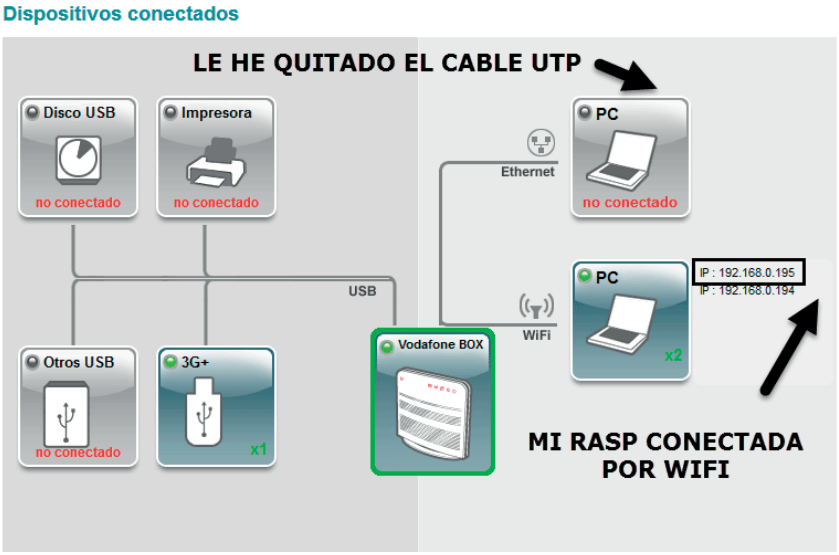


Figura 2.14

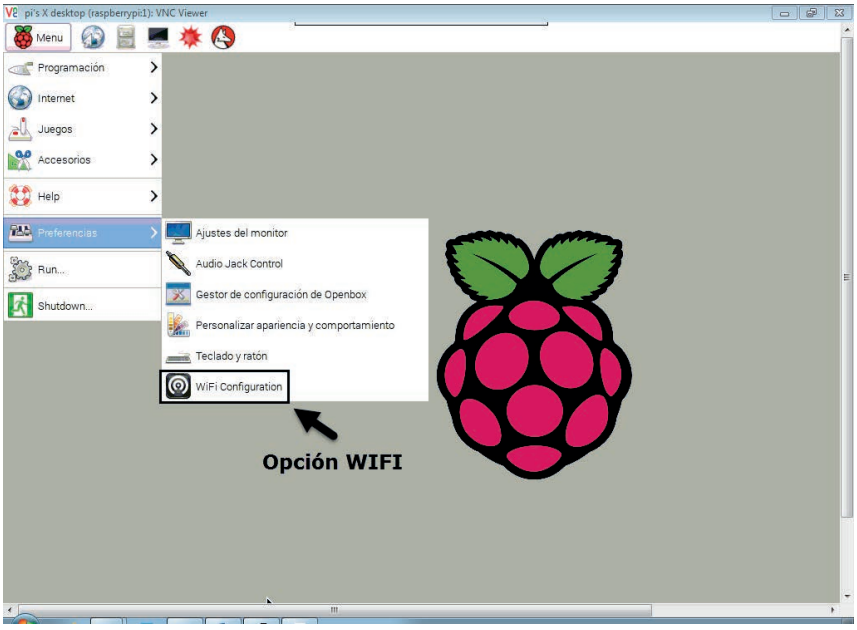


Figura 2.15

Dentro de la ventana de configuración (figura 2.16) realizamos un escaneo de redes inalámbricas con el botón Scan. A continuación seleccionamos el nombre o SSID de nuestro router (figura 2.17).

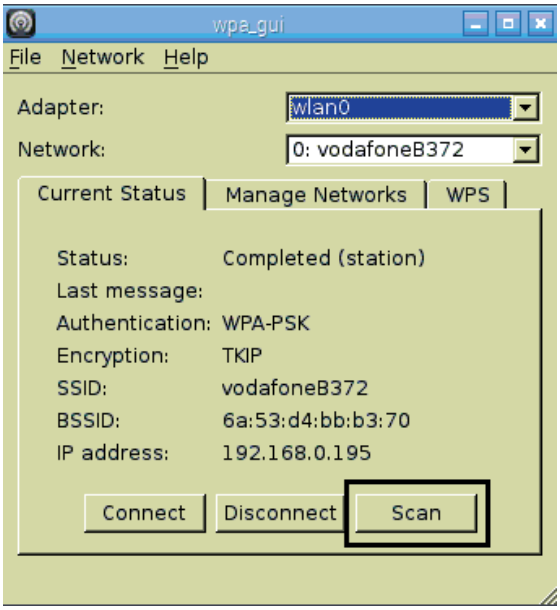


Figura 2.16

Figura 2.17

Por último, debemos editar la configuración del router para especificar su clave cifrada de conexión a Internet, como se observa en las figuras 2.18 y 2.19:

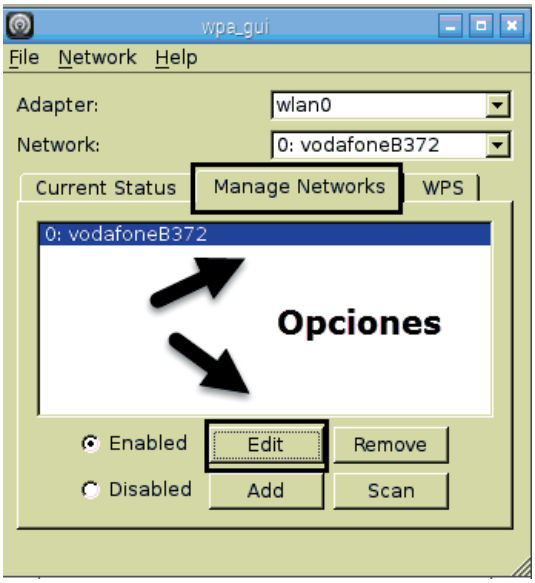


Figura 2.18

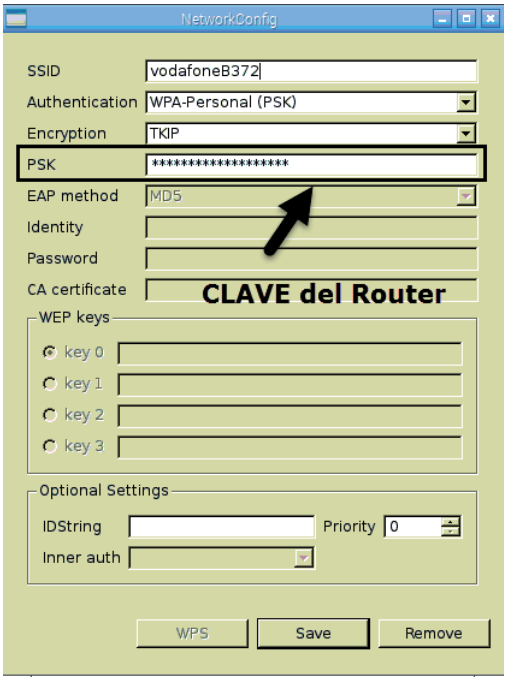


Figura 2.19

VncServer también está disponible para Linux. Podemos utilizar una aplicación llamada: **Remote Desktop Viewer**. Desde el centro de software de nuestro Linux (en este caso Ubuntu) tendremos este programa tal y como se observa en la figura 2.20:

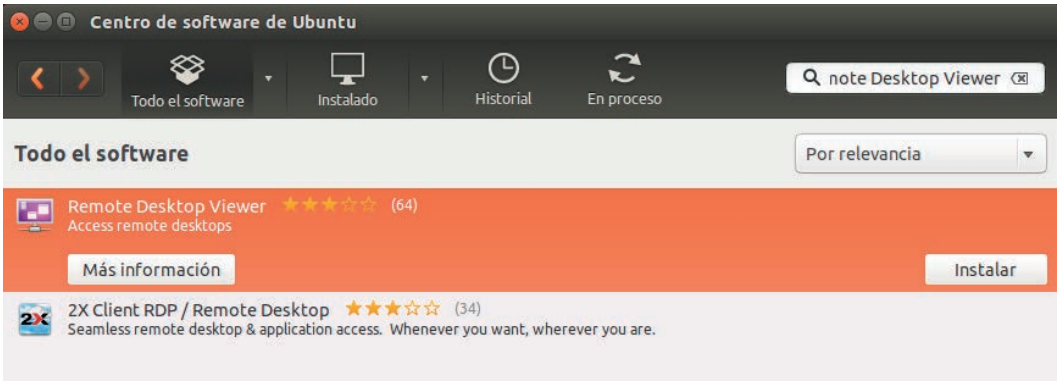


Figura 2.20

Tras instalar este software, accedemos a la ventana de conexión (figura 2.21). Es necesario asegurarse de que la Raspberry está corriendo el **vncserver**.

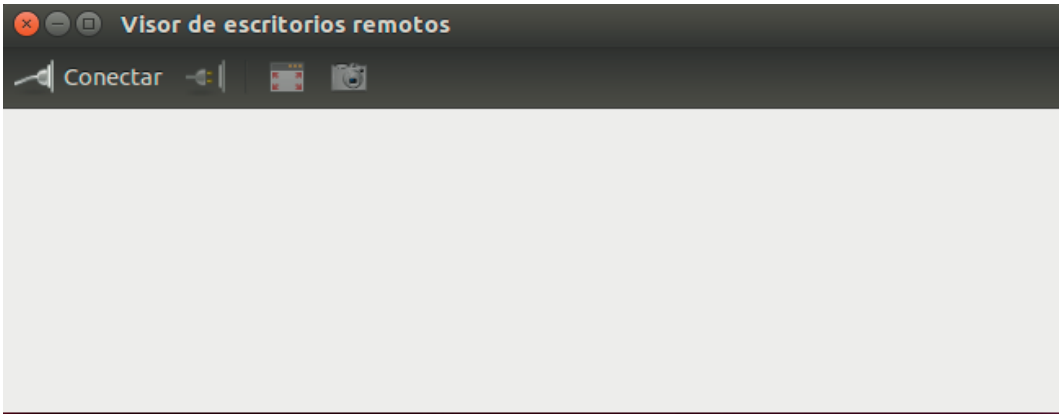


Figura 2.21

Bajo la ventana de conexión nueva que nos aparece, es imprescindible configurar el modo **VNC** en el protocolo de comunicación e incluir ":1" al final de la dirección IP de la Raspberry.

A continuación se nos pedirá la contraseña de conexión al servidor VNC y luego entraremos en el entorno gráfico tal como hicimos bajo Windows (figura 2.22).

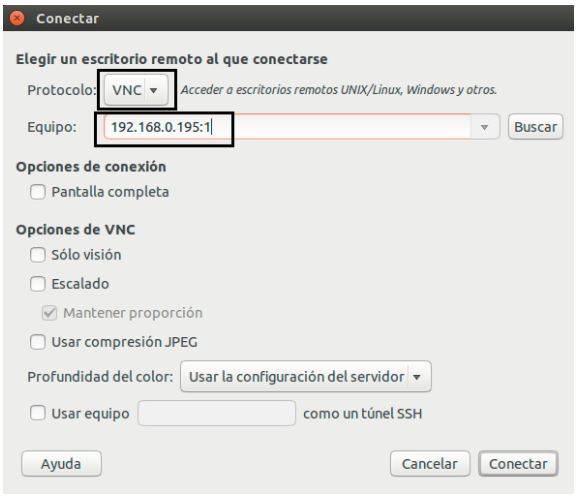


Figura 2.22

Uno de los retos a la hora de acceder a la Rasp de forma remota es que necesitamos conocer su dirección IP. Si tenemos la placa conectada a un teclado y a una pantalla, siempre podemos ejecutar el comando **ifconfig** para obtener esa información. Pero si vamos a utilizar la Rasp de forma independiente, debemos descubrir su IP mediante el uso de una aplicación de escaneo IP. Existen varias de estas disponibles gratuitamente bajo Windows. Una particularmente fácil de usar es **Advanced IP Scanner** (www.advanced-ip-scanner.com/es/). Tan pronto lo descarguemos y ejecutemos (es portable y por tanto no es necesario instalarla) aparece la siguiente pantalla tal y como se observa en la figura 2.23.

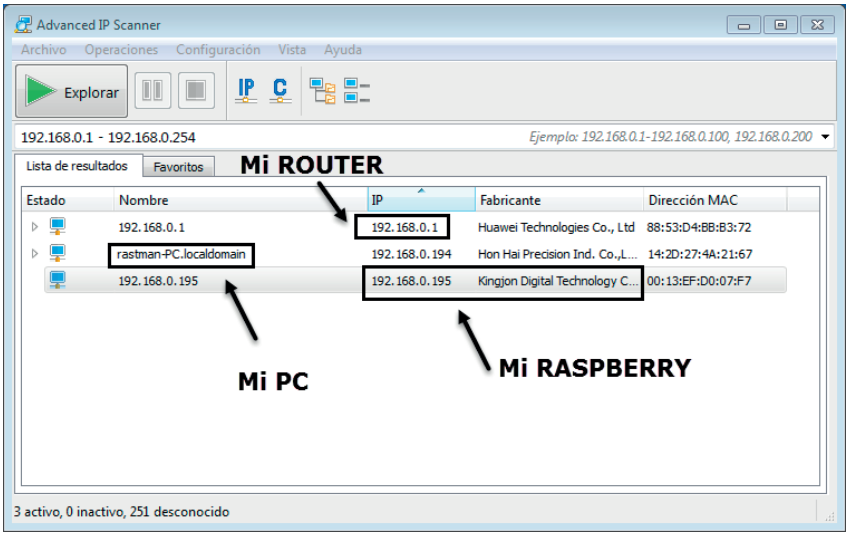


Figura 2.23

Exploramos los dispositivos conectados y podemos observar la IP de nuestra Raspberry fácilmente para poderla utilizar en un posible acceso. También podemos hacer esto en Linux utilizando la utilidad simple denominada: **Nmap**.

2.3 INSTALACIÓN Y USO DE LA DISTRIBUCIÓN UBUNTU MATE EN LA RASPBERRY Pi2

Desde mayo del 2015 se ha portado una interesante distribución de Ubuntu para nuestra Rasp 2. Se trata de la **Ubuntu Mate 15.04** basada en un entorno GNOME2 y, por lo tanto, con una carga de trabajo para el hardware bastante baja, propiciando que sea una buena opción para los tradicionales usuarios de Ubuntu que no desean cambiarse a la distribución Raspbian típica de la Raspberry Pi. (figura 2.24).

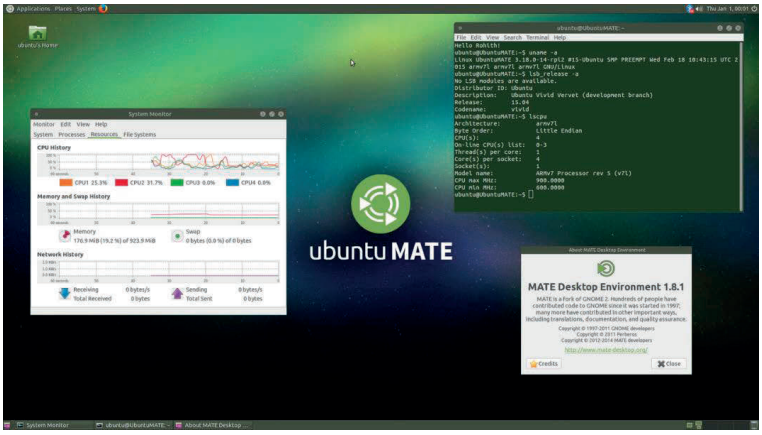


Figura 2.24

Tal ha sido el éxito de Mate que los equipos de Linux Mint y Ubuntu han tomado la decisión de apoyar su desarrollo.

Para descargar la imagen de esta Ubuntu Mate acudimos a la página oficial de Raspberry (www.Raspberry.org) y entramos en la sección de descargas, tal y como se muestra en la figura 2.25:

Figura 2.25

Una vez bajamos el fichero, lo descomprimimos, generando un archivo tipo imagen (extensión iso) que debemos grabar en la microSD de la Raspberry mediante la utilidad gratuita bajo Windows llamada: **Win32 Disk Imager.** A continuación la introducimos en nuestra Pi2 y arrancamos el sistema. Tras un cierto tiempo de instalación automática, dispondremos de nuestro nuevo y flamante escritorio Ubuntu en completo español. Evidentemente es un Linux completo que podremos personalizar a nuestro gusto. Para afinar el sistema es conveniente consultar el enlace: https://ubuntu-mate.org/raspberry-pi/ donde se recomiendan varios pasos para cambiar el tamaño del sistema de ficheros por defecto y ajustar la aceleración de vídeo entre otros.

Para acceder remotamente con una conexión SSH a la Rasp 2 es necesario instalar un servidor de SSH como es el llamado **openssh** utilizando la siguiente orden:

```
sudo apt-get install openssh server
```

Para arrancar el servidor de SSH:

```
sudo /etc/init.d/ssh start
```

Para parar el servidor SSH:

```
sudo /etc/init.d/ssh stop
```

En la configuración del servidor no hay que "tocar" nada. Tan solo nos resta utilizar la aplicación **PuTTY** en la parte del cliente (por ejemplo, Windows 7) para acceder remotamente a un terminal de la Pi2.

Por otra parte para acceder remotamente al escritorio del Ubuntu Mate es necesario seguir una serie de pasos diferentes a los que se expusieron para la distribución Raspbian.

Los pasos que seguiremos serán los siguientes:

1. Instalación en Ubuntu del soporte para el protocolo de escritorio remoto XRDP.
2. Instalación en Ubuntu del entorno gráfico **xfce4** (no funciona con Gnome).
3. Configuración de la sesión XRDP en Ubuntu.
4. Verificar desde Windows la conexión con Ubuntu mediante escritorio remoto.

El soporte para escritorio remoto puede instalarse desde los propios repositorios de Ubuntu tecleando en el terminal:

```
$ sudo apt-get install xrdp
```

El siguiente paso es instalar el escritorio **xfce4**. Esto es necesario ya que el escritorio por defecto de Ubuntu no funciona actualmente como escritorio remoto. El escritorio **xfce4** es un entorno ligero que se ajusta sin problemas a los que estamos acostumbrados a trabajar con Gnome. Nuevamente, **xfce4** está disponible desde los repositorios:

```
$ sudo apt-get install xfce4
```

Para que XRDP emplee este escritorio en las sesiones remotas hay que indicarlo en el archivo *~/.xsession*, en el que se define el entorno a usar por defecto. Esta configuración no implica que siempre que usemos la Raspberry en la que está instalada Ubuntu Mate se usará el **xfce4**, ya que en estos casos entramos usando el gestor de sesiones en el cual se indica qué escritorio se quiere usar. El archivo *.xsession* simplemente configura el entorno por defecto cuando no se loguea uno usando el gestor de sesiones.

```
$ echo xfce4-session > ~/.xsession
```

Ya solo queda reiniciar el servicio XRDP para que tome la nueva configuración y verificar que es posible conectarnos desde Windows.

```
$ sudo service xrdp restart
```

Para ello utilizaremos la aplicación **Remote Desktop Connection** incluida en Windows 7/8 y accesible desde el botón Inicio de Windows.

La ventana que se abre presenta la apariencia de la figura 2.26. En ella tecleamos la dirección IP de nuestra Pi2 tras lo cual se abre una ventana en la que se nos pide la autentificación como usuario de nuestra Raspberry (figura 2.27).

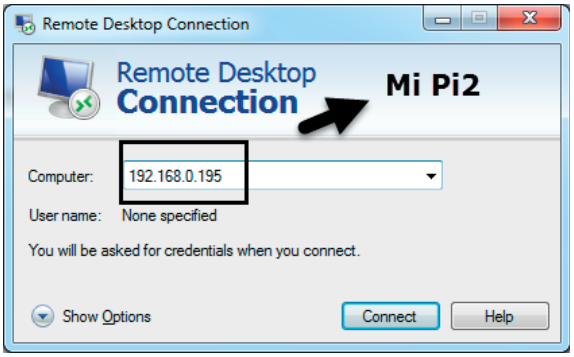


Figura 2.26

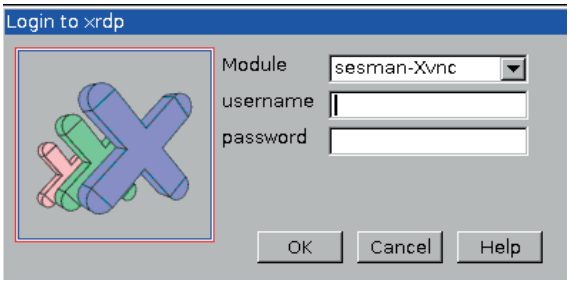


Figura 2.27

Ya estamos dentro y podemos observar todas las aplicaciones que estaban bajo Ubuntu Mate pero ahora ordenadas de distinta manera con este entorno gráfico más ligero (xfce4) y más apropiado para una conexión remota. En la figura 2.28 se puede observar el resultado en una máquina con Windows 7 con conexión remota a nuestra Pi2.

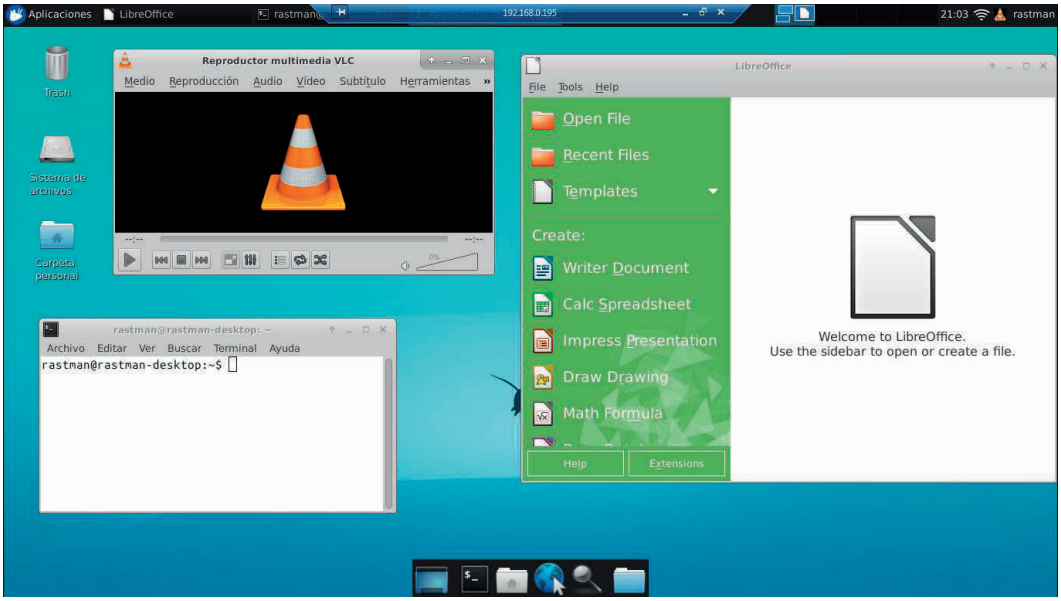


Figura 2.28

CAPÍTULO

3 PROGRAMANDO LA RASPBERRY Pi2

Hasta ahora hemos instalado el sistema operativo y configurado nuestra Raspberry Pi. A partir de este punto deseamos empezar a trabajar con ella. Casi siempre, esto requiere que seamos capaces de crear nuestros propios programas o editar los de otras personas.

En este capítulo se expondrá una breve introducción a la programación de nuestra Raspberry. Si bien es divertido construir hardware en nuestros proyectos electrónicos, no llegaremos muy lejos sin utilizar ciertos conocimientos básicos de programación. Este capítulo nos ayudará a introducir y presentar ciertos conceptos de edición y programación para que nos sea cómodo crear y depurar algunos de los programas que vamos a discutir en este libro. También aprenderemos cómo cambiar los programas existentes para realizar variaciones personales que mejoren lo expuesto.

En este capítulo vamos a cubrir los siguientes temas:

- ✓ Comandos básicos de Linux y navegación por el sistema de ficheros en la Raspberry Pi.
- ✓ Crear, editar y guardar archivos en la Raspberry Pi.
- ✓ Creación y ejecución de programas de lenguaje Pyhton.
- ✓ Parte de la programación básica que se construye sobre la Raspberry Pi, como el lenguaje de programación C.

Todas estas tareas las vamos a realizar desde una conexión SSH o utilizando VNC.

3.1 COMANDOS BÁSICOS DE LINUX

Linux es un proyecto de código abierto que fue fundado originalmente para crear un núcleo de libre disponibilidad para cualquier persona. El núcleo o kernel es el corazón de un sistema operativo y se ocupa de la comunicación entre el usuario y el hardware.

Aunque el término “Linux” es correctamente empleado para referirse al “núcleo”, es a menudo utilizado para referirse a una colección de diferentes proyectos (“sabores”) de código abierto de distintas compañías. La versión original de Linux fue combinada con una colección de herramientas creadas por un grupo denominado GNU. El sistema resultante fue conocido como GNU/Linux y era muy básico pero potente. A diferencia de los otros sistemas operativos de la época, este ofrecía facilidades como: múltiples cuentas de usuario, en donde varios usuarios podían compartir una misma máquina.

Antes de empezar, es recomendable echar un vistazo a algunos de los términos y conceptos utilizados en el mundo Linux, como los que se muestran en la tabla 3.1. No te preocupes si no entiendes alguna de estas definiciones; la irás comprendiendo en la medida que profundicemos poco a poco en este terreno.

Función	Descripción
Bash	El shell o intérprete de comandos más popular, utilizado en la mayoría de las distribuciones Linux.
Terminal o consola	Ventana en modo texto en la que introducimos comandos u órdenes al Linux.
GUI	Interfaz gráfica de usuario (el típico escritorio de Windows).
Entorno de escritorio	GNOME y KDE son los entornos de escritorio más populares para Linux. LXDE es el escritorio de la distribución Raspbian.
Directorio	El término Linux para lo que Windows denomina *carpetas*, en donde los archivos se almacenan.
Distribución	Una versión de Linux concreta. Raspbian y Openlec son distribuciones posibles para la Raspberry.
EXT2/3/4	El sistema de archivos EXTendido, el formato más común utilizado en Linux.
Paquete	Una colección de archivos requeridos para la ejecución de una aplicación, comúnmente administrados por el gestor de paquetes.
Gestor de paquetes	Una herramienta que sigue la pista del software ya instalado y de las nuevas instalaciones o actualizaciones.
Root	La cuenta de usuario principal en Linux, equivale a la cuenta de *administrador* en Windows. También denominada *superusuario.*
Shell	Un intérprete de comandos basado en texto, cargado dentro de una terminal.
sudo	Un programa que permite a los usuarios limitados o restringidos ejecutar un comando como el usuario root.

Tabla 3.1

En Linux existen normalmente dos principales formas de llevar a cabo una determinada tarea: a través de la interfaz gráfica de usuario (GUI) y a través de la línea de comandos (conocida en la jerga Linux como la consola o la terminal).

La apariencia de las distintas distribuciones Linux pueden ser completamente diferentes, dependiendo del entorno de escritorio que utilicen. En este libro, la distribución que recomendamos utilizar es la Raspbian o la Ubuntu Mate, pero la mayoría de los comandos que vas a aprender aquí son introducidos desde la terminal y casi siempre son los mismos para todas las distribuciones.

En este capítulo empezaremos a trabajar con el sistema operativo de nuestra Raspberry. Lo vamos a poder hacer desde estas dos últimas perspectivas:

- ✓ Utilizando un sencillo y preinstalado escritorio “tipo Windows” denominado LXDE.
- ✓ Usando el potente pero más complejo modo terminal a base de comandos tipo MS-DOS.

Pero antes de nada vamos a introducirnos en el sistema operativo Raspbian que llevará nuestra Raspberry y que será el que utilicemos a lo largo de todo el libro.

Raspbian es el nombre dado a una variante personalizada de la popular distribución Debian Linux. Debian es una de las distribuciones Linux más antiguas y se enfoca en la alta compatibilidad y un rendimiento excelente incluso sobre el hardware modesto, convirtiéndose en un gran socio para la Raspberry Pi. Raspbian toma a Debian como su base o distribución padre y agrega herramientas y software para hacer que la Pi2 sea lo más fácil de utilizar.

Para mantener el tamaño del archivo de descarga al mínimo posible, la imagen de la Raspberry Pi para Raspbian solo incluye un subconjunto del software que encontraría en una versión normal de escritorio. Esta imagen incluye herramientas para navegar en la web, programar en Python y utilizar la Raspberry Pi2 con una GUI. El software adicional puede instalarse rápidamente utilizando el gestor de paquetes **apt** de la distribución, o adquiriéndolo a través de la Raspberry Pi Store con el enlace sobre el escritorio. Raspbian incluye un entorno de escritorio conocido como **LXDE** (Lightweight X11 Desktop Environment). Diseñado para ofrecer una interfaz de usuario atractiva utilizando el software del Sistema de Ventanas X (X Windows), LXDE ofrece una interfaz familiar, la cual puede ser rápidamente accesible por cualquiera persona que haya utilizado en el pasado Windows.

Una vez que accedemos a la Raspberry mediante VNC con el nombre de usuario y contraseña apropiados observamos el escritorio o interfaz gráfico mencionado anteriormente (figura 3.1).

Figura 3.1

Dentro de éste entorno pinchamos en el icono **LXTerminal** que nos abrirá un terminal tipo MS-DOS de igual manera que si hubiésemos accedido mediante SSH utilizando el programa **putty.exe** visto en el capítulo anterior. Son dos caminos para llegar al mismo objetivo: Disponer de una consola de comandos para trabajar con el **shell** de Linux (figura 3.2).

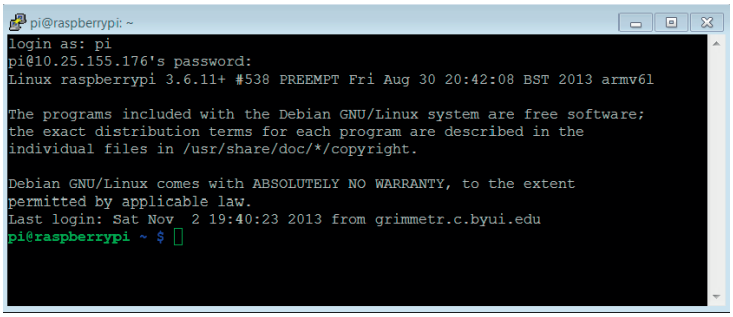


Figura 3.2

Observamos como el cursor está en el símbolo del sistema. A diferencia de Microsoft Windows o Apple OS, en Linux, la mayor parte de nuestro trabajo se realizará por medio de comandos que escribiremos en la línea de comandos. Así que vamos a tratar algunos comandos. Si ahora tecleamos el comando: **ls** veremos cómo se listan por pantalla todos los ficheros y subdirectorios que contiene el presente directorio apreciando su tipo por su diferente color (figura 3.3).

Figura 3.3

Podemos navegar a través de la estructura de directorios con el comando **cd** (cambio de directorio). Por ejemplo, si deseamos ver el contenido del directorio: python_games tecleamos el comando: **cd python_games** seguido del comando **ls** observando el resultado en la figura 3.4:

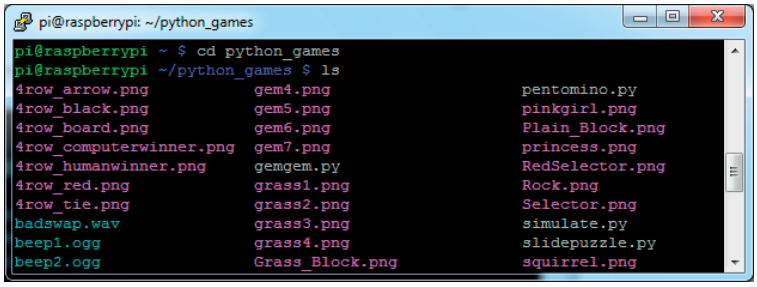


Figura 3.4

Ahora se debe aclarar que podemos utilizar un atajo y escribir **cd ./python_games**. El punto (.) es un carácter de acceso directo para el directorio predeterminado. El comando cd es la abreviatura del cambio de directorio. También se pudo haber escrito **cd/home/pi/python_games** recibiendo el mismo resultado por pantalla. Esto se debe a que estamos en el directorio **/home/pi** que es al cual accedemos por defecto cuando iniciamos una sesión en el sistema Linux Raspbian. Si se desea saber en cualquier momento en que directorio estamos, simplemente utilicemos el comando **pwd** recibiendo lo que se muestra en la figura 3.5:

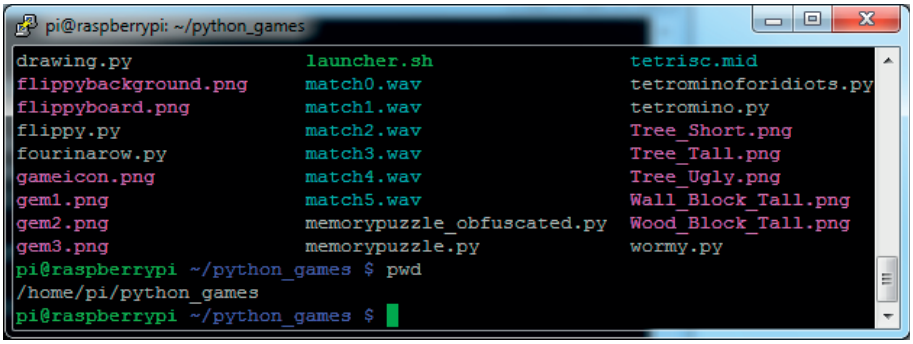


Figura 3.5

El resultado de ejecutar el comando **pwd** es **/home/pi/python_games**. Ahora, podemos utilizar dos diferentes atajos para navegar de vuelta al directorio predeterminado. La primera es que escribir **cd..** en el terminal; esto nos llevará al directorio justo encima del directorio actual en la jerarquía. A continuación, escribimos de nuevo **pwd** y deberíamos ver la siguiente pantalla reflejada en la figura 3.6:

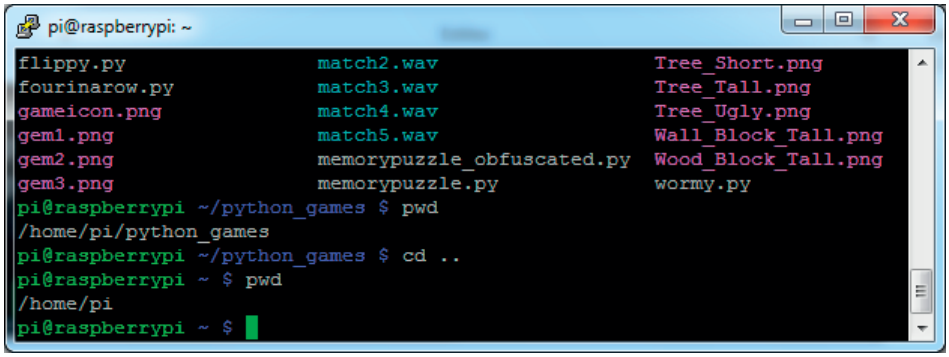


Figura 3.6

La otra manera de volver al directorio principal es escribiendo **cd ~** . Siempre regresamos al directorio inicial o **home**. (figura 3.7). Otra forma de ir a un archivo especifico es utiliza su ruta completa. En este caso, si se desea ir al directorio **/home/Raspbian/Desktop** desde cualquier sitio, simplemente escribimos **cd/home/Raspbian/Desktop**. Existen una amplia gama de comandos que puede ser interesante resumir en la tabla 3.2.

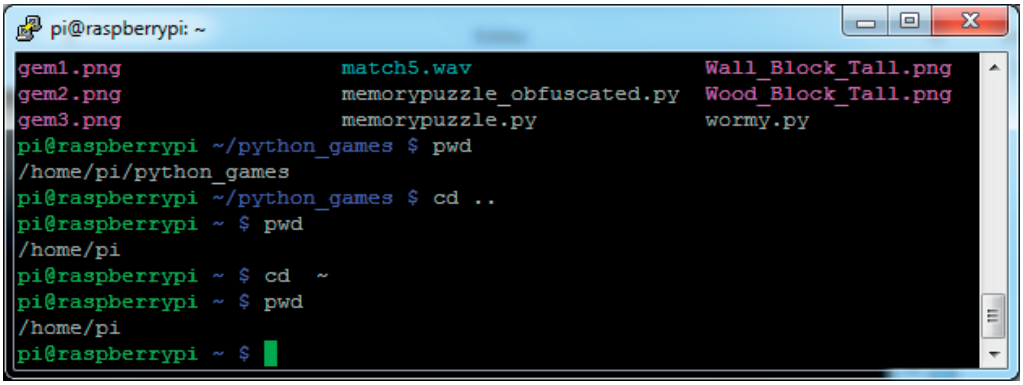


Figura 3.7

Comando	Descripción
ls	Lista todos los subdirectorios y archivos que contiene el actual directorio.
rm archivo	Borra el archivo especificado.
mv archivo1 archivo2	Renombra el archivo1 con el nombre de archivo2.
cp archivo1 archivo2	Copia el archivo1 al archivo2.
Mkdir nombdirectorio	Crea un nuevo directorio con el nombre: nombdirectorio.
clear	Limpia la pantalla del terminal.

Tabla 3.2

Ahora podemos jugar un poco con los comandos e investigar los directorios y archivos contenidos en la distribución Raspbian. Pero con cuidado, ya que Linux no es como Windows y las acciones de borrado de ficheros o directorios no presentan ventana de confirmación.

3.2 CREAR, EDITAR Y GUARDAR ARCHIVOS EN LA RASPBERRY Pi2

Ahora que podemos acceder y movernos fácilmente entre directorios y ver qué archivos contienen, abordaremos la tarea de editar dichos archivos. Para ello se necesita un software que nos permita editar texto. Bajo Linux podemos utilizar un programa editor llamado **emacs**. Otras alternativas son editores como **nano**, **vi**, **vim** o **gedit**. Para utilizar, por ejemplo, un editor de textos muy sencillo llamado nano que viene por defecto instalado con nuestra Raspbian. Simplemente tecleamos: **nano nombrefichero**. Si no existe dicho fichero, se creará. La figura 3.8 muestra lo que se verá si se escribe **nano ejemplo.py**.

A diferencia de Windows en Linux no se asignan automáticamente las extensiones a los ficheros. Depende de nosotros especificar el tipo de archivo que deseamos crear. Por otro lado, si estamos ejecutando nano desde el entorno gráfico podemos utilizar el ratón; si lo hacemos desde el terminal o consola no tendremos el puntero disponible por lo que necesitaremos las teclas o una combinación de ellas. Por ejemplo, para guardar debemos pulsar CONTROL+X. Este tipo de editor es

apropiado, como su nombre indica (nano), para realizar pequeñas tareas de texto como la edición de ficheros de configuración del sistema y similares. Por su carencia de funciones de edición y procesado de texto, no es conveniente usarlo para construir programas o como procesador de textos tipo Word.

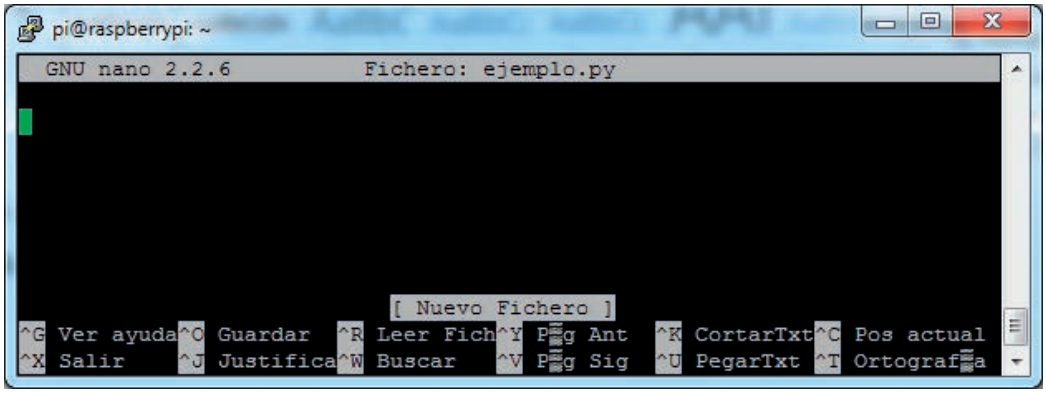


Figura 3.8

Por otra parte, la distribución Raspbian trae un editor apropiado para trabajar con nuestro lenguaje de programación favorito con la Raspberry Pi, que va a ser a partir de este momento, el archiconocido y estrella del momento entre los programadores: **PYTHON**.

3.3 CREACIÓN Y EJECUCIÓN DE LOS PROGRAMAS EN PYTHON CON LA Pi2

Ahora que estamos listos para comenzar la programación, tendremos que elegir un lenguaje. Existen muchas posibilidades: C, C++, Java, Python, Perl, etc. Nosotros vamos a utilizar **Python** por tres razones principales:

- ✓ Es un lenguaje sencillo y muy intuitivo.
- ✓ Existe mucha documentación al respecto.
- ✓ Está de moda hoy en día entre los programadores.

Para trabajar con el lenguaje Python, podemos utilizar cualquier editor de textos; por ejemplo, **nano**. Para ejecutarlos, tecleamos en el terminal **python nombreprgrama.py** (figura 3.9).

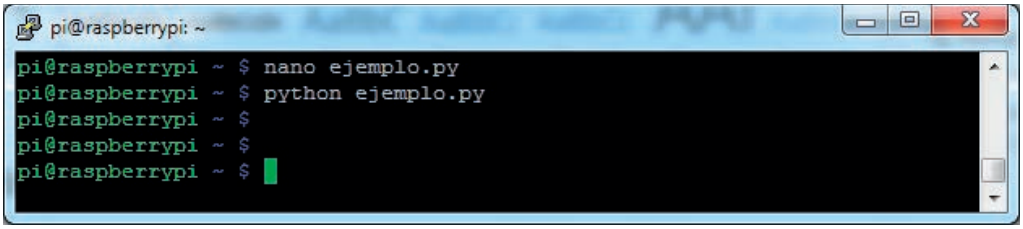


Figura 3.9

Como apuntamos en el apartado anterior, Raspbian trae por defecto un editor especializado para trabajar con Python denominado **IDLE**. Con este entorno podemos crear, editar, guardar y ejecutar nuestros propios programas. Realmente IDLE es un intérprete de programación; es decir, no vamos a obtener un compilación propiamente dicha, pero sí nos va a resultar muy útil para aprender este lenguaje y crear interesantes aplicaciones en el mundo de la electrónica con la Raspberry.

Para acceder a IDLE debemos arrancar el entorno gráfico remotamente. Ya se expuso anteriormente cómo instalar y ejecutar un servidor gráfico, por lo que lo único que tenemos que hacer es ejecutar un cliente remoto gráfico como el **VNC Viewer** (figura 3.10):

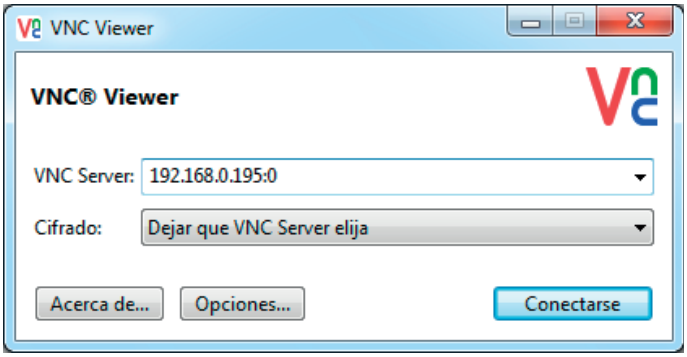


Figura 3.10

Tras establecer la conexión, observamos el escritorio remotamente y nos fijaremos que bajo la pestaña de programación, en el menú principal, disponemos de dos opciones para trabajar con Python. Escogeremos la versión de Python 2 (figura 3.11) por ser la más compatible.

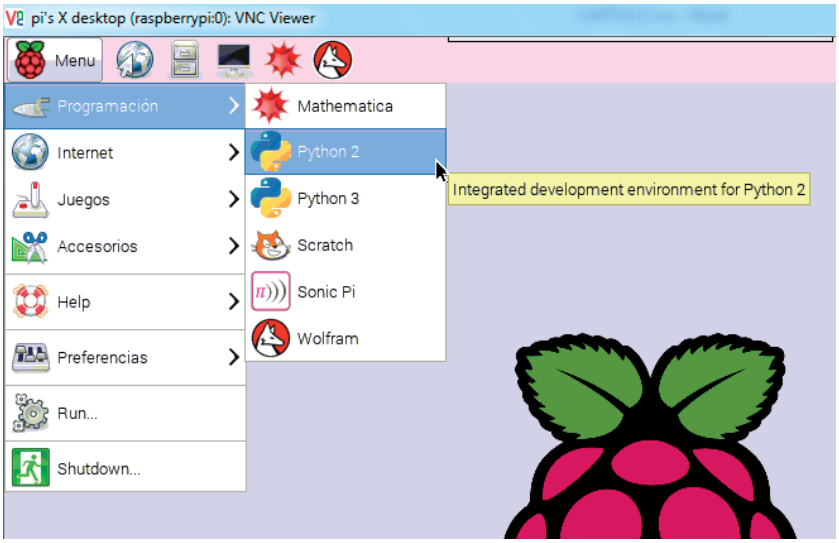


Figura 3.11

El entorno IDLE que se presenta podemos interactuar de manera básica, escribiendo, por ejemplo, una simple orden: **print 'Hola mundo'**. Observaremos el resultado directamente en pantalla (figura 3.12). Es decir, podemos lanzar simples líneas de código y obtener fácilmente el resultado. Esta función es muy interesante para testear y probar lo más básico de Python antes de acometer tareas más complejas que impliquen muchas líneas de programación.

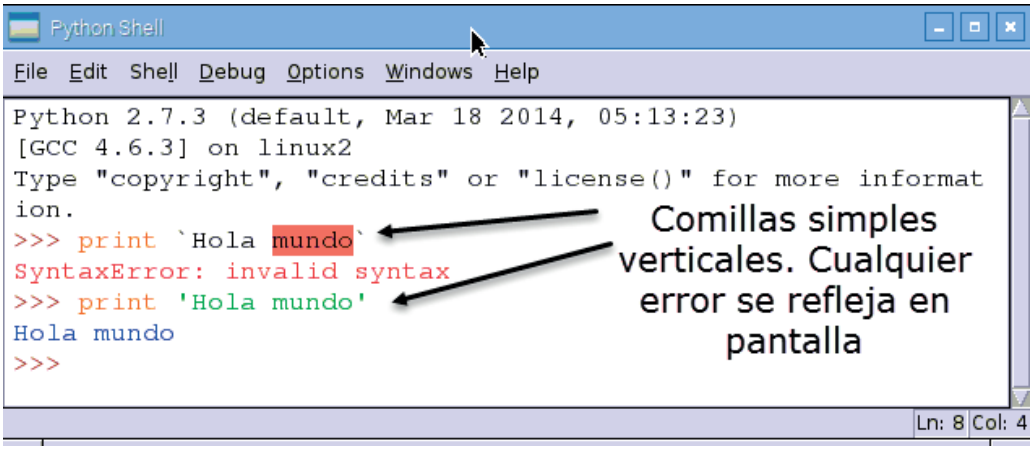


Figura 3.12

Para mayor claridad, Python colorea automáticamente el texto mientras escribes. Las instrucciones aparecen en naranja, y el resultado o salida en azul y los mensajes de error en color rojo. De todas maneras todo esto e incluso la fuente y tamaño del texto se pueden configurar posteriormente desde el entorno.

3.3.1 Trabajando con números

Podemos utilizar Python para realizar cálculos aritméticos. Como la mayoría de los lenguajes de programación, Python trata números enteros, de punto flotante o con decimales. Sin embargo, el resultado de una operación con dos números enteros siempre producirá un número entero. Esto hay que tenerlo en cuenta en la operación de división ya que se nos redondea el resultado. Para obtener el resultado correcto debemos considerar los dos números con decimales simplemente añadiendo 0.0. En la figura 3.13 se pueden observar diferentes operaciones aritméticas destacando la operación módulo (%) que obtiene solo el entero de una operación de división despreciando el resto.

3.3.2 Creando variables

Las variables sirven para almacenar números, texto y otra información relevante. Una variable es una forma sencilla de almacenar información de cualquier tipo. Para crear una variable, es necesario en primer lugar, asignarle un nombre que comience por lo menos por una letra (los nombres de variables no pueden comenzar con un número). A continuación se teclea el operador de asignación = y después se escribe el valor que debe tomar dicha variable. En la figura 3.14 se observan distintas operaciones con variables y de qué manera es sencillo reasignarles un valor. La verdad es que es como jugar con una calculadora.

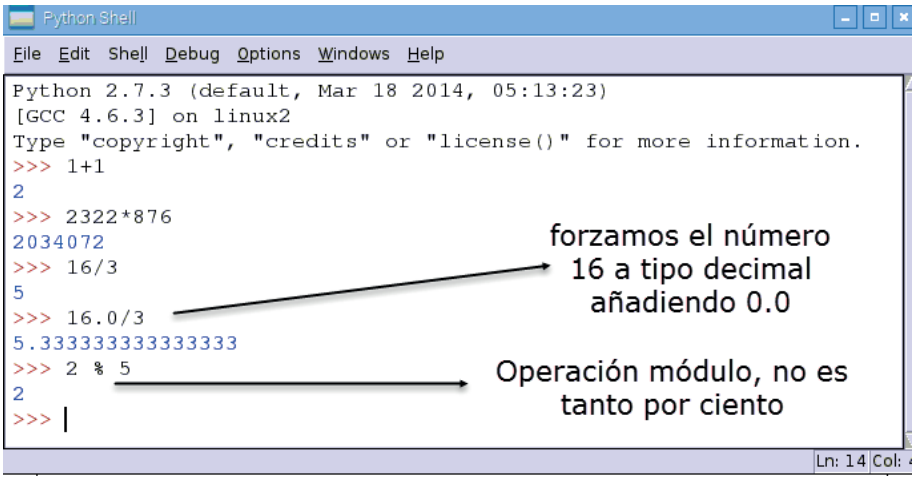


Figura 3.13

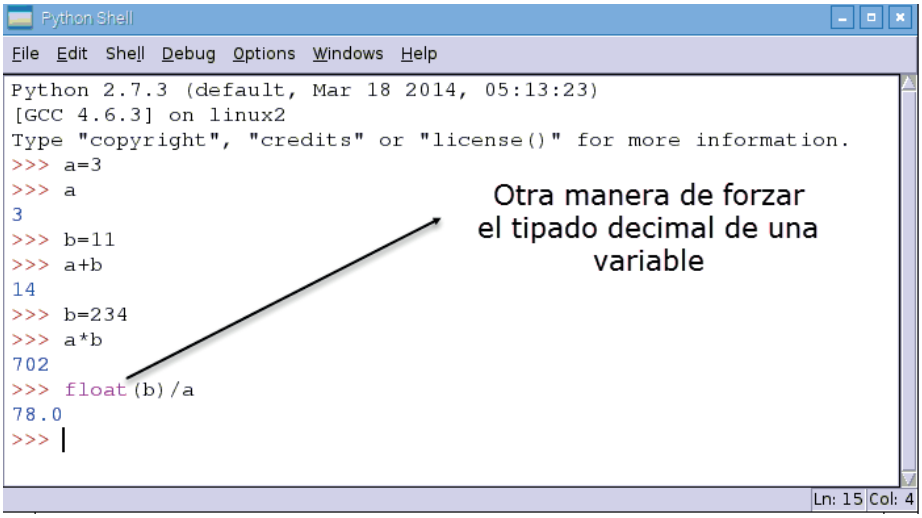


Figura 3.14

Por si no lo has notado, en Pyhton no hay que decidir de antemano el tipo de variable que vamos a crear. Este procedimiento inicial, que es común en otros lenguajes de programación como C o Java, no afecta a Python. De esta manera no tenemos que preocuparnos por el tipo de variable que vamos a utilizar, ya que Python lo hace por nosotros. Esto es una de las razones por las que empieza a ser muy popular en los colegios e institutos cuando se abordan temas relacionados con la programación.

3.3.3 Comenzando con cadenas

Python utiliza variables de cadenas (*strings*) para almacenar y manipular texto. Las cadenas se pueden dividir, unir, convertir en números, etc. Tenemos nuestra disposición de funciones para operar con cadenas, de ahí que le prestemos un poco de atención más detallada.

Como podemos ver en la figura 3.15, la operación de suma o concatenación de cadenas es sumamente fácil de llevar a cabo. Las otras operaciones aritméticas, como la resta, el producto o la división, no se podrán realizar de esta manera tan sencilla. La función print para mostrar cadenas la utilizaremos a menudo cuando enviemos texto a un LCD desde nuestra Pi2.

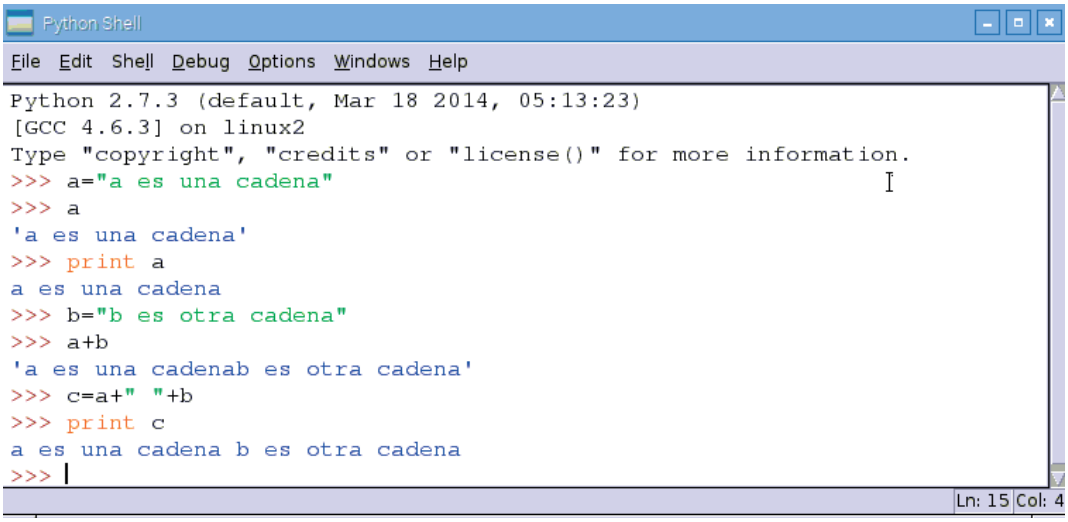


Figura 3.15

Las cadenas se pueden dividir en cadenas más pequeñas. En Python las cadenas se almacenan como una lista de caracteres: letras, espacios, puntuación y otros símbolos. Tenemos la posibilidad de contarlos, seleccionar un rango de caracteres según un patrón determinado, etc.

Además, podemos escoger palabras que estén fuera de las cadenas de dos maneras distintas. Una forma sería la de buscar una cadena de una secuencia de caracteres según su índice. También podríamos dividir una cadena en una lista (una colección de elementos dispuestos en orden) y seleccionar elementos de esa lista.

Para buscar una cadena, utilizamos la función de búsqueda **string.find** (‘*cadena a encontrar*’) retornando el índice al principio de la cadena; si lo encuentra, o -1 si no lo hace. Esto se puede observar en la figura 3.16. Para dividir una cadena, podemos utilizar la función **string.split (' ')**. El resultado es una lista de elementos. Seleccionaremos los elementos mediante un índice. Vemos un ejemplo de este método en la figura 3.17.

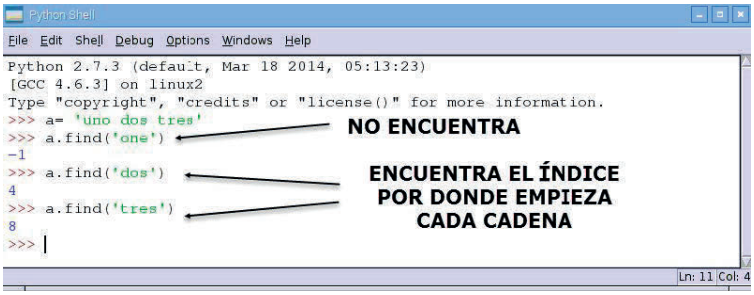


Figura 3.16

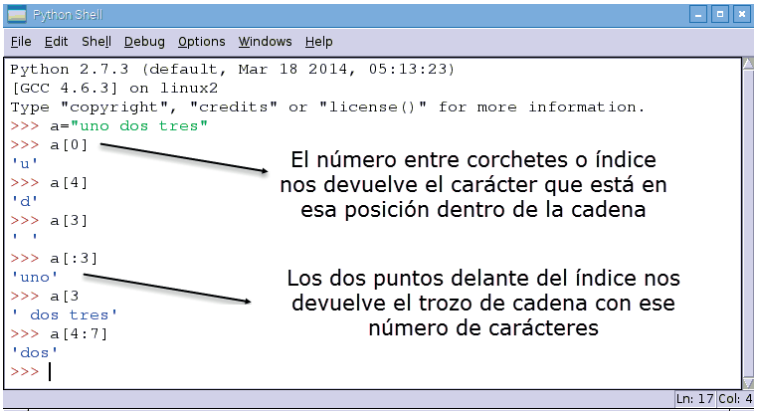


Figura 3.17

En la figura 3.18 vemos que el intérprete de Python visualiza un error ya que la lista de elementos **b** solo contiene tres elementos(0-2) por lo que a pedirle un cuarto elemento muestra un error en color rojo.

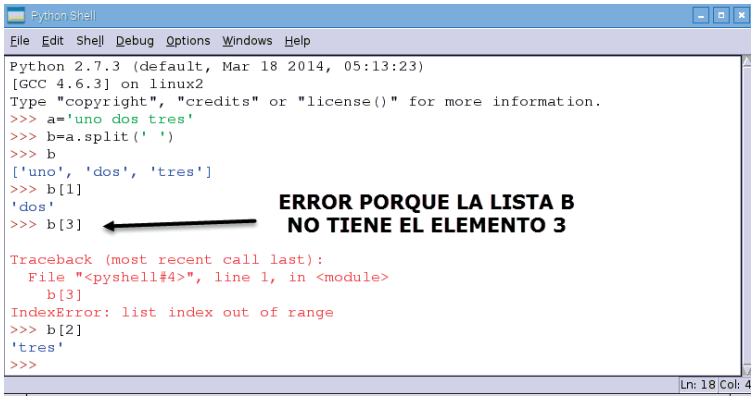


Figura 3.18

Además de todo lo expuesto para trabajar con cadenas, también podemos convertir cadenas a números y viceversa. Por ejemplo, podemos convertir una cadena como '1000' en el número 1000. Esta característica solo funciona si una cadena contiene números, y no para texto puro como "mil".

Es posible utilizar cadenas como plantillas para controlar cómo se muestran los números. Una cadena de plantilla controla cuántos dígitos decimales de un número se deben mostrar. Un ejemplo básico de este tipo de conversiones se puede observar en la figura 3.19:

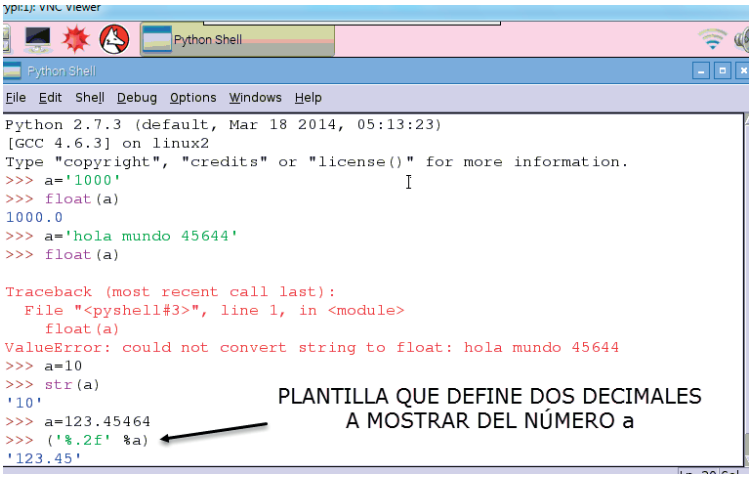


Figura 3.19

3.3.4 Trabajando con ficheros

Los archivos sirven para guardar información y recuperarla de nuevo en cualquier momento. Un archivo contiene información de forma permanente. Si no se guardan las variables en un archivo antes de salir de una aplicación, la información se pierde por completo y es imposible recuperarla.

Para utilizar un archivo se debe abrir, acceder al mismo y, a continuación, cerrarlo para fijar la información. También podemos escribir o guardar texto y números a través de la función **str()**; añadir más información al final del fichero para actualizarlo, etc.

En la figura 3.20 se muestran algunas de estas operaciones básicas:

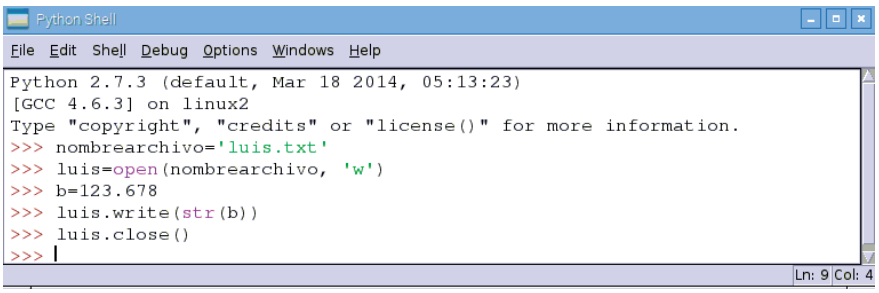


Figura 3.20

Realmente lo que hacemos es crear un fichero nuevo ('w') llamado *luis.txt* y escribir dentro del mismo un número convertido a cadena de caracteres mediante la variable **b**. Finalmente cerramos este fichero de texto. Lo podemos ver en el directorio **/home/pi directory** e incluso editar para ver que su contenido es la cadena: 123.678.

En la figura 3.21 se muestra cómo abrir el fichero anterior *luis.txt* para realizar una operación de lectura ('r') y volcar el contenido de este a través de una variable **b**.

```
>>> luis=open(nombrearchivo, 'r')
>>> b=luis.read()
>>> b
'123.678'
>>> luis.close()
>>>
```

Figura 3.21

3.3.5 Crear y ejecutar un script de Python

Podemos usar el entorno IDLE para crear, ejecutar y guardar una secuencia de comandos de Python. Cuando se crea un script, podemos guardarlo en un archivo con la extensión **.py**. Ello nos permite cargar de nuevo en IDLE para editarlo o ejecutarlo otra vez. También es posible ejecutar el script desde la línea de comandos de Linux y ver que los resultados aparecen después del símbolo del sistema. En este pequeño ejemplo vamos a crear un script sencillo que cuente el número de veces que lo ejecutemos. Se guarda el conteo en un fichero y se visualiza mediante la función **print**.

Creamos un fichero de texto (*contaje.txt*) con el editor simple **nano** y le insertamos el número cero como inicio de la cuenta. Guardamos y salimos al sistema (figura 3.22).

```
nano contaje.txt
```

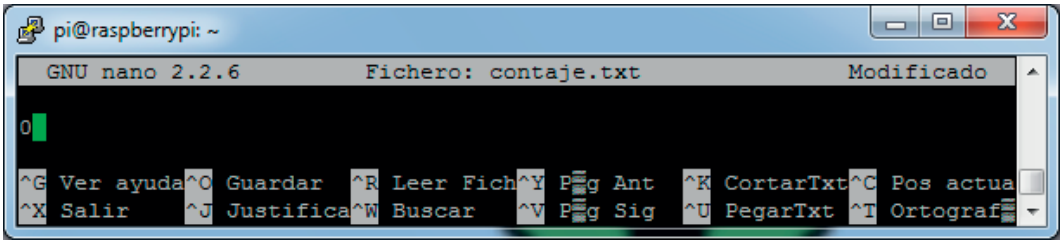


Figura 3.22

Ahora vamos a crear el script en Python llamado **cuenta.py**. Desde la ventana de su intérprete accedemos a la opción **File** del menú superior y desde allí pinchamos en **New Window** (figura 3.23).

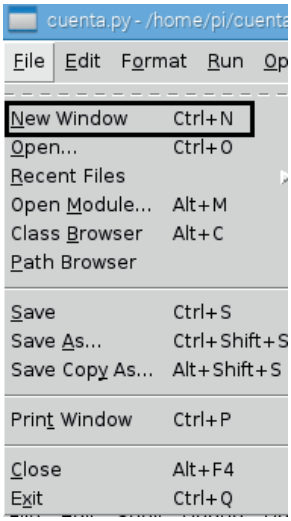


Figura 3.23

Dentro de esa nueva ventana que hemos abierto, tecleamos el código que se muestra en la figura 3.24. El script lo que hace básicamente es abrir un fichero (*contaje.txt*) cada vez que se ejecuta y lee su contenido, que es un número (**nveces**) que representa las veces que se ha ejecutado nuestro script. Actualiza el contaje (**nveces+1**) y guarda o escribe ese número como cadena (**str**) dentro del fichero *contaje.txt*. Por otra parte al abrir un archivo con '**r+**' se puede leer y escribir en él. Pyhton no elimina los contenidos existentes y también posiciona el puntero de escritura al final del archivo. Así que si escribimos un nuevo número, Python, desgraciadamente en nuestro caso, lo escribiría a continuación del anterior. Mediante la función **seek(0)** el puntero de escritura se mueve al inicio del archivo, de tal manera que todos los números añadidos se sobrescribirán en la misma posición permitiendo tener un solo número de contaje actualizado. Para ejecutar el script, seleccionamos la opción **run module** bajo la opción **run** del menú principal (figura 3.25). En el shell de Python observamos cómo cada vez que corramos el script nos va mostrando el número de veces que lo hace (figura 3.26).

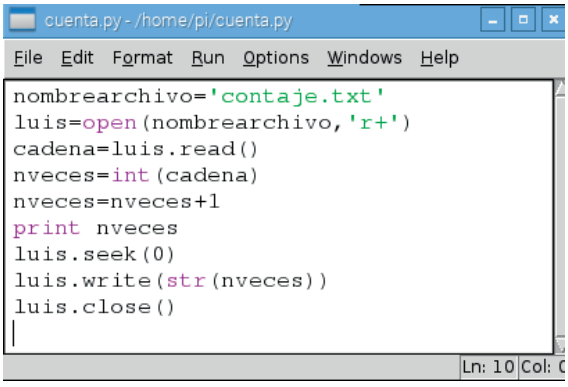
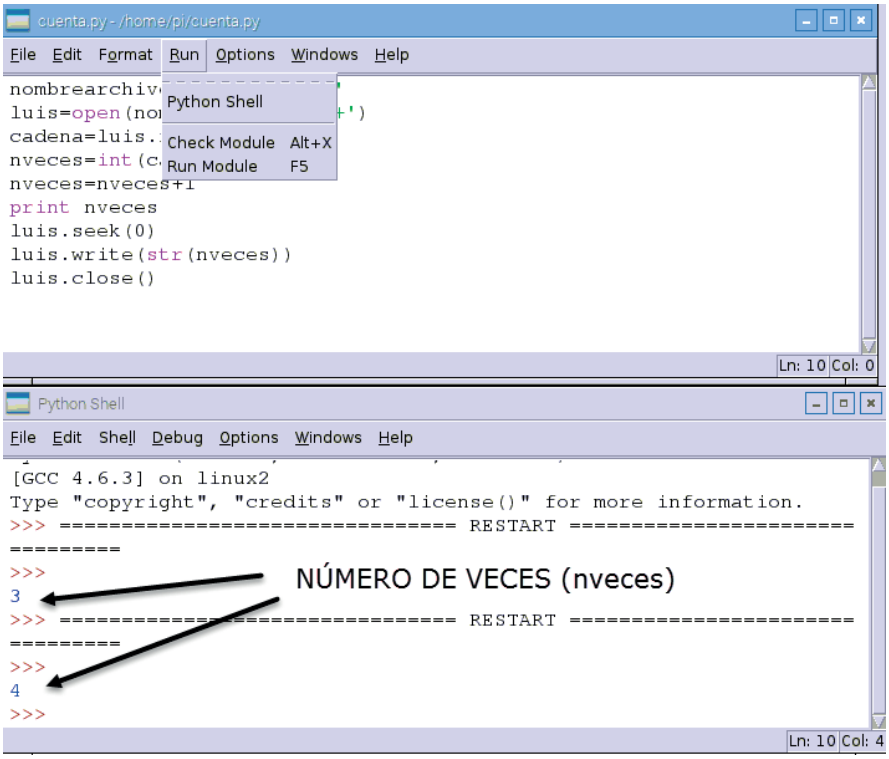


Figura 3.24



Figuras 3.25 y 3.26

3.3.6 Comenzando con las listas

Una lista sirve para recopilar información de más de un tipo. Las listas pueden contener números, cadenas y otros tipos de datos más complejos que veremos más adelante. Seleccionaremos un elemento por su índice o por su valor. Para crear una lista, colocaremos los elementos entre corchetes " [ ] " y pondremos comas entre cada elemento. Las listas son "mutables", lo que significa que pueden cambiar sus elementos e incluso realizar una operación aritmética en un elemento de la lista sin cambiar los otros. En la figura 3.27 se observa la creación de una lista simple:

```
>>> listaluis[0]
1
>>> listaluis[4]
'uno'
>>> listaluis[1] = listaluis[1] + 10
>>> listaluis
[1, 12, 3, 4, 'uno', 'dos', 'tres', 'cuatro']
>>>
```

Ln: 16 Col: 4

Figura 3.27

En la figura 3.28 se muestra la manera tan sencilla de acceder a un elemento concreto de la lista anteriormente creada, e incluso cómo realizar una operación aritmética básica.

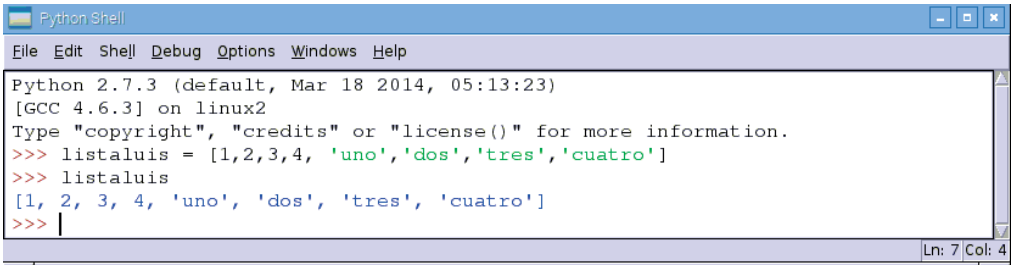


Figura 3.28

Como vemos, el elemento 1 ha cambiado de valer 2 a valer 12 (10+2). Incluso podemos concatenar dos listas mediante la operación suma. Los métodos de la lista sirven para trabajar con listas en formas más complejas. Los métodos constituyen una pequeña biblioteca de herramientas integradas de procesamiento de lista. Las podemos utilizar para contar los elementos de la lista, unir listas y modificarlas de varias maneras útiles. Los métodos de lista te permitirán ahorrarte tiempo en el diseño de un programa o script en Python. Por ejemplo, para añadir un nuevo elemento al final de una lista utilizaremos el método denominado **append()** como se muestra en la figura 3.29.

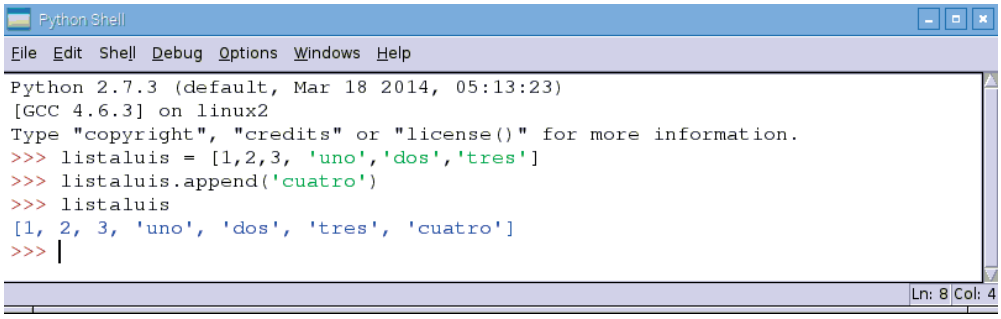


Figura 3.29

El método **insert()** inserta un elemento en una posición determinada de lista como se observa en la figura 3.30.

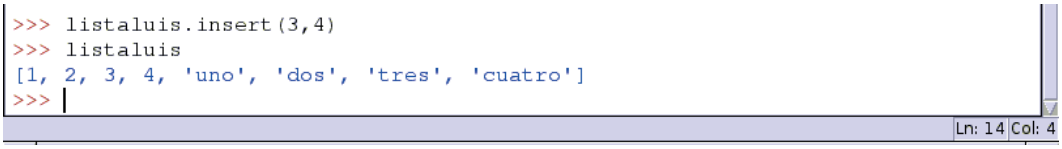


Figura 3.30

Para remover un elemento de una lista basta con utilizar el método **remove()**. Un ejemplo de ello se puede observar en la figura 3.31.

```
>>> listaluis.insert(3,4)
>>> listaluis
[1, 2, 3, 4, 'uno', 'dos', 'tres', 'cuatro']
>>>
>>> listaluis.remove(4)
>>> listaluis
[1, 2, 3, 'uno', 'dos', 'tres', 'cuatro']
>>>
```

Ln: 18 Col: 4

Figura 3.31

Existe una lista completa de métodos que se puede consultar en la documentación de Python accediendo libremente a su página web.

3.3.7 Explorando las tuplas

Una tupla guarda información fija. No se pueden agregar o quitar elementos de una tupla. Los contenidos son fijos o inmutables. No obstante, se puede acceder a todos los elementos de la forma habitual. También es posible comprobar si un elemento se encuentra en una tupla. Las tuplas son más rápidas y más eficientes que las listas. Es conveniente usarlas cuando tenemos una lista de elementos que no cambian. Las tuplas se encierran entre paréntesis — **(+)** — y los elementos de tupla están separados por comas. Para evitar una posible confusión con otras características de Python, un único elemento de una tupla siempre tiene una coma después de ella. En la figura xxx se observa la creación de una tupla simple. Por otra parte no podemos utilizar el método **append()** que usábamos con las listas. Véase la figura 3.32 donde se nos indica el error:

```
>>> tuplaluis
(1, 2, 3, 'uno', 'dos', 'tres')
>>> tuplaluis.append('cuatro')

Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    tuplaluis.append('cuatro')
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

Ln: 13 Col: 4

Figura 3.32

En general los métodos descritos para las listas no funcionan con la tuplas.

Python devuelve *True o False* cuando se utiliza para comprobar si un elemento se encuentra en una tupla como se demuestra en la siguiente figura 3.33:

```
>>> 1 in tuplaluis
True
>>> 3 in tuplaluis
True
>>> 'birmania' in tuplaluis
False
>>>
```

Ln: 31 Col: 4

Figura 3.33

A pesar de que no podemos variar los elementos de una tupla, lo que sí podemos hacer es unir varias tuplas usando el operador de suma tal y como se muestra en las figuras 3.34 y 3.35:

```
>>> tuplagerman = tuplaluis + tuplaluis
>>> tuplagerman
(1, 2, 3, 'uno', 'dos', 'tres', 1, 2, 3, 'uno', 'dos', 'tres')
>>>
```

Figura 3.34

Python Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7.3 (default, Mar 18 2014, 05:13:23)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> tuplaluis = (1,2,3,'uno','dos','tres')
>>> tuplaluis
(1, 2, 3, 'uno', 'dos', 'tres')
>>>
```

Ln: 7 Col: 4

Figura 3.35

3.3.8 Trabajando con diccionarios

Los diccionarios sirven para organizar la información en parejas. Mientras que las matrices utilizan un índice para acceder a un elemento, los diccionarios usan una llave o clave. Esta clave puede ser cualquier cadena o un número. También podemos utilizar una tupla como clave. El elemento vinculado a la llave es su valor. Los diccionarios son asociativos, lo que significa que enlazan una información con otra. Se debe usar un diccionario cuando se desee, de una manera fácil y rápida, buscar un valor dada una llave. A diferencia de una matriz o lista, podemos acceder a un valor de un diccionario sin tener que buscar en un determinado orden.

En la figura 3.36 se observa cómo se crea un diccionario y cómo se definen los distintos elementos que contiene, determinados por la pareja **clave/valor**. En la figura 3.37 se muestra cómo se accede a los distintos elementos que contiene el diccionario usando su clave o llave. En primer lugar

vemos que cambia el tipo de corchete y que, si en lugar de la llave usamos el valor el shell, nos informa de un error.

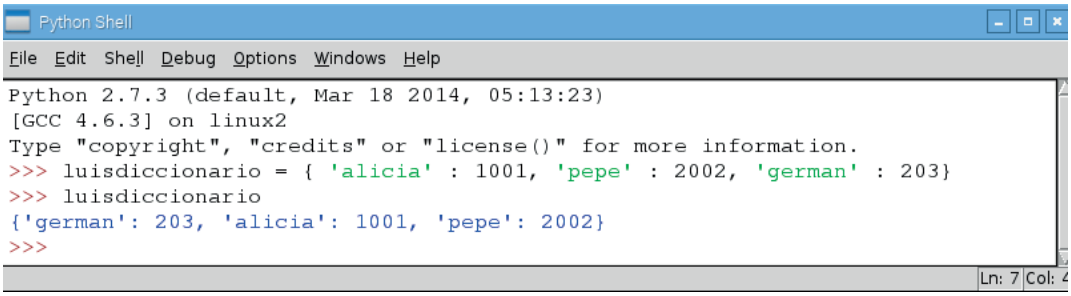


Figura 3.36

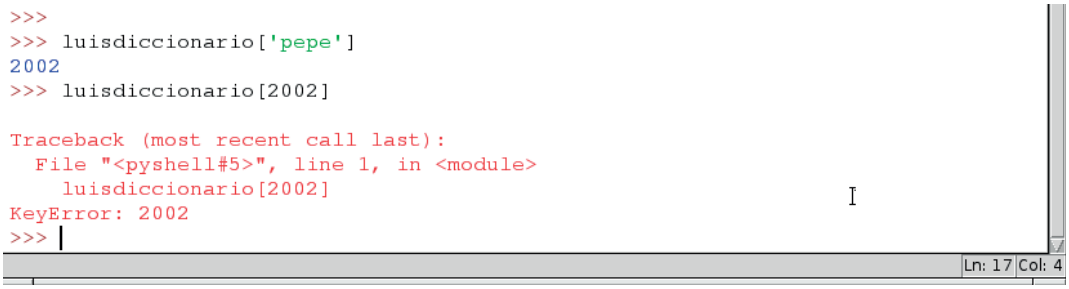


Figura 3.37

Si escribimos el nombre del diccionario seguido de la función **.keys()**, nos muestra todas las claves que contiene ese diccionario. Si le sigue la palabra **.values()**, lo que produce como resultado es el total de todos sus valores (figura 3.38).

Figura 3.38

Si escribimos el nombre de nuestro diccionario seguido de la función **.has_key('clave')** , nos imprimirá en pantalla el valor *True*, si encuentra esta clave, o *False* si no es así. Si utilizamos la función **.items()**, creará una lista de todos los elementos que contiene. Esto se observa en la figura 3.39:

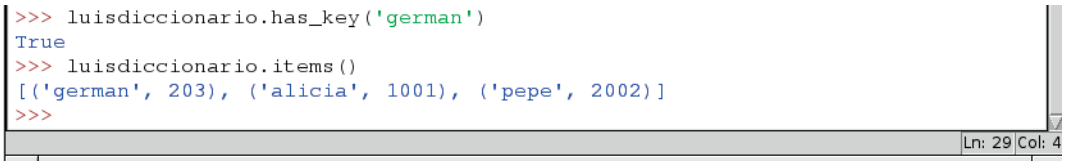


Figura 3.39

En general, un diccionario se debe utilizar cuando haya una referencia directa entre una clave y su valor. Por ejemplo, en el caso de un diccionario de texto, tenemos una correspondencia entre las palabras (clave) y su descripción o significado (valor). El orden de las claves y valores no es significativo. A diferencia de una matriz, donde los artículos se almacenan en estricta secuencia, los diccionarios no están organizados en un orden estricto. Si das un paso a través de cada par clave/valor en un diccionario, utilizando las herramientas de bucles y repetición que se describen en el siguiente apartado, no puedes confiar en que exista un orden establecido dentro del diccionario.

3.3.9 Comprendiendo las repeticiones y las decisiones

Todos los lenguajes de programación, incluyendo Python, incluyen muchas opciones para repetir código y realizar decisiones. Es posible utilizar estas características para diseñar aplicaciones "inteligentes" que puedan tomar decisiones para, por ejemplo, crear una aplicación en la que capte un fotograma determinado procedente de una webcam y que encienda una luz solo por las noches.

Las repeticiones y decisiones se describen a menudo con pseudocódigo, que es más fácil de entender que escritas en código Python real. Las figuras que se muestran a continuación están realizadas en pseudocódigo. Más adelante las traduciremos a código Python normal. Para hacer uso de repeticiones y decisiones, debe escribir código Python en múltiples líneas. Algo así como un script o una secuencia de comandos de Linux. Como hemos visto anteriormente el shell de Python trabaja a través de cada línea de una sola vez. No se puede escribir código de líneas múltiples directamente en este shell. Para utilizar varias líneas, debemos seguir las instrucciones que se expusieron cuando escribimos nuestro primer script en el apartado 3.3.5.

Una sentencia **if** (decisión) sirve para verificar si se cumple una determinada condición y en función de si el resultado es verdadero realizar algo y si es falso no hacerlo. Se puede añadir una sentencia **else** opcional que hace algo más si el resultado no es cierto. Python también incluye una declaración llamada **elif** que encadena varias condiciones posibles entre varias declaraciones. A menudo se necesita que un parte del código se repita un número de veces. Por ejemplo, es posible que desees escribir código que recorra todos los elementos de una lista o que se repite hasta que alguna condición se haga verdadera (*True*). Python incluye varias herramientas para implementar bucles de repetición como ser verá más adelante. Una sentencia **for** sirve para ejecutar un trozo de código de una forma controlada, indicando cuántas veces lo va a hacer y en qué condiciones lógicas. Ello será útil para contar a través de una lista o tupla. Podemos utilizar una sentencia **while** para realizar continuamente en bucle la ejecución de una parte de código hasta que una determinada condición lógica o aritmética deje de cumplirse.

3.3.10 Tomando decisiones

En la figura 3.40 observamos la utilización básica de la toma de decisiones con la sentencia **if**. Lanzamos, en primer lugar, el intérprete o shell de Pyhton y después seleccionamos la opción *New Window*. Dentro de esa nueva ventana es donde escribiremos nuestro ejemplo.

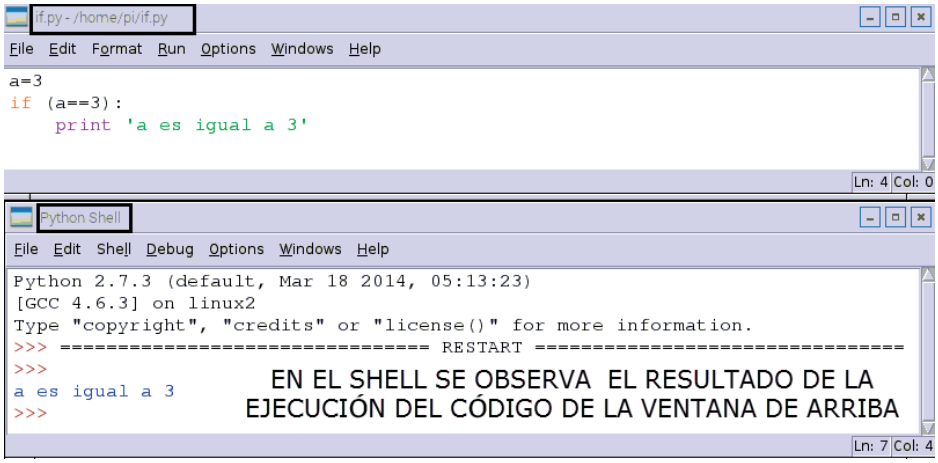


Figura 3.40

Utilizamos la condición (**a==3**) seguida de dos puntos. Si se cumple esta condición (que se cumple porque hemos definido en la línea anterior que a es igual a 3), ejecutamos la línea tabulada que nos imprime por pantalla el mensaje "a es igual a 3" como se puede observar en el Python shell de la figura anterior.

Otras posibilidades a tener en cuenta con la sentencia **if** son otros operadores de comparación de dos valores como >, > =, <=, <.

3.3.11 Trabajando con bucles y repeticiones

Una sentencia **for** sirve para repetir un bloque de código un número de veces especificado. Si por otra parte, deseamos testear que se cumpla o no, una determinada condición, es conveniente utilizar la sentencia **while**. Vamos a ver esto, de una forma práctica, con dos ejemplos básicos.

En la figura 3.41 se muestra la utilización de la sentencia **for**. En el programa se "imprime" por pantalla la letra "a" diez veces, debido a que ello se especifica en el rango de repetición del **for**.

En la figura 3.42 observamos el uso de la sentencia **while** con el mismo objetivo del ejemplo anterior. Es decir, imprimir la letra "a" diez veces. En este caso, el código lleva más líneas con lo que, en realidad, sería más conveniente usar la sentencia **for**. Ahora veremos cómo las funciones nos van a permitir reutilizar código y cómo los objetos en Python son útiles para empaquetar información y crear una serie de herramientas muy válidas a la hora de programar.

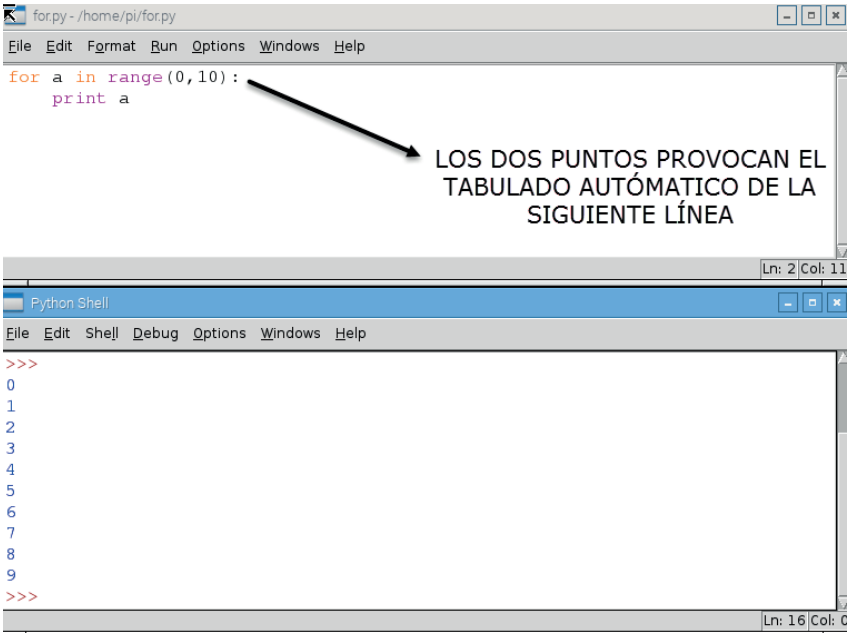


Figura 3.41

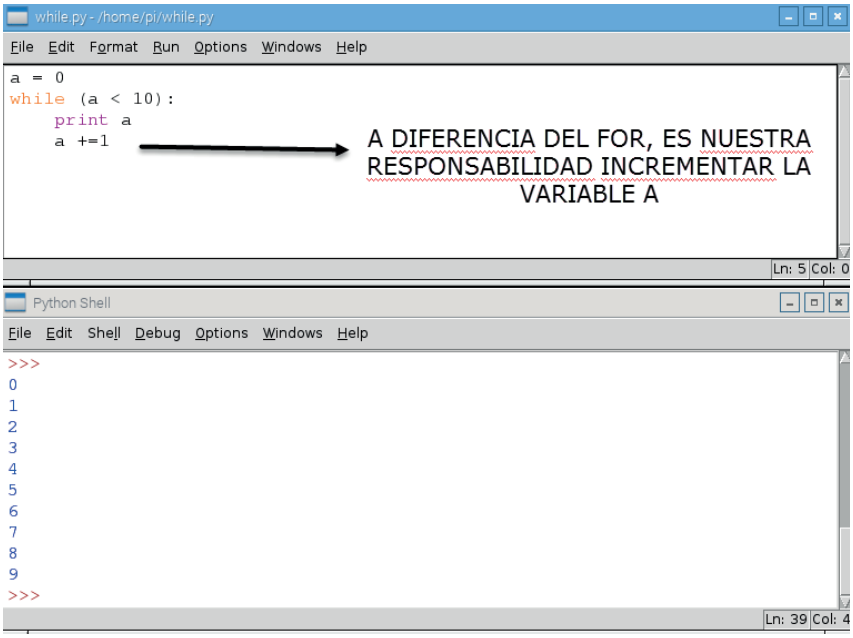


Figura 3.42

3.3.12 Comprendiendo las funciones y los objetos

En proyectos complejos a menudo se necesita realizar los mismos pasos una y otra vez. Es decir, reutilizar un trozo del código en varias partes distintas. Por ejemplo, en un juego es posible que se desee comprobar si alguno de los objetos del juego ha chocado con otros objetos. En lugar de copiar y pegar el código cada vez que lo utilice, podemos crear una función que detecte esta colisión. El código de la función se mantiene en un solo lugar. De esta manera llamaríamos a la función cada vez que necesitemos comprobar si se ha producido un choque de objetos.

Algunas funciones trabajan tal y como son y no es necesario más datos. Para ejemplo, una función como la descrita en el párrafo anterior u otra que devuelva la fecha y la hora no necesitan ninguna entrada. Otras funciones trabajan con la información que se les pasa. Esto se conoce como **paso de parámetros**. Cada elemento de información que enviemos a la función es un parámetro independiente. Las funciones pueden tomar cualquier número de parámetros, pero no se suelen pasar más que unos pocos debido a la claridad que debe presentar de cara al programador,

Cuando se define un parámetro, es válido para todo el código dentro de la función. Incluso si utiliza el mismo nombre en otra parte, Python mantiene automáticamente los nombres separados. Del mismo modo, si utilizamos las variables dentro de una función, sus nombres son "privados" con respecto a la función. No se puede acceder a ellas fuera de la función. Esta separación se llama **alcance variable**. Ayuda a mantener el código "saneado" en lo referente a los nombres de las variables, porque no tenemos que pensar en un nuevo nombre para cada variable en nuestro código.

Por otra parte, las funciones suelen devolver un resultado cuando finaliza su ejecución. Aunque es verdad que es posible escribir funciones que no retornen nada (*void*).

Otro concepto importante en muchos lenguajes de programación, incluido Python, es el concepto de **clase.** La clase es como un molde. Incluye una lista de variables que puede acceder y una lista de métodos, que son funciones que se construyen en la clase. Las clases son útiles cuando se quiere crear un montón de objetos similares y definir cuál es la información que poseen y lo que hacen. Por ejemplo, podría utilizar una clase para definir un azulejo de un juego tipo arcade y almacenar información sobre su posición y el color. Tendríamos una biblioteca de métodos para crear el azulejo, eliminarlo y, tal vez, para moverlo de un lugar a otro.

Después de definir una clase, podemos crear instancias de ella. El concepto de **instancia** es también importante en los modernos lenguajes de programación. Cada instancia se hizo desde un objeto del molde clase y posee los mismos atributos. Podemos dar a cada objeto un nombre de variable estándar cuando lo creemos. Python usa la notación punto para acceder a los atributos de los objetos. Por ejemplo, si creamos un objeto llamado *miazulejo* con los atributos de posición llamados *x* e *y*, se puede acceder a estas coordenadas cartesianas mediante *miazulejo.x* y *miazulejo.y*.

3.3.13 Creando una función

Usamos la palabra clave **def** para definir una nueva función seguida del nombre de la función y cero o más parámetros entre paréntesis. Ponemos fin a la primera línea de la definición con dos puntos (la línea siguiente aparecerá tabulada). Ahora nos referiremos a la función mediante su nombre. Si ha definido algún parámetro, lo incluiremos entre los paréntesis. En el ejemplo mostrado en la figura 3.43 se crea una función simple que duplica simplemente un número.

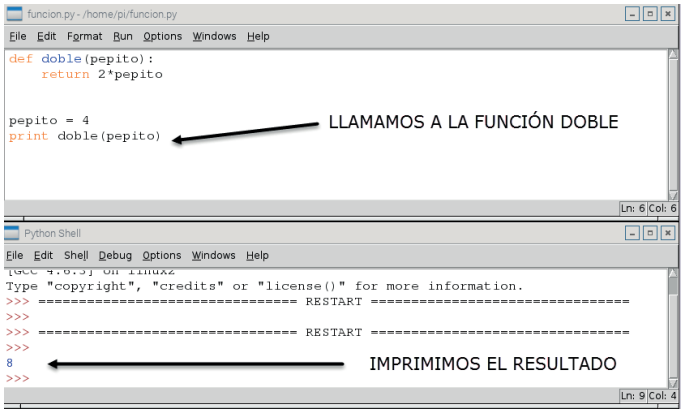


Figura 3.43

3.3.14 Definiendo una clase

Podemos utilizar la clase para crear nuestras propias clases. Para crear una clase útil, debemos incluir una selección de variables y métodos. Después de definir una clase, podemos crear tantas instancias de ella como necesitemos. Cada instancia posee sus propios datos. Las clases incluyen, a menudo, una función especial llamada **_init_** que se ejecuta automáticamente cuando se crea una. Podemos usar este método para establecer los valores de todas las variables utilizadas en su clase o para devolver un valor de utilidad. Las definiciones de clases a menudo se refieren a **self**, que es una manera abreviada de decir "esta instancia".

En la figura 3.44 se muestra como crear una clase en Python.

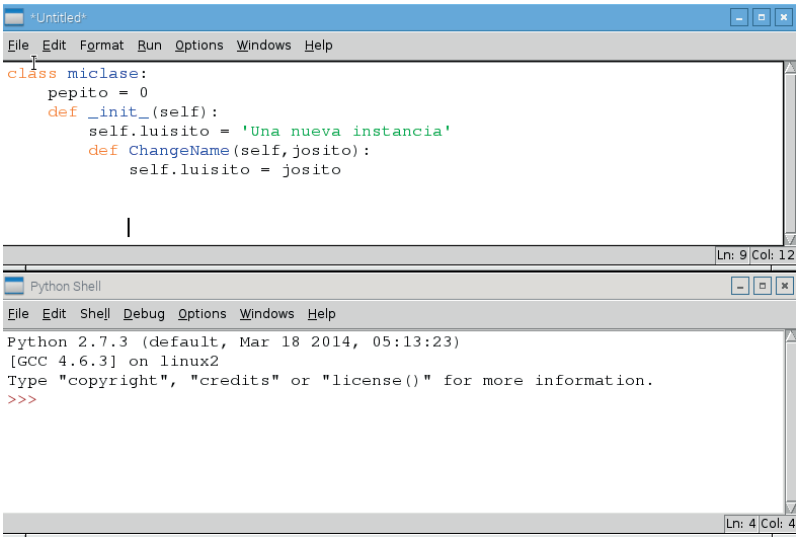


Figura 3.44

3.3.15 Cargando módulos en Python

En los próximos capítulos utilizaremos módulos para empaquetar y reutilizar el código y para ampliar las características de Python. Los módulos son bloques escritos en Python. Muchas de las funciones de Python solo están disponibles si los importamos previamente desde un módulo. El proceso de importación va normalmente al comienzo de nuestro código. Para utilizar las funciones de un módulo lo importaremos por su nombre y accederemos a sus funciones con notación de puntos. Por ejemplo, si importamos el módulo de matemáticas, podemos utilizar cualquiera de las funciones que contiene este módulo poniendo matemáticas delante del nombre de la función. También es posible importar características de forma selectiva utilizando **from**. En la figura 3.45 se observa que si pretendemos usar el método factorial del módulo de matemáticas sin haberlo importado previamente nos da error. Si ahora importamos el módulo **math**, ya estamos en condiciones de usar el método factorial y obtendremos un resultado (figura 3.46). En la figura 3.47 se observa el procedimiento para importar todos los métodos desde el modulo decimal (aritmética) y cómo utilizarlo. La lista disponible de módulos se puede consultar en la web oficial de Python. En los próximos capítulos a menudo procederemos a importar distintos módulos según la aplicación que estemos diseñando.

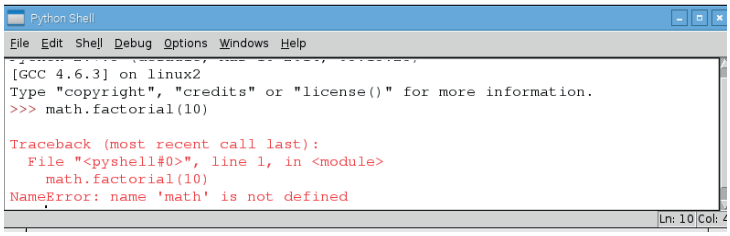


Figura 3.45

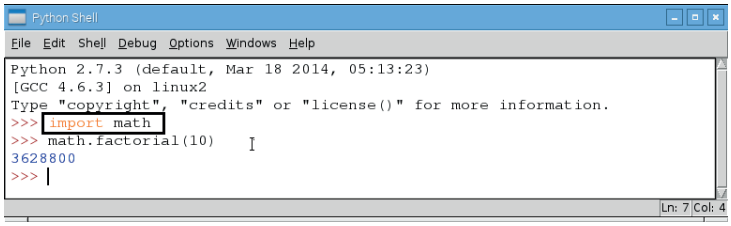


Figura 3.46

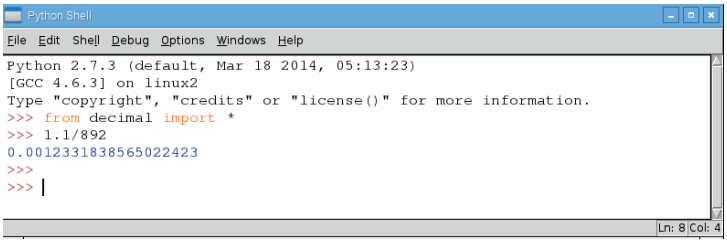


Figura 3.47

CAPÍTULO

4 ENTRADAS Y SALIDAS EN LA RASPBERRY: GPIO

Los pines **GPIO** (**G**eneral **P**urpose **I**nput **O**utput) de la Raspberry Pi nos permiten interactuar con el mundo exterior y, cómo no, con la electrónica que la rodea. Son pines o terminales que están pensados para comunicarnos con los más diversos dispositivos electrónicos que deseemos controlar o de los cuales queramos obtener datos e información en general.

Vamos a trabajar con ellos de aquí al final del libro y, por ello, es necesario hacer una pequeña introducción de su disposición y cometido. Debido a que todos los proyectos electrónicos los realizaremos basándonos en la novedosa Raspberry Pi2, o en su defecto a su modelo B+, que es su antecesora, pero que guarda una compatibilidad al 100 % en lo que al hardware se refiere. En otras palabras, si aún no has adquirido la Pi2 y posees la anterior, no te preocupes, porque todos los ejemplos expuestos de ahora en adelante funcionarán igualmente.

Cada pin del puerto GPIO tiene su propósito, varios pines trabajando en conjunto pueden formar un circuito en particular. El diseño del puerto GPIO sobre una Pi2 puede verse en la figura 4.1.

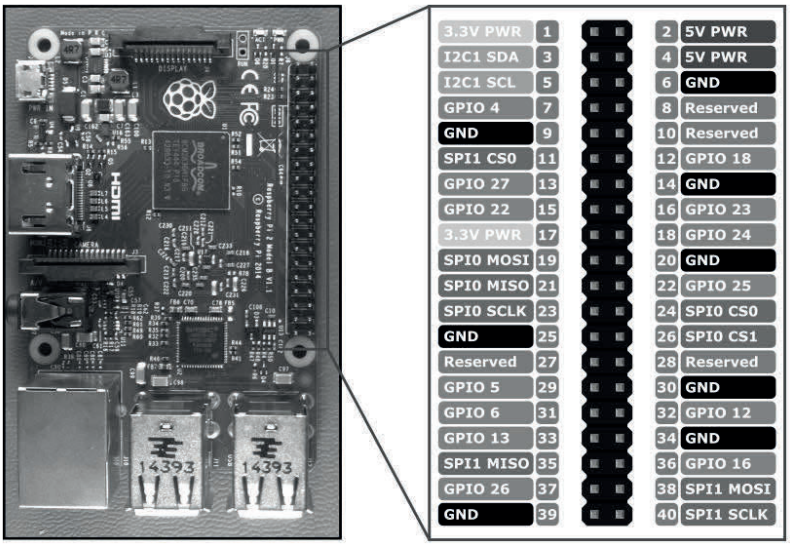


Figura 4.1

La numeración de los pines del puerto GPIO está dividida en dos filas, la fila inferior toma los números impares y la fila superior los números pares. Es importante tener esto en cuenta a la hora de trabajar con el puerto GPIO de la RasPi: la mayoría de los otros dispositivos electrónicos utilizan un sistema diferente de enumeración de pines y, debido a que no existen serigrafías sobre la misma placa de la Pi2, es fácil confundir cuál pin es cuál.

El corazón de la Pi2 se basa en un chip BCM2836. A diferencia de los microprocesadores tradicionales, estos están diseñados para ser utilizados en un sistema embebido. Un sistema embebido posee una especie de ordenador miniaturizado. Este tipo de chips tienen un número de conexiones o pines para que el software pueda controlar infinidad de periféricos. El BCM2836 tiene 54 pines de entrada/salida (GPIO). Algunas de estas señales se utilizan para controlar los dispositivos periféricos que convierten el BCM2836 en un miniordenador, permitiendo funciones como el lector de tarjetas SD, el controlador USB o la propia conexión Ethernet. El resto de los pines son libres.

En lugar de simplemente ignorarlos, los diseñadores de la Raspberry Pi han puesto a disposición del usuario, fuera del chip, algunos de estos GPIO excedentes y los han llevado a un conector denominado **P1** para que los usemos libremente. Otros pines van a otros conectores, como el de conexión a la cámara, y algunos ni siquiera están conectados a nada en absoluto. Precisamente, los pines GPIO son conocidos como de propósito general porque los podemos utilizar para cualquier cosa que queramos controlar mediante un programa. Como se comentó anteriormente, se llaman pines de entrada/salida debido a que el software puede configurarlos para ser una entrada o una salida.

Cuando un pin es una entrada, el programa puede leer si este tiene un alto o un bajo voltaje. Cuando, por el contrario, el pin es una salida, el software puede generar un voltaje alto o bajo en esa patilla. Además, muchos pines tienen una o más características alternativas que les dotan de funcionalidades extras para controlar hardware específico como protocolos SPI o I2C.

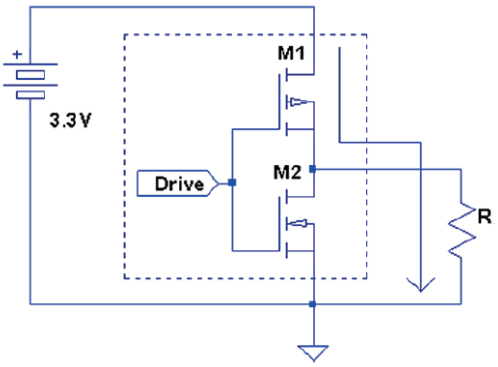
Aunque el puerto GPIO de la RasPi ofrece un suministro de energía de 5 V, proveniente de la entrada de alimentación del conector microUSB, su funcionamiento interno se basa en lógica de 3,3 V. Esto significa que sus componentes funcionan con un voltaje de 3,3. Es necesario asegurarse de que se están utilizando componentes compatibles con lógica de 3,3 V o que el circuito está pasando a través de un regulador de voltaje antes de llegar a la Raspberry.

Conectar un voltaje de 5 V a cualquier pin del puerto GPIO de la Raspberry Pi, o hacer cortocircuito directamente con cualquiera de los dos pines de alimentación (Pin 1 y Pin 2) hacia cualquier otro pin, dañará irreversiblemente la RasPi debido a que el puerto está directamente conectado a los pines del procesador SoC BCM2836 y este trabaja con 3,3 voltios.

La figura 4.2 muestra el circuito equivalente de un pin GPIO cuando es configurado como una salida. Cuando se configura un pin GPIO como salida, el usuario es plenamente responsable de la limitación de corriente a través del mismo. No existe una limitación de corriente por defecto. La figura ilustra cómo la corriente fluye desde el suministro de 3,3 V a través del transistor M1, el pin GPIO, y la R hasta tierra. Debido a esto, se necesita un nivel alto (1 lógico) para enviar corriente a la carga.

La figura 4.3 muestra cómo la salida GPIO absorbe corriente ya que el pin GPIO está configurado con un voltaje bajo como salida. Debido a esto la R está conectada al suministro de 3,3 V; la corriente fluye a través de R, por el pin de salida GPIO y a través del transistor M2 hacia tierra.

Cuando el pin de salida está en el estado alto, se comporta como una fuente de tensión que trata de suministrar 3,3 V o de absorber corriente si está a 0 voltios (dentro de los límites del transistor). Si esta salida está en cortocircuito con tierra (peor caso), la corriente excesiva producirá un daño permanente en la Raspberry. La corriente que estas salidas pueden suministrar están limitadas a una corriente de aproximadamente 16 mA. Este límite es la cantidad de corriente máxima que debe suministrar a una carga.



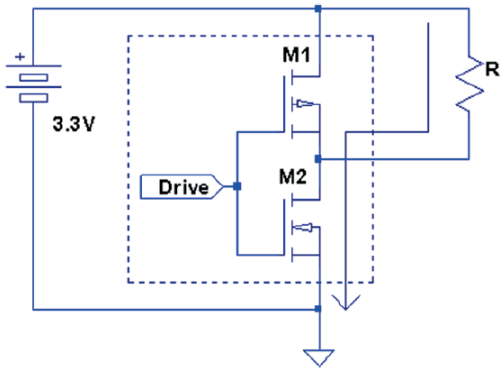


Figura 4.2 **Figura 4.3**

4.1 PRÁCTICA 1: PARPADEO DE UN LED

Construiremos un circuito simple que consistirá en un LED y una resistencia. El LED proporcionará una confirmación visual de que el puerto GPIO hace lo que su programa en Python le dice que haga y la resistencia limitará la corriente consumida por el LED para protegerlo y, también, para evitar dañar la Raspberry.

Un LED necesita una resistencia que limite la corriente para protegerlo de que se queme. Sin una resistencia, un LED probablemente solo funcionaría durante un tiempo corto antes de fallar y necesitaría ser reemplazado. Es importante saber que las resistencias son necesarias, pero también es importante saber escoger la resistencia adecuada. Un valor demasiado alto y el brillo del LED será extremadamente débil o no encenderá en absoluto; un valor demasiado bajo y se quemará. Para calcular el valor de la resistencia requerida, necesitaremos conocer la corriente de trabajo de nuestro LED. Esta es la máxima intensidad de corriente que puede circular a través del LED sin que se dañe midiéndose en miliamperios (mA).

La manera más fácil de calcular el valor máximo de la resistencia es utilizando la fórmula siguiente, en donde *R* es la resistencia en ohmios, *Vfuente* es la tensión aplicada al LED, *Vdiodo* es la tensión directa del LED e *I* es la máxima corriente de trabajo del LED.

$$R = \frac{V_{fuente} - V_{diodo}}{I}$$

Tomando un típico LED rojo con una corriente de trabajo de 25 mA y una tensión directa de 1,7 V y alimentándolo mediante el suministro de 3,3 V del puerto GPIO, se puede calcular la resistencia requerida con (3,3 − 1,7) / 0,025 = 64. Por lo tanto, una resistencia de 64 Ω o mayor protegerá al LED. Las cifras obtenidas raramente coinciden con los valores comunes de las resistencias que se venden, así que cuando elija una resistencia, siempre hay que redondear al valor superior para asegurarse de que el LED estará protegido. El valor disponible de resistencia comúnmente más cercano es de 68 Ω, el cual protegerá apropiadamente el LED.

Si no conoce le tensión directa y la corriente de trabajo de sus LEDS (por ejemplo, si el LED no viene con documentación o fueron rescatados de entre la chatarra electrónica), podemos tratar por prueba y error, colocando una resistencia razonablemente grande. Si la luz es demasiado débil, podemos probar con una resistencia más baja (pero es necesario recordar que es imposible reparar un LED quemado).

Para montar el circuito, necesitaremos una breadboard, un cable y conector Pi Cobbler Plus (figura 4.4), dos cables jumpers, un LED y una resistencia limitadora de corriente apropiada (figura 4.5). El GPIO 6 es tierra y el GPIO 18 es un pin de propósito general que debe configurarse como salida y que se corresponde con la patilla 12 del puerto P1 de la Raspberry (a menudo no existe correspondencia en la numeración de orden de las patillas con el número de GPIO, tal y como se observó en la figura 4.1).

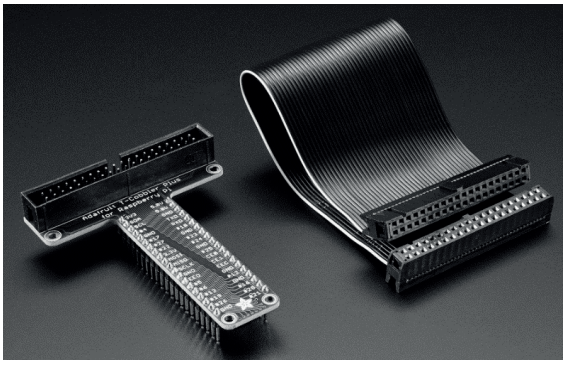


Figura 4.4

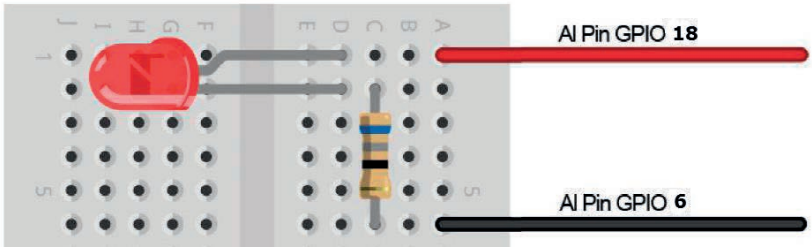


Figura 4.5

El montaje final se observa en la figura 4.6. Llegados a este punto, nada sucederá. Esto es perfectamente normal: por defecto, los pines GPIO de la Raspberry Pi se encuentran apagados. Para hacer que el LED haga algo útil, debemos comenzar un nuevo proyecto en Python. De la misma forma que se hicieron en el capítulo anterior, podemos utilizar un editor de texto plano o el software IDLE en el entorno gráfico para trabajar con estas prácticas.

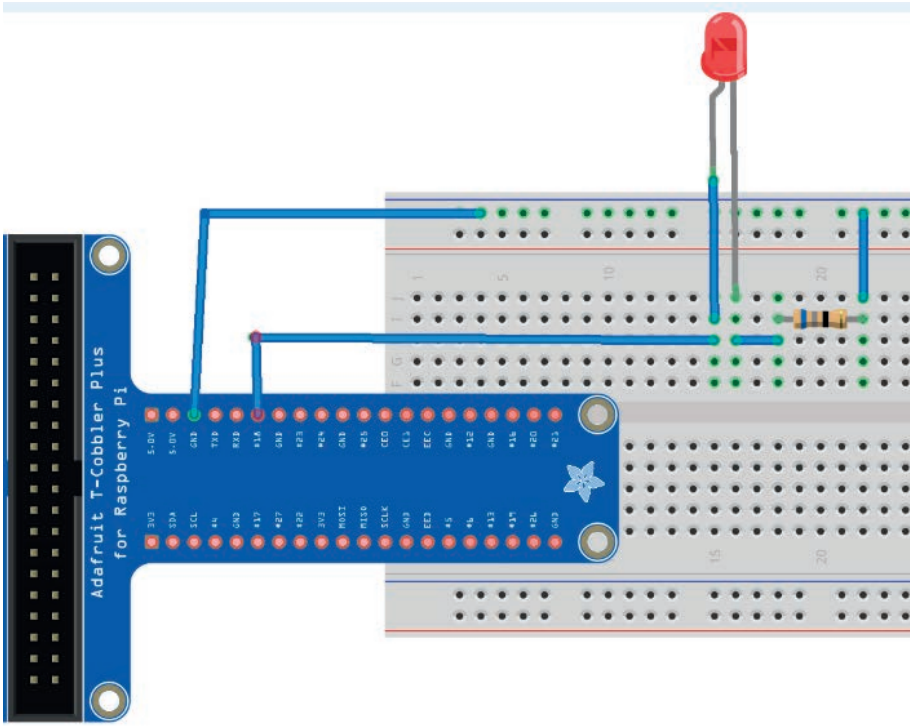


Figura 4.6

Antes de que podamos utilizar el puerto GPIO de la Raspberry Pi en Python, necesitaremos importar una librería dentro de nuestro proyecto Python. Para hacerlo, debemos comenzar el archivo con la siguiente línea:

```
import RPi.GPIO as GPIO
```

Es preciso recordar que Python es sensible a las mayúsculas y minúsculas, así que debemos asegurarnos de escribir **RPi.GPIO** exactamente como aparece. Para permitir a Python entender el concepto de tiempo (en otras palabras, hacer que el LED parpadeé, encendiéndose y apagándose), necesitaremos también importar el módulo **time**. Agregamos la siguiente línea al proyecto:

```
import time
```

Con las librerías ya importadas, es momento de controlar los puertos GPIO. La librería GPIO nos facilita controlar los puertos de propósito general a través de las instrucciones **GPIO.output** y **GPIO.input**, pero antes de poder utilizarlas necesitaremos establecer la librería GPIO al modo **BOARD** o al modo **BCM**. Existen dos sistemas de numeración de los pines GPIO: **BCM** y **BOARD**. El sistema **BCM** usa el número de pin GPIO correspondiente. En nuestro caso usamos el GPIO 18, por lo tanto ponemos el número 18. En la figura 4.7 se corresponden las etiquetas externas. En el sistema **BOARD** la numeración se basa en el orden de los pines de arriba abajo de la placa. En la figura 4.7 se corresponden la numeración interna o etiquetas que figuran dentro (en el caso de utilizar el sistema **BOARD** usaríamos el número 12).

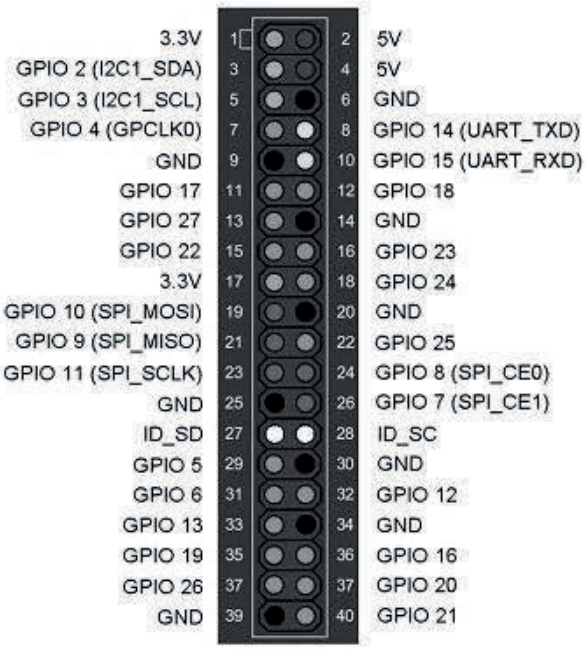


Figura 4.7

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(18,GPIO.OUT)
```

La última línea le dice a la librería GPIO que el pin 18 sobre el puerto GPIO de la Raspberry Pi debe ser configurado como una salida. Si va a controlar dispositivos adicionales, podemos agregar más líneas **GPIO.setup** al proyecto. Sin embargo, por ahora, con una sola bastará. Ya con el pin configurado como una salida, se puede conmutar su suministro de 3,3 V entre encendido y apagado en una simple demostración de lógica binaria. La instrucción GPIO.output (18, *True*) transformará al pin en activo, mientras GPIO.output (18, *False*) lo transformará en inactivo. El pin recordará su último estado, de modo que si solo da la orden de cambiar el pin a su estado activado y luego se sale del programa Python, el pin mantendrá su estado activo hasta que se indique lo contrario.

Aunque con solo agregar GPIO.output (18, *True*) al proyecto Python se puede activar el pin, será más interesante hacer que este parpadeé. Primero, agregamos la siguiente línea para crear un bucle infinito en el programa:

```
while True:
```

A continuación, agregamos las siguientes líneas para activar el pin, esperamos dos segundos y luego los desactivamos volviendo a esperar otros dos segundos. Nos cercioramos de que cada línea comienza con cuatro espacios para indicarle a Python que estas líneas son parte del bucle **while** infinito:

```
    GPIO.output(18, True)
                                time.sleep(2)
    GPIO.output(18, False)
    time.sleep(2)
```

El programa terminado se muestra en la figura 4.8 utilizando el IDLE de Python bajo el entorno grafico de la Raspbian. Guardamos el archivo como *led.py*. La mayoría de las distribuciones Linux Raspberry Pi limitan el uso del puerto GPIO al usuario **root**, por lo que será necesario que el programa sea ejecutado utilizando el comando **sudo python led.py** en la terminal para ponerlo en marcha. Si todo ha salido bien, deberíamos observar que el LED comienza a parpadear, prendiéndose y apagándose a intervalos regulares de 2 segundos.

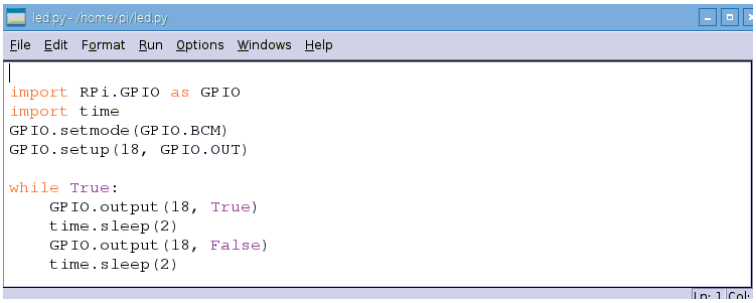


Figura 4.8

Si las cosas no funcionan, no nos asustemos. Primero, verificamos todas las conexiones. Los orificios en la breadboard son bastante pequeños y es fácil pensar que ha insertado un componente en una fila para descubrir que en realidad lo insertó en otra. A continuación, verificamos que ha conectado el circuito a los pines correctos en el puerto GPIO (al no contar con etiquetas la Raspberry Pi, desafortunadamente los errores son fáciles de cometer). Finalmente, revisamos dos veces sus componentes, si la tensión directa de su LED es más alta que los 3,3 V o si la resistencia limitadora de corriente es demasiado grande, el LED no se iluminará.

Aunque este ejemplo es básico, resulta una buena demostración de algunos de los conceptos fundamentales. Para extender su funcionalidad, el LED podría reemplazarse con un zumbador o *buzzer* para crear una alerta audible, o con un servo o motor como parte de una plataforma robótica. El código

utilizado para activar y desactivar el pin GPIO puede ser incorporado dentro de otros programas, provocando que un LED se ilumine cuando un nuevo correo electrónico nos llega o cuando un amigo se ha unido a un canal IRC, por ejemplo.

4.2 PRÁCTICA 2: LECTURA DE UN PULSADOR

En el siguiente ejemplo veremos cómo conectar un pulsador en otro pin del puerto GPIO y leer su estado desde Python. Construimos el circuito de la manera siguiente según se muestra en las figuras 4.9 y figura 4.10. Volveremos a utilizar el "Adafruit Cable Cobbler Plus" que, como hemos visto en la práctica 1, es muy cómodo para conectar adecuadamente la Raspberry a la placa breadboard.

El circuito que acaba de construir crea una situación en la que el pin de entrada (que en este caso, es el Pin 18 del puerto GPIO) alcanza constantemente su estado alto (*high*) gracias a la resistencia "pull-up" conectada a un voltaje de 3,3 V. Cuando el pulsador es presionado, el circuito se conecta a tierra y toma un estado bajo (*low*), proporcionando la señal para que nuestro programa Python sepa que el pulsador ha sido activado.

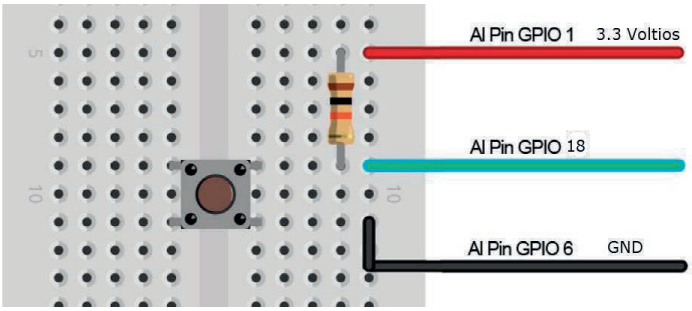


Figura 4.9

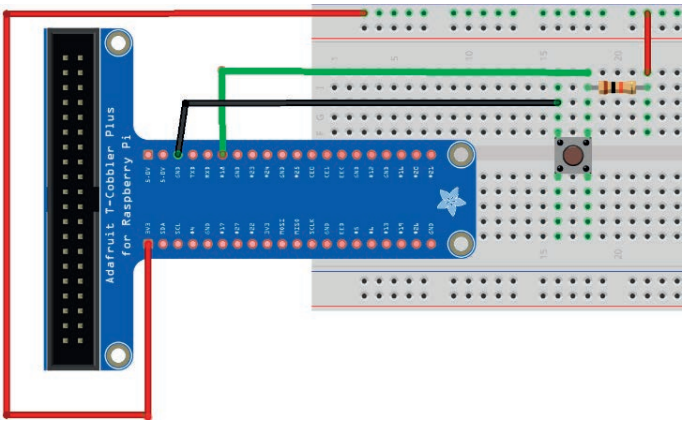


Figura 4.10

Abrimos un nuevo archivo Python, ya sea en el editor de texto o utilizando uno de los entornos de desarrollo integrado Python (IDLE) disponibles en la Raspberry Pi. Para comenzar, necesitaremos importar la misma librería GPIO que en el ejemplo pasado de la salida GPIO y establecer la configuración del GPIO 18 como entrada.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.IN)
```

Esta vez, no necesitamos importar la librería **time**, porque en esta práctica no se requieren instrucciones relacionadas con el tiempo.

Al igual que en el ejemplo pasado, el siguiente paso es crear un bucle infinito que constantemente comprobará si el pin de entrada se encuentra en su estado bajo (en otras palabras, si ha sido presionado). Comenzamos el bucle con la siguiente línea de código:

```
while True:
```

Leer el estado de un pin de entrada es muy similar a establecer el estado de un pin de salida, con una excepción: antes de poder hacer algo útil con el valor obtenido, necesitará almacenar este valor en una variable. La siguiente instrucción le dice a Python que cree una nueva variable llamada **input_value** y que le asigne el valor actual del GPIO 18:

```
input_value = GPIO.input(18)
```

Aunque el programa puede ser ejecutado ahora y funcionar, no hace nada útil. Para asegurarse de saber qué es lo que está pasando, agregamos la siguiente instrucción **print** para obtener retroalimentación:

```
if input_value == False:
    print("El pulsador se ha presionado.")
    while input_value == False:
          input_value = GPIO.input(18)
```

Las dos últimas líneas (el segundo **while** y el segundo **input_value**) son importantes. Aun en el procesador de la Raspberry Pi (que es relativamente menos potente si se compara con los procesadores de los portátiles y ordenadores de escritorio de alto rendimiento), Python se ejecuta muy rápidamente. Este bucle anidado le dice a Python que se mantenga comprobando el estado del pin 18 hasta que ya no sea bajo, en cuyo caso sabrá que el botón ha sido liberado. Sin este bucle, el programa se repetiría mientras el pulsador esté siendo presionado (y no importa cuán rápidos sean sus reflejos, verá en la pantalla imprimirse el mensaje varias veces, lo cual no es lo correcto).

El programa final debe verse como en la figura 4.11:

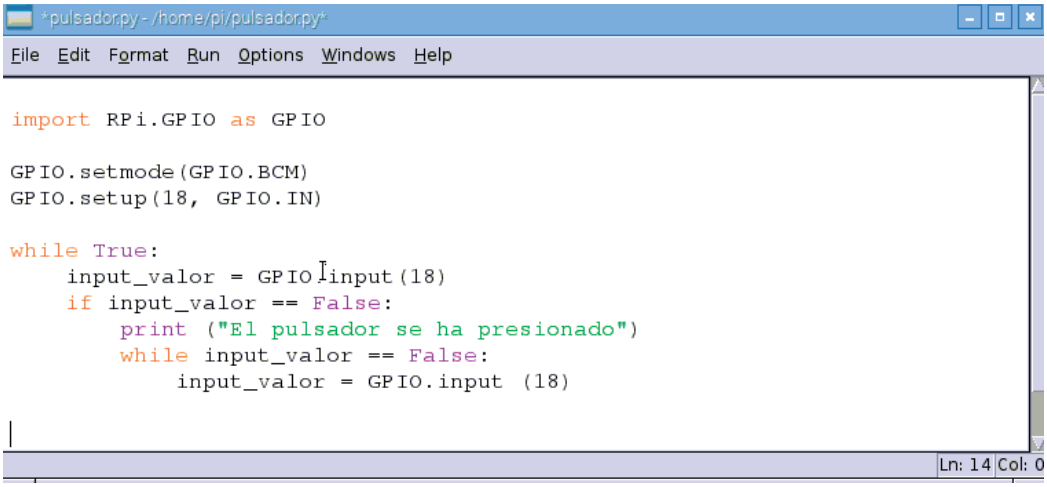


Figura 4.11

Guardamos el archivo como *pulsador.py* y luego los ejecutamos desde la terminal con `sudo python pulsador.py`. Al comenzar el programa, nada se muestra sobre la pantalla, pero si activamos el pulsador, el programa empezará a imprimir en la terminal el mensaje de la línea número siete de nuestro programa (figura 4.12). Soltamos el pulsador y al volver a presionarlo, el mensaje se repetirá.

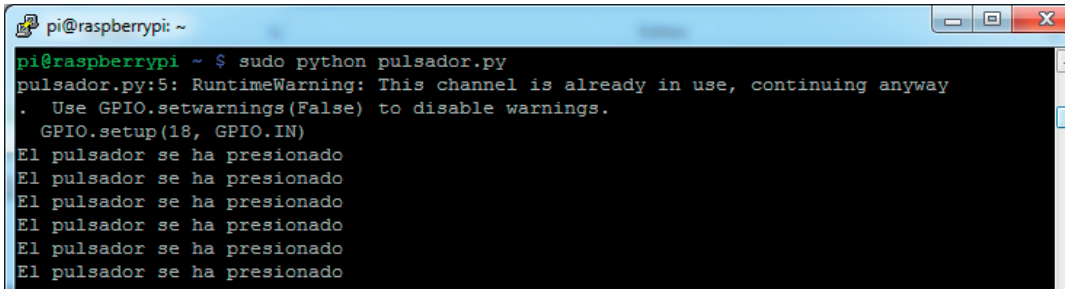


Figura 4.12

Del mismo modo que el ejemplo pasado de la entrada GPIO, este ejemplo es aparentemente un simple programa que puede ser utilizado para muchos propósitos. Además de ser capaz de leer cuando un botón es presionado, el mismo código se puede utilizar para leer cuando los pines de un dispositivo aparte (como puede ser un sensor o microcontrolador externo) han sido cambiados al estado alto (*high*) o bajo (*low*).

4.3 CONTROLANDO GPIO A TRAVÉS DE LA LIBRERÍA WIRINGPI

WiringPi es una librería para la Raspberry Pi creada por Gordon Henderson para acceder y usar el puerto GPIO a través del lenguaje C. De esta manera, programar los pines GPIO es muy parecido a programar un Arduino. Esta librería soporta lectura y escritura analógica a través de módulos externos (recordemos que los pines GPIO no tienen entradas analógicas). Esta librería tiene su propia numeración de los pines GPIO tal y como se muestra en la figura 4.13.

P1: The Main GPIO connector							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

Figura 4.13

Por tanto, si eres de los aficionados que vienen del mundo Arduino o simplemente te gusta el lenguaje C para programar dispositivos, lo que se expone a continuación, sin duda, te llamará la atención.

La página oficial de referencia es: http://wiringpi.com/

Para instalar la librería en la Raspberry Pi se debe descargar a través de GIT. Si aún no tenemos instalado GIT en la Raspberry Pi, escribimos el siguiente comando:

```
sudo apt-get install git-core
```

Ahora, descargamos WiringPi:

```
git clone git://git.drogon.net/wiringPi
```

Vamos al directorio recién creado:

```
cd wiringPi
git pull origin
```

Lo instalamos:

```
cd wiringPi
./build
```

Una vez instalada la librería, creamos un archivo:

```
sudo nano led.c
```

Escribimos el siguiente código:

```
#include <wiringPi.h> //Importamos la libreria WiringPi
int main()
{
wiringPiSetup () ;
pinMode (7, OUTPUT); //GPIO4 corresponde al pin 7 de WiringPi
    for (;;)
    {
        digitalWrite (7, HIGH);
        delay (500);
        digitalWrite (7, LOW);
        delay (500);
    }
}
```

Para guardar pulsamos CTRL+X, luego S e INTRO. Ahora tenemos que compilar el código:

```
gcc -Wall -o led led.c -lwiringPi
```

Lo ejecutamos y observamos el resultado en el hardware conectado. Tened en cuenta que hemos cambiado el número del GPIO con respecto al ejemplo anterior del apartado 3.2.

```
sudo ./led
```

Para salir del programa, pulsamos CTRL+C.

4.4 MIDIENDO TEMPERATURAS CON UN SENSOR DIGITAL DS1820

Vamos a utilizar el sensor de temperatura **D18B20 Dallas** para medir temperaturas con nuestra Pi2. Podemos comprarlo en Amazon y eBay. El sensor es pequeño pero sofisticado (figura 4.14). Incluye un microprocesador sencillo. Se conecta al GPIO de la Pi utilizando un bus **1-wire**. Se debe instalar el software adecuado para configurarlo y leer sus valores. Las lecturas aparecen como un archivo en un directorio con un nombre que incluye el número de serie único de cada sensor.

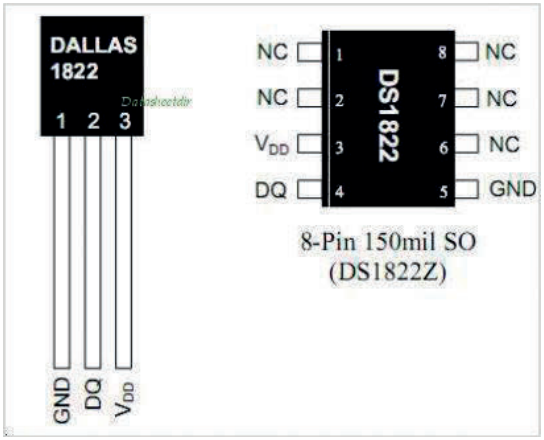


Figura 4.14

Los sensores 1-wire DS18B2Xx se pueden conectar en paralelo (a diferencia de casi cualquier otro sensor). Todos los sensores deben compartir los mismos pines, pero solo se necesita una resistencia 4,7 K para todos ellos. La resistencia se utiliza como un pull-up para la línea de datos y es necesaria para mantener una transferencia de datos estable. Observad cuidadosamente la figura 4.15 para realizar las conexiones adecuadamente y no estropear el sensor. Hay que tener en cuenta que la alimentación del sensor debe ser de 3,3 voltios. La patilla DQ de datos irá conectada al pin 4 GPIO.

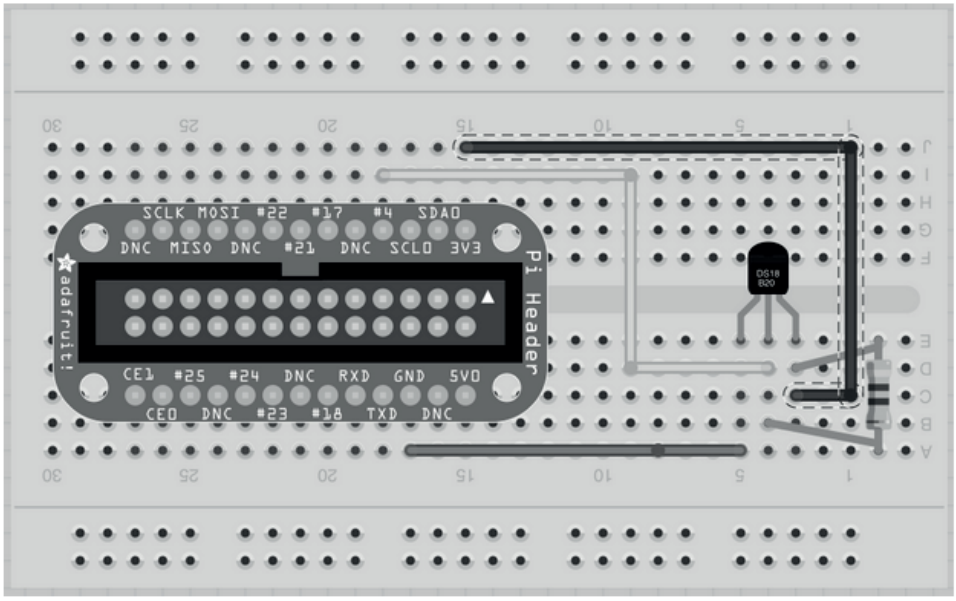


Figura 4.15

Comenzamos agregando la siguiente línea al /boot/config.txt. Podemos editar el archivo con nano ejecutando `sudo nano /boot/config.txt` y luego desplazarse a la parte inferior y escribiendo allí:

```
dtoverlay=w1-gpio
```

Luego reiniciamos la Rasp con el reinicio sudo. Ahora desde el terminal escribimos los siguientes comandos que se muestran en la figura 4.16. El nombre del directorio 22-xxxxxxx puede cambiar según el tipo de sensor y la configuración de la Raspberry.

Figura 4.16

La respuesta será SÍ (*yes*) si el sensor es reconocido. Si es así, entonces la temperatura se mostrará en la siguiente línea multiplicada por 1.000. Si tenemos más de un sensor conectado, podremos ver varios archivos 22-xxx. Cada uno tendrá el número de serie único por lo que puede conectar uno a la vez, observar lo que el archivo presenta, y etiquetar cada uno de los sensores.

4.5 MIDIENDO TEMPERATURAS CON UN CONVERTIDOR ADC Y UN TMP36

Una de las carencias de la Raspberry Pi son las entradas analógicas tal y como existen en otros dispositivos, con lo cual muchos sensores analógicos no los podemos utilizar directamente. Para ello precisaremos de un conversor analógico-digital (ADC) para poder dar uso a estos sensores tan necesarios para determinados proyectos.

El **MCP3008** es un convertidor ADC de 8 canales de 10 bits. Es barato, fácil de conectar y no requiere ningún componente adicional. Se utiliza el protocolo de bus **SPI** que está soportado por el GPIO de la Pi2.

Pero vamos por pasos, echemos un vistazo a las características del sensor de temperatura **TMP36** que poco a poco va sustituyendo en uso al popular LM35.

El TMP36 es un sensor de bajo voltaje y alta precisión (figura 4.17). Está diseñado para proporcionar un voltaje en proporción lineal a la temperatura en la natural escala Celsius. Con este

sensor no tendrás que preocuparte por calibraciones externas para obtener exactitudes de ±1 °C a +25 °C y ±2 °C en el espectro de −40 °C a +125 °C. Solo tienes que alimentarlo con un voltaje de 2,7 a 5,5 voltios y ya podemos leer el voltaje de salida. Finalmente, debemos mapear o ajustar el voltaje de salida a la temperatura usando el factor de 10 mV/°C.

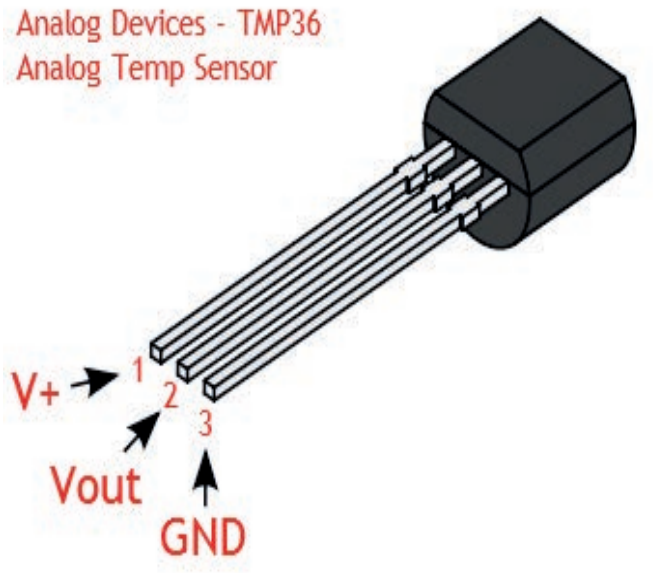


Figura 4.17

Características:

- ✓ Entrada: 2,7 V a 5,5 VDC.
- ✓ Factor de 10 mV/ °C.
- ✓ Exactitud de ±2 °C.
- ✓ Linealidad ±0,5 °C.
- ✓ Rango de operación: −40 °C a +125 °C.

Como se mencionó anteriormente, la salida del sensor es un voltaje proporcional a la temperatura y por ello es de naturaleza analógica. De ahí que necesitemos añadir a nuestro proyecto un convertidor ADC.

Vamos a utilizar el chip MCP3008 (figura 4.18). Es muy cómodo ya que tiene 8 entradas analógicas que podemos controlar fácilmente mediante un único bus SPI. Es especialmente útil con Raspberry Pi ya que no dispone de este tipo de entradas analógicas, y con la ayuda de esta placa y un pequeño script en Python podemos leer fácilmente los valores de todo tipo de sensores analógicos, potenciómetros, etc.

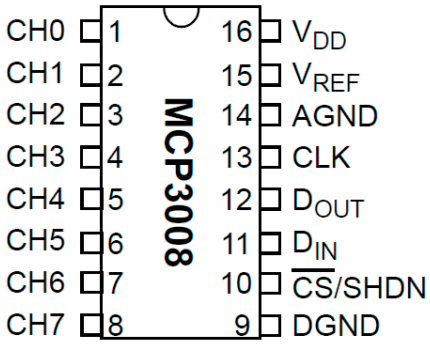


Figura 4.18

Características:

- ✓ Interfaz de datos: serie, SPI.
- ✓ Frecuencia de muestreo: 200 kSPS.
- ✓ Temperatura de trabajo máx.: −40 ºC a 85 °C.
- ✓ Resolución: 10 bits.
- ✓ Tensión de alimentación máx.: 2,7 V a 5,5 V.
- ✓ Dimensiones: 22 × 17 mm.

Lo podemos adquirir con encapsulado DIP o montado en SMD, quizá mucho más práctico para nuestros proyectos y con poca diferencia de precio (figura 4.19).

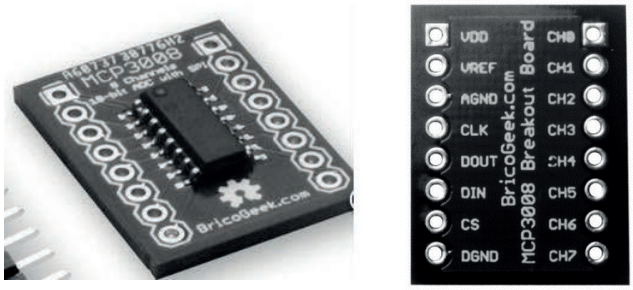
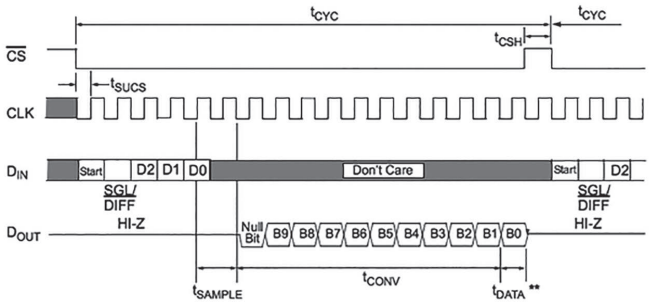


Figura 4.19

La Interfaz Periférico Serial (SPI) (figura 4.20) es un protocolo de comunicación utilizado para transferir datos entre dispositivos electrónicos como la Raspberry Pi. Estos dispositivos periféricos pueden ser tanto sensores o actuadores. En este ejemplo, vamos a aprender a utilizar un convertidor analógico-digital del sensor. Un análogo al sensor digital toma una tensión analógica y la convierte en un número digital que puede ser entendido por la Pi2.



COMUNICACIÓN CON EL MCP3008

Figura 4.20

SPI utiliza 4 conexiones separadas para comunicarse con el dispositivo de destino. Estas conexiones son el reloj de serie (**CLK**), Maestro Esclavo de Salida y Entrada (**MISO**), Maestro Esclavo de Entrada y Salida (**MOSI**) y Chip Select (**CS**).

Los impulsos de reloj con una frecuencia regular establecen la velocidad a la que la Raspberry Pi y el dispositivo SPI están de acuerdo para transferirse datos entre sí. Para el MPC3008 los pulsos de reloj se muestrean en su flanco de subida y en la transición de baja a alta.

El pin MISO es un pin de datos utilizado por el maestro (en este caso la Raspberry Pi) para recibir datos del ADC. Los datos se leen desde el bus después de cada pulso de reloj.

El pin MOSI envía datos de la Raspberry Pi al ADC. El ADC tomará el valor del bus en el flanco de subida del reloj.

Por último, la línea de Chip Select elige qué determinado dispositivo SPI está en uso. Si hay varios dispositivos SPI, todos pueden compartir la misma CLK, MOSI y MISO. Sin embargo, solo el dispositivo seleccionado tiene la línea de Chip Select en estado bajo, mientras que los demás dispositivos tienen sus líneas CS en estado alto. Un pin en alto de selección de chip le dice al dispositivo SPI que ignore todos los comandos y el tráfico del resto del bus.

Como ya sabemos, el ADC usado en este ejemplo es el MCP3008. Se trata de un convertidor de 8 canales analógicos de 10 bits. Puede aceptar hasta 8 diferentes tensiones analógicas; sin embargo, solo puede convertir un voltaje en un momento dado. La propiedad de 10 bits es la resolución del ADC o la precisión a la que se puede medir una tensión. La gama de tensiones analógicas se representa como un número de 10 bits en la salida. Si el ADC está alimentado a 3,3 V, cada paso en el valor de salida representa un cambio de 0,003 voltios.

Para leer datos analógico tenemos que utilizar los siguientes pines: **VDD** (poder), **DGND** (tierra digital) para alimentar el chip MCP3008. También necesitamos cuatro pines de datos **SPI**: **DOUT** (Out datos de MCP3008), **CLK** (pin del reloj), **DIN** (Datos a partir de la Raspberry Pi), y **CS** (Chip Select). Por último, por supuesto, una fuente de datos analógicos, y vamos a utilizar el sensor de temperatura TMP36

El MCP3008 además necesita otros pines que debemos conectar: **AGND** (tierra analógica, que se utiliza, a veces, en los circuitos de precisión) se conecta a **GND**, y referencia **VREF** (voltaje

analógico, utilizado para cambiar la “escala” a 3,3 V). A continuación se muestra un diagrama de cableado (figura 4.21):

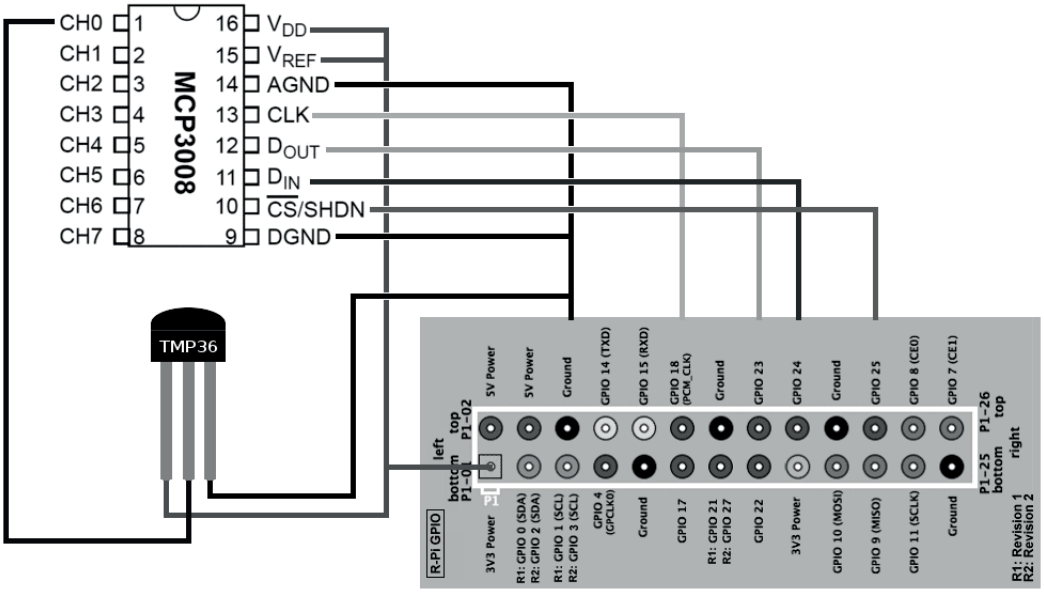


Figura 4.21

MCP3008 VDD -> 3,3 V

MCP3008 VREF -> 3,3 V

MCP3008 AGND -> GND

MCP3008 CLK -> # 18

MCP3008 DOUT -> # 23

MCP3008 DIN -> # 24

MCP3008 CS -> # 25

MCP3008 DGND -> GND

MCP3008 CH0 -> Vout (TMP36)

Los usuarios avanzados pueden observar que la Raspberry Pi tiene una interfaz SPI hardware (los pines se etiquetan: MISO / MOSI / SCLK / CE0 / CE1). La interfaz SPI hardware es muy rápida, pero no está incluida en todas las distribuciones. Por esa razón estamos utilizando una implementación SPI estándar por lo que los pines SPI pueden ser cualquiera de los pines GPIO.

El MCP3008 es un ADC de 10 bits. Eso significa que va a leer un valor de 0 a 1.023 donde 0 es lo mismo que ‘tierra’ y ‘1.023’ es lo mismo que 3,3 voltios.

4.5.1 Habilitación del interface SPI utilizando raspi-config

La imagen Raspbian predeterminada desactiva SPI por defecto, así que antes de que se pueda utilizar la interfaz, debe estar habilitado. Entramos en la configuración, tecleamos en modo terminal el comando `sudo raspi-config` y accedemos en las opciones avanzadas (figura 4.22):

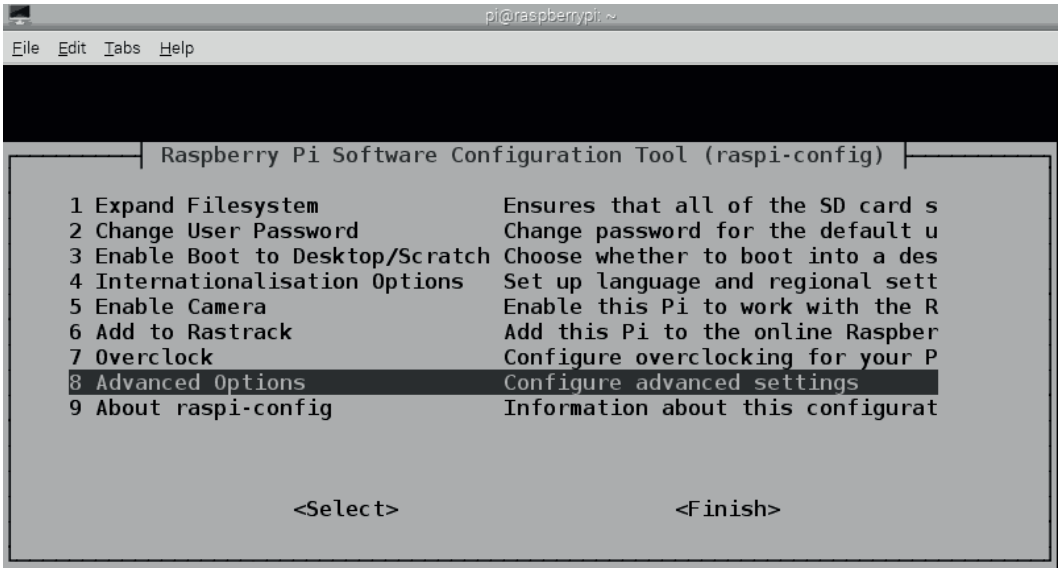


Figura 4.22

Después accedemos a la habilitación del protocolo permitiendo su uso (figura 4.23):

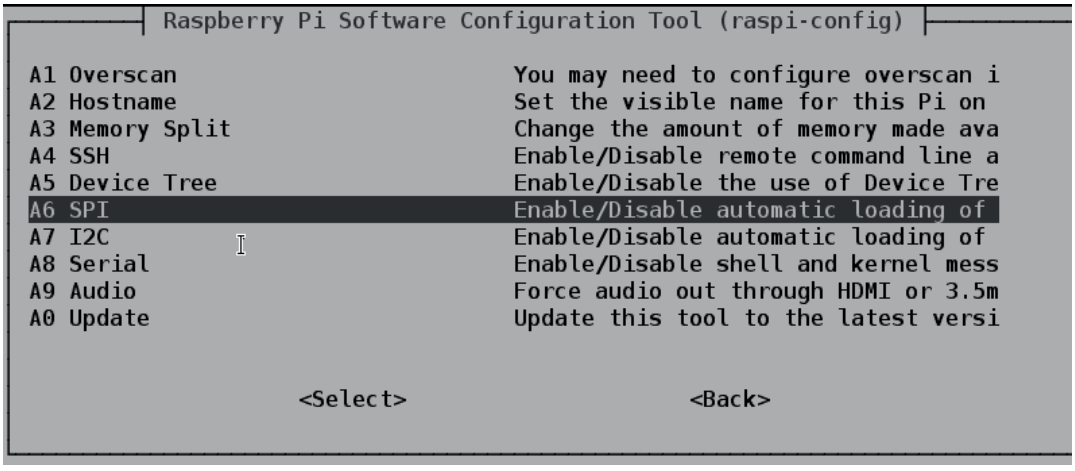


Figura 4.23

Integramos además el módulo dentro del kernel y salimos de la configuración. Finalmente reiniciamos nuestra Raspberry. A continuación comprobamos si el módulo se ha cargado correctamente utilizando el comando **lsmod**. Observaremos si aparece en la salida del terminal el módulo **spi_bcm2708** (figura 4.24).

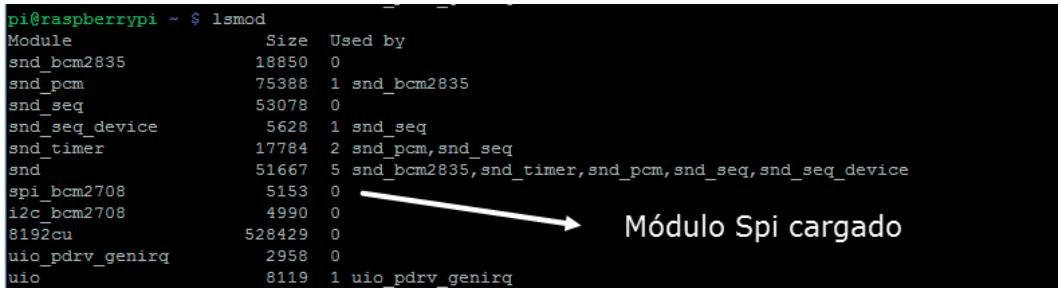


Figura 4.24

4.5.2 Instalación de la envoltura SPI para Python

Con el fin de leer los datos del bus SPI en Python, debemos además instalar una biblioteca llamada **py-spidev**. Para usarla, primero tenemos que instalar **python-dev**.

```
sudo apt-get install python2.7-dev
```

Si obtenemos algún error en la instalación, será necesario ejecutar estos dos comandos y volver a ejecutar la instalación de **python27-dev**.

```
sudo apt-get update
sudo apt-get upgrade
```

Entonces ya podemos descargar **py-spidev** y compilarlo para su uso:

```
wget https://github.com/Gadgetoid/py-spidev/archive/master.zip
unzip master.zip
unzip master.zip
rm master.zip
cd py-spidev-master
sudo python setup.py install
cd ..
```

Instalamos el módulo **rpi.gpio** para acceder a los GPIO.

```
sudo apt-get install python-setuptools
sudo easy_install rpi.gpio
```

Ahora deberíamos estar listos para comunicarnos con dispositivos SPI (por ejemplo, el MCP3008 ADC). El script en Python es relativamente sencillo y se muestra en la figura 4.25.

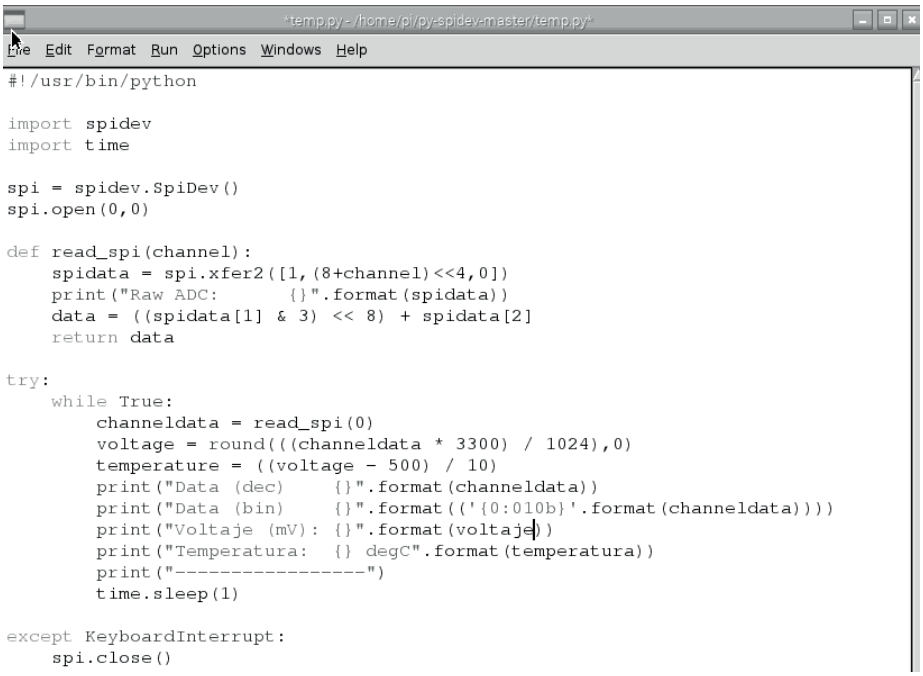


Figura 4.25

4.6 AÑADIENDO UN RELOJ DE TIEMPO REAL A LA Pi2 (DS3231)

En este apartado se muestra cómo agregar un **reloj de tiempo real** (RTC por sus siglas en inglés) a nuestra Raspberry, pero antes de todo ¿qué es un RTC? Un reloj de tiempo real es un circuito integrado que mantiene la hora actual y la guarda para su posterior consulta o manipulación.

¿Y esto para qué nos sirve? Cuando apagamos la Rasp y la vuelves a encender no se conserva la fecha ni la hora actual, aunque la coloquemos correctamente antes de apagarla. Esto se debe a que no se cuenta con un RTC para guardar la hora y la fecha.

En qué casos se puede usar:

- ✓ Medición de pruebas.
- ✓ Medición de eventos con registro de hora y fecha.

Aunque controlar el tiempo puede hacerse sin un RTC, usarla tiene beneficios:

- ✓ Bajo consumo de energía (importante cuando está funcionando con una pila).
- ✓ Libera de trabajo al sistema principal para que pueda dedicarse a tareas más críticas.
- ✓ Algunas veces más preciso que otros métodos.

En primer lugar adquirimos un RTC basada en el integrado DS3231 (más preciso que el clásico DS1307 y de coste parecido). En la figura 4.26 se observa este módulo comprado en www.bricogeek.com.

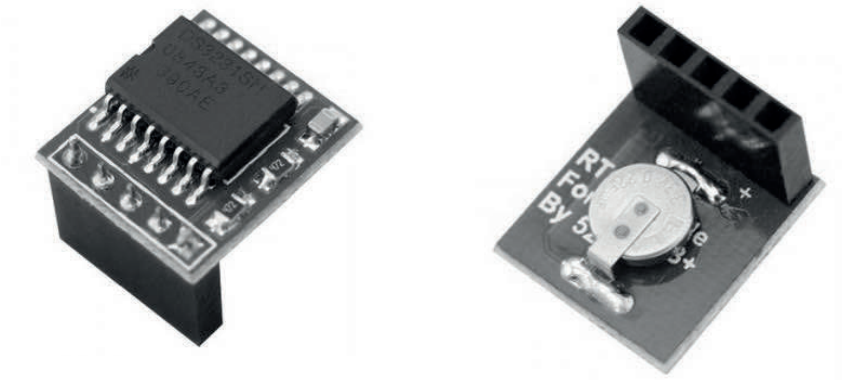


Figura 4.26

La forma de conectarla a nuestra Raspberry se muestra en la figura 4.27:

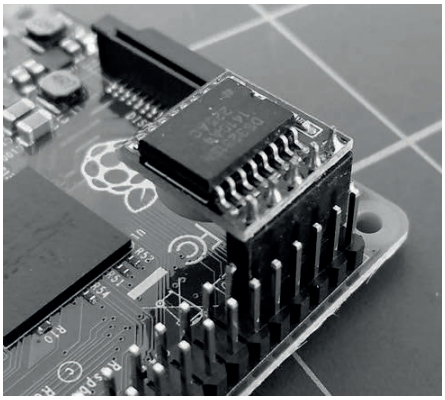


Figura 4.27

La RTC es un dispositivo I2C, por lo que este protocolo permite que varios dispositivos se conecten a nuestra Pi2, y cada uno con una dirección única que puede a menudo ajustarse cambiando la configuración de los puentes en el módulo. Es muy útil ser capaz de ver qué dispositivos están conectados a nuestra Raspberry como una forma de asegurarse de que todo está funcionando. Para ello, es conveniente instalar un paquete de herramientas ejecutando lo siguiente desde una ventana de terminal:

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
```

De igual manera que hicimos en el apartado anterior, habilitamos el protocolo I2C utilizando las opciones avanzadas mediante la utilidad **raspi-config** (figura 4.28).

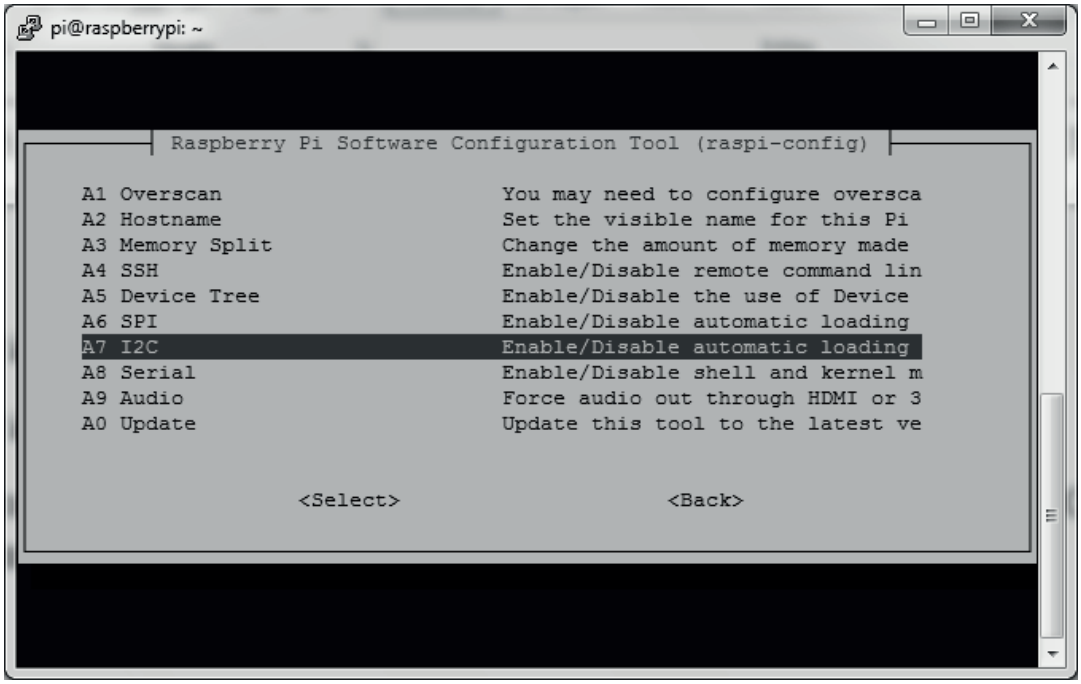


Figura 4.28

También es recomendable pasar por los siguientes pasos para comprobar manualmente todo lo que fue introducido por **raspi-config**.

Editamos el fichero *modules.txt* (`sudo nano /etc/modules`) y añadimos las dos líneas:

```
i2c-bcm2708
i2c-dev
```

Además, editamos el fichero **raspi-blacklist.conf** :

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

y añadimos las dos líneas:

```
blacklist spi-bcm2708
blacklist i2c-bcm2708
```

Finalmente, editamos el fichero *config.txt*:

```
sudo nano /boot/config.txt
```

y le añadimos al final las dos líneas:

```
dtparam=i2c1=on
dtparam=i2c_arm=on
```

Reiniciamos la Raspberry y comprobamos la correcta configuración I2C del RTC (figura 4.29) mediante el comando:

```
sudo i2cdetect -y 1
```

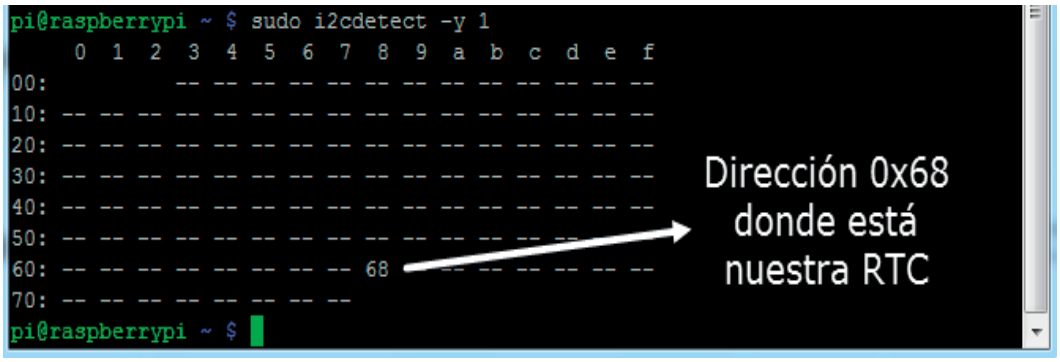


Figura 4.29

Una vez activado hay que notificarle a Linux el nuevo dispositivo que hemos insertado (RTC):

```
echo ds3231 0x68 | sudo tee /sys/class/i2c-adapter/i2c-
1/new_device
```

Testeamos (figura 4.30) que Linux pueda ver el módulo de RTC mediante el comando:

```
sudo hwclock
```

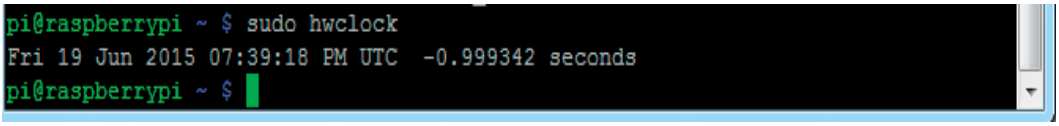


Figura 4.30

Al estar conectado a Internet, actualiza la fecha y hora perfectamente. Ahora, si queremos que desde el arranque nuestra Raspberry detecte nuestro RTC, debemos ejecutar los siguientes comandos, y con esto ya no tendremos que volver a ejecutar los primeros comandos en cada arranque:

```
sudo sed -i 's#^exit 0$#echo ds1307 0x68 > /sys/class/i2c-
adapter/i2c-1/new_device#' /etc/rc.local
echo exit 0 | sudo tee -a /etc/rc.local
```

Para explorar otras opciones del comando **hwclock,** podemos estudiar el contenido de la siguiente web: http://linux.about.com/library/cmd/blcmdl8_hwclock.htm

4.7 PEQUEÑO PROYECTO UTILIZANDO EL EXPLORER HAT PRO

El **Explorer HAT PRO** (figura 4.31) (http://shop.pimoroni.com/products/explorer-hat) es un tipo de shield o mochila para la Raspberry que permite realizar pequeños prototipos o diseños de nuestros proyectos de una forma sencilla, evitando demasiado cableado y simplificando el hardware.

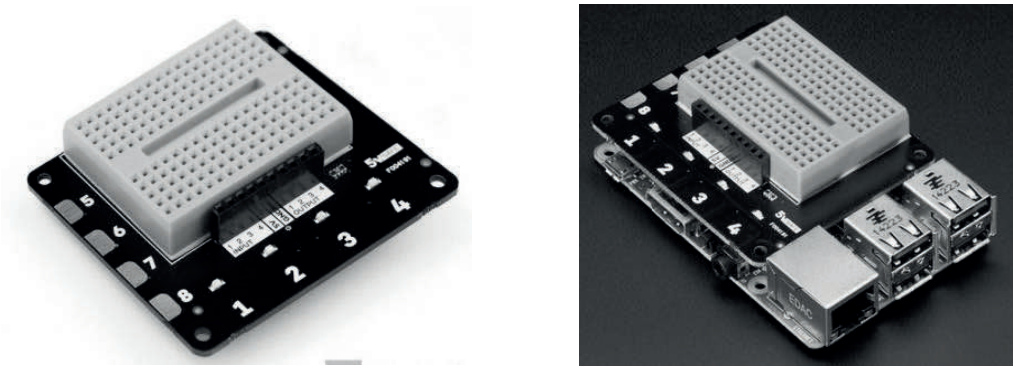


Figura 4.31

Características:

- ✓ Cuatro entradas tolerantes a 5 V con capacidad de buffer.
- ✓ Cuatro salidas de 5 V (hasta 500 mA).
- ✓ Cuatro almohadillas táctiles capacitivas.
- ✓ Cuatro almohadillas capacitivas.
- ✓ Cuatro LEDS de colores.
- ✓ Cuatro entradas analógicas para integrar señales analógicas en tu proyecto.
- ✓ Dos drivers de motores (H-Bridge) para controlar dos motores bidireccionalmente con un máximo de 200 mA por canal. Incluso disponemos de la posibilidad PWM para control de velocidad completa.
- ✓ Una mini breadboard para montar nuestros proyectos electrónicos.

Las entradas tolerantes a los 5 voltios nos permiten aceptar entradas de otros sistemas como Arduino. Disponemos de 5 canales que aceptarán cualquier nivel desde 2 V-5 V como lógico alto. Las salidas de potencia de 500 mA por canal para motores paso a paso, solenoides y relés. Las 8 entradas capacitivas y táctiles son etiquetadas por delante con los números 1, 2, 3 y 4. En un lado de la shield disponemos de las restantes entradas, y son apropiadas para sujetar pinzas de cocodrilo. También tenemos cuatro LEDS de colores (rojo, verde, azul y amarillo).

La relación de todas las patillas de la explorer HAT con los pines de la Raspberry se muestran en la figura 4.32.

LED	GPIO pin
LED 1	GPIO 4
LED 2	GPIO 17
LED 3	GPIO 27
LED 4	GPIO 5

Output	GPIO pin
Output 1	GPIO 6
Output 2	GPIO 12
Output 3	GPIO 13
Output 4	GPIO 16

Input	GPIO pin
Input 1	GPIO 23
Input 2	GPIO 22
Input 3	GPIO 24
Input 4	GPIO 25

Function	Motor	GPIO pin
+	2	GPIO 21
-	2	GPIO 26
+	1	GPIO 19
-	1	GPIO 20

Figura 4.32

Además, tenemos a nuestra disposición una completa librería de Python con amplia documentación y ejemplos.

Este proyecto expondrá cómo combinar dos dispositivos diferentes de salida con los botones capacitivos táctiles de la Pimoroni Explorador HAT PRO para crear un sistema de entrada mediante la combinación correcta de un PIN numérico. Cuando el PIN es correcto se encienden dos LEDS, y cuando es erróneo suena un zumbador.

Vamos a aprender cómo conectar los LEDS de control usando las salidas HAT Explorer y un pequeño zumbador piezoeléctrico mediante el canal PWM (modulación por ancho de pulso).

Material necesario:

- ✓ Pimoroni Explorador HAT.
- ✓ Tres resistencias de 470 Ω resistencias.
- ✓ Dos LEDS de color diferente.
- ✓ Un zumbador piezoeléctrico.

Si es la primera vez que utilizamos el explorador HAT PRO y no está configurado todavía, debemos hacer lo siguiente bajo un terminal o consola.

```
get.pimoroni.com/i2c curl | fiesta
sudo apt-get install python-SMBus
sudo apt-get install python-pip
sudo pip install explorerhat
```

Esos comandos configurarán el protocolo I2C. Además, instalarán la librería de Python específica para trabajar de forma sencilla con el explorador.

A continuación, conectaremos físicamente el explorerador HAT PRO en el conector de 40 pines GPIO de la Raspberry Pi2. Podemos comprobar que está funcionando escribiendo lo siguiente línea en el terminal:

```
sudo python -c 'import time, explorerhat;
explorerhat.light.on();
time.sleep(1);
explorerhat.light.off()'
```

Esto debería iluminar los cuatro LEDS del explorador HAT por un segundo y luego apagarlos a todos de nuevo. Si funciona, entonces nuestro explorador HAT está perfectamente configurado y listo para emprender el proyecto. El diagrama de cableado se muestra en la figura 4.33.

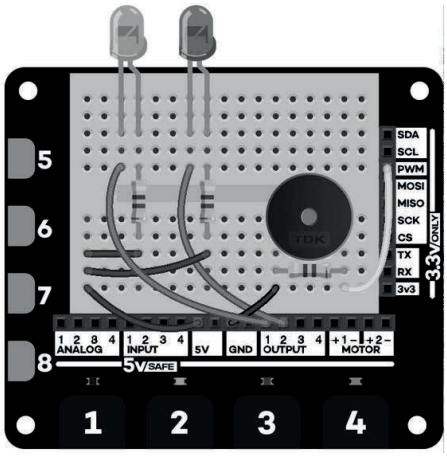


Figura 4.33

Siempre que escribo algo de código para resolver un problema me gusta detallar un plan general de lo que tengo que hacer (lo llaman pseudocódigo). Esto es lo que tenemos que hacer:

- ✓ Establecer un código PIN de cuatro dígitos.
- ✓ Lleve un registro de los números de botón pulsados.
- ✓ Mostrar información visual cuando se pulsan los botones.
- ✓ Cuando se han introducido cuatro números, compruebe si el PIN es correcto.
- ✓ Si el PIN es correcto, parpadeará el LED verde. Establecer un tono alto en el zumbador.
- ✓ Si el PIN es incorrecto, parpadea el LED rojo. Establecer un tono bajo en el zumbador.
- ✓ Mostrar algunos comentarios a la terminal.

El script en Python es el siguiente:

```
import time
import explorerhat as eh
import RPi.GPIO as GPIO
```

Define GPIO pin 18 como una salida PWM con una frecuencia de 400 Hz.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
buzzer = GPIO.PWM(18, 400)
```

Lista para el correcto PIN y otra lista vacía para añadir dígitos.

```
correct_pin = [1,2,3,4]
pin = []
```

Función para agregar un dígito presionado el botón.

```
def add_to_pin(channel, event):
        if channel > 4:  ## Solo usa los canales 1-4
                return
        if event == 'press':
                global pin
                pin.append(channel)
                eh.light[channel-1].on()
                time.sleep(0.05)
                eh.light[channel-1].off()
try:
        while True:
                while len(pin) < 4:
                        eh.touch.pressed(add_to_pin)
                        time.sleep(0.05)
```

```
                if pin == correct_pin:
                        print 'PIN correcto'
                        for i in range(5):
                                buzzer.ChangeFrequency(400)
                                buzzer.start(50)
                                eh.output.one.on()
                                time.sleep(0.1)
                                buzzer.stop()
                        eh.output.one.off()
                                time.sleep(0.1)
                else:
                        print 'PIN incorrecto. Inténtalo de nuevo'
                        for i in range(5):
                                buzzer.ChangeFrequency(50)
                                buzzer.start(50)
                                time.sleep(0.1)
                                buzzer.stop()
                                eh.output.two.off()
                                time.sleep(0.1)
                pin = []
```

Atiende a la combinación de teclas CONTROL-C.

```
except KeyboardInterrupt:
        pass
except Exception:
        pass
```

Limpia los GPIO antes de salir.

```
finally:
        GPIO.cleanup()
```

Las primeras líneas de código que hacen referencia a **import** permiten trabajar con las funciones de tiempo y, por supuesto, utilizamos las propias de la librería **explorerhat** que simplemente nos ahorran tiempo a la hora de escribir código. Por último, necesitamos **RPi.GPIO** para controlar nuestro zumbador con **PWM**.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
buzzer = GPIO.PWM(18, 400)
```

Estas dos primeras líneas, en primer lugar, establecen los números de pines de la configuración BCM del GPIO. En segundo lugar, la configuración del pin GPIO 18 como una salida. El canal de PWM en el explorador HAT está conectado al pin GPIO 18 en la Raspberry Pi; a ese pin conectaremos nuestro timbre. Esta última línea crea un objeto PWM llamado zumbador utilizando la patilla 18 y estableciendo una frecuencia inicial de 400 Hz.

```
def add_to_pin(channel, event):
        if channel > 4:
                return
        if event == 'press':
                global pin
                pin.append(channel)
                eh.light[channel-1].on()
                time.sleep(0.05)
                eh.light[channel-1].off()
```

Creamos una breve función para hacer tres cosas: detectar qué botón ha sido presionado, añadir el número a la lista y hacer parpadear su luz asociada brevemente.

Pasaremos esta función para el método **eh.touch.pressed()** más tarde, y también pasaremos el número de canal que ha sido presionado y qué tipo de evento era. Tenemos que restar 1 del canal, ya que los canales están indexados a partir del 1 y los LEDS del 0.

```
while True:
        while len(pin) < 4:
                eh.touch.pressed(add_to_pin)
                time.sleep(0.05)
```

while True asegura que este bloque de código se mantendrá en funcionamiento hasta que salgamos fuera del programa con la combinación de teclas CONTROL+C. Añadiremos dígitos a la lista siempre y cuando la lista contenga menos de 4 dígitos. La función **eh.touch.pressed (add_to_pin)** se ejecuta siempre que se presione un botón.

```
if pin == correct_pin:
        print 'PIN correcto'
        for i in range(5):
                buzzer.ChangeFrequency(400)
                buzzer.start(50)
                eh.output.one.on()
                time.sleep(0.1)
                buzzer.stop()
                eh.output.one.off()
                time.sleep(0.1)
```

En primer lugar, comprobamos si el pin tecleado coincide con el pin correcto (**correct_pin**). Si lo hace, entonces mostramos un mensaje de aprobación. Entonces tenemos un bucle que se ejecuta en cinco ocasiones. En cada ciclo se establece una frecuencia del zumbador para un tono alto con **buzzer.ChangeFrequency (400)**. Luego se inicia el zumbador con **buzzer.start (50)**. Al mismo tiempo, encendemos el LED verde conectado a la salida 1 con **eh.output.one.on ()**. El **time.sleep (0,1)** significa que la alarma sonará y el LED se iluminará durante 0,1 segundos.

Por último, pararemos el zumbador con **buzzer.stop ()**, apagaremos el LED con **eh.output.one.off ()**. Este bucle se ejecuta en cinco ocasiones, por lo que la luz parpadea y el zumbador suena de forma intermitente cinco veces para un total de 1 segundo.

```
else:
        print ''PIN incorrecto. Inténtalo de nuevo'
        for i in range(5):
                buzzer.ChangeFrequency(50)
                buzzer.start(50)
                eh.output.two.on()
                time.sleep(0.1)
                buzzer.stop()
                eh.output.two.off()
                time.sleep(0.1)
```

Este bloque **else** es esencialmente el mismo que el bloque anterior, excepto que parpadea el otro LED rojo y el zumbador suena a una frecuencia inferior. Evidentemente, se ejecuta cuando el PIN es incorrecto.

El funcionamiento real se puede observar en la figura 4.34:

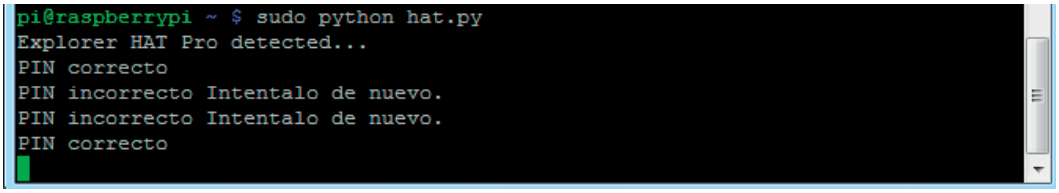


Figura 4.34

4.8 MIDIENDO DISTANCIAS CON EL SENSOR ULTRASÓNICO HC-SR04

Si algún día queremos construir nuestro propio robotito, necesitaremos medir distancias para que el artilugio móvil que hemos creado no choque con todo lo que se encuentre delante.

Para ello podemos utilizar el sensor **PING de Parallax** (figura 4.35). Funciona como un sónar mediante ultrasonidos y es capaz de detectar objetos a una distancia de entre 2 centímetro a 3 metros.

Dispone de un indicador LED y tan solo requiere de un pin para su funcionamiento. Se puede utilizar en una placa de prototipo o directamente en tu robot.

El sensor envía ultrasonidos por un lado y mide el tiempo de rebote del sonido. En su pin de salida podremos medir el ancho de pulso PWM en función de la distancia del obstáculo. Funciona exactamente igual que un radar, de hecho es un pequeño radar. Emite un pulso de sonido a una frecuencia tan alta que es imperceptible para el oído humano y cronometra el tiempo que el sonido tarda en llegar a un obstáculo, rebotar y volver al sensor. Como la velocidad de propagación del sonido en el aire es un dato bien conocido (343,2 m/s) echamos mano de una conocidísima formula ($e = v \cdot t$) y calculamos la distancia recorrida por el sonido.

En la figura 4.35 se observa el aspecto real del sensor que no es precisamente barato. De todas maneras se pueden encontrar "clónicos" a un precio más asequible. En la figura 4.36 vemos el conexionado a nuestra Raspberry.

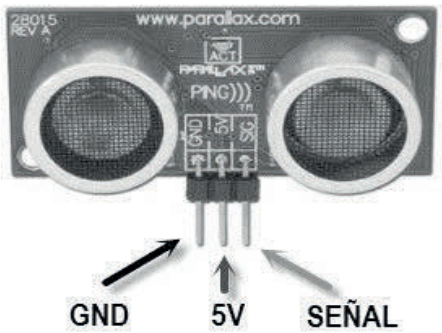


Figura 4.35

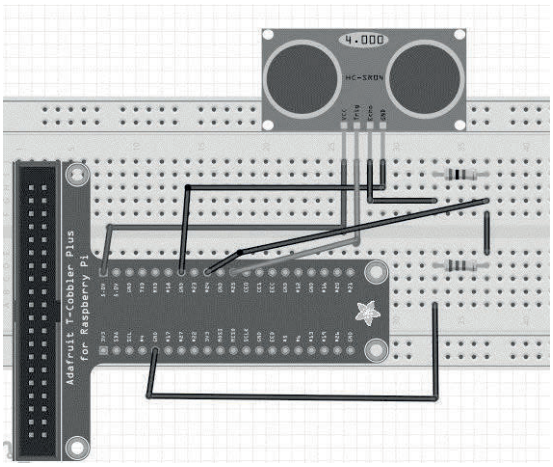


Figura 4.36

Para alimentar el sensor he conectado la patilla VCC del módulo en el pin GPIO de 5 V de la Raspberry Pi2 y la patilla GND a un pin GPIO GND. Con esto ya tendríamos el sensor alimentado. Para comunicar el sensor con la Raspberry hemos conectado la patilla TRIGGER al pin GPIO 23 y la patilla ECHO al pin GPIO 24.

La señal de salida del sensor (ECHO) en el HC-SR04 es de 5 V. Sin embargo, el pin de entrada en la Rasp GPIO tiene un voltaje máximo de 3,3 V. El envío de una señal de 5 V en ese puerto de entrada de 3,3 V sin protección podría dañar los pines GPIO, que es algo que queremos evitar. Tendremos que usar un pequeño circuito divisor de tensión, que consta de dos resistencias, para bajar la tensión de salida del sensor a la tensión de nuestra Raspberry pueda manejar. Por ello añadimos una resistencia de 1 KΩ y otra de 2 KΩ que conforman un divisor de tensión típico.

El funcionamiento del sensor es muy simple y se basa en que la señal ECHO será "bajo" (0 V) hasta que el sensor se activa cuando recibe el pulso de eco. Una vez que un impulso de retorno ha sido localizado, ECHO se establece "alta" (5 V) para la duración de ese pulso. La duración del pulso es el tiempo completo entre el sensor de la salida de un pulso ultrasónico y el impulso de retorno como una detección por el receptor del sensor. Por tanto, nuestra secuencia de comandos de Python debe medir la duración del pulso y luego calcular la distancia de esta.

Como sabemos el tiempo que tarda la señal para viajar a un objeto y volver de nuevo, podemos calcular la distancia utilizando la siguiente fórmula:

$$\text{velocidad} = \frac{\text{espacio}}{\text{tiempo}}$$

La velocidad del sonido es variable, dependiendo del medio de propagación, además de la temperatura de ese medio. Sin embargo, algunos físicos inteligentes han calculado la velocidad del sonido al nivel del mar, por lo que vamos a tomar como base ese dato que es de 343 m/s.

También tenemos que dividir nuestro tiempo por dos, porque lo que hemos calculado anteriormente es en realidad el tiempo que tarda el pulso ultrasónico para viajar hasta el objeto y regresar. Simplemente, solo queremos la distancia hasta el objeto, por lo que dividiremos por dos el tiempo.

El script en Python es fácil de seguir, según lo expuesto anteriormente:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
TRIG = 23
ECHO = 24
print "Medida de distancia en progreso"
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)
GPIO.output(TRIG, False)
print "Esperando que el sensor se estabilice"
```

```
time.sleep(2)
GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False)
while GPIO.input(ECHO)==0:
      pulse_start = time.time()
while GPIO.input(ECHO)==1:
      pulse_end = time.time()
pulse_duration = pulse_end - pulse_start
distancia = pulse_duration * 17150
distancia = round(distance, 2)
print "Distancia:",distancia,"cm"
GPIO.cleanup()
```

El resultado de la ejecución de este script se puede observar en la figura 4.37:

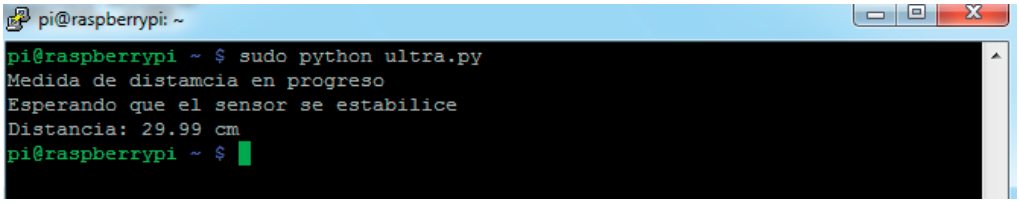


Figura 4.37

Una vista real del proyecto se muestra en la figura 4.38:

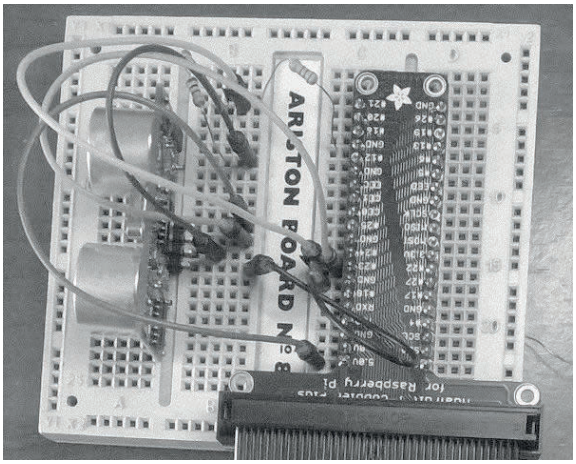


Figura 4.38

CAPÍTULO

5 MOTORES CON LA RASPBERRY Pi2

En este capítulo vamos a dotar de "movimiento" a nuestra Raspberry. Se expondrá cómo programar y controlar los diferentes tipos de motores básicos, o por lo menos los más utilizados en el terreno de la electrónica, y en concreto en la microrrobótica. Se trata de un tema imprescindible cuando nos sintamos predispuestos a construir nuestro propio robotito y deseemos comprender exactamente cómo gobernarlos mediante la programación en Python.

Abordaremos el asunto desde un punto de vista muy práctico. Es decir, nos vamos a centrar principalmente en la descripción y programación de HATS, específicamente diseñadas para este cometido. Hace algún tiempo era imprescindible utilizar electrónica convencional, porque el mundo de la Raspberry no ofrecía shields expresamente dedicadas al control de motores. Hoy en día, los desarrolladores e ingenieros han puesto en el mercado, por un coste asequible, infinidad de mochilas que sin duda, aligeran y simplifican el gobierno de motores con la Raspberry. No solo nos ahorran cableado, sino que nos ofrecen librerías gratuitas para que las usemos de una forma sencilla y podamos evitar la engorrosa electrónica discreta basada en componentes convencionales.

Vamos a empezar con los motores, desde mi punto de vista, más simples de controlar. Estos son los llamados servomotores.

5.1 CONTROL DE LA POSICIÓN DE UN SERVOMOTOR

Un servomotor o servo (figura 5.1) es un motor electrónico de baja inercia al que se le puede controlar tanto la velocidad de giro como la posición dentro de su rango de operación. El cuerpo de los servomotores está formado por un motor eléctrico, una caja reductora con engranajes y un circuito electrónico de control. Los servomotores son comúnmente utilizados en modelismo para controlar los sistemas de dirección; por ejemplo, el timón de un barco o los alerones de un avión. Los servos usan la modulación por ancho de pulsos (PWM) para controlar la posición del motor eléctrico.

Los servos son, sin duda, uno de los dispositivos más útiles para cualquier aficionado a la robótica, ya que nos permiten crear toda clase movimientos de una forma controlada. Lo que normalmente se denomina servo también se conoce como servomotor, motor servo controlado o incluso servorc, y, en cualquier caso, hacen referencia a un sistema compuesto por un motor eléctrico, un sistema de regulación que actúa sobre el motor y un sistema de sensor que controla el movimiento del motor. Es precisamente este sensor el que marca la diferencia con respecto a un motor controlado

electrónicamente, ya que, gracias a la información obtenida del sensor, se puede saber en tiempo real lo que está haciendo un motor.

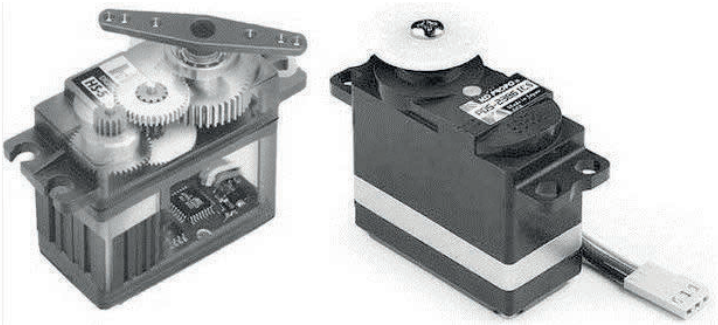


Figura 5.1

Para entenderlo mejor, veamos un ejemplo. Supongamos que tenemos una plataforma giratoria movida por un motor y queremos controlar su giro de forma que la plataforma se pare en determinadas posiciones. Si utilizamos un motor y un circuito de control, la única forma que tenemos de hacerlo sería alimentando el motor durante un tiempo en función del número de grados que queremos que gire. Este sistema, además de ser muy poco preciso, tiene el gran inconveniente de que no sabemos si realmente la plataforma ha girado o no, es decir, que carece de un sistema sensor que retroalimente al circuito de control para que este sepa que es lo que está haciendo la plataforma. Por otra lado, en el momento en que la alimentación del motor varíe un poco, o simplemente la plataforma tenga más o menos carga, la cantidad de giro con respecto al tiempo también varía por lo que el sistema no es práctico.

Para solventar este hecho, se ha creado un sistema de control basado en el ancho de un pulso para controlar la posición del motor. Este pulso normalmente es de 1,5 ms (figura 5.2) y mantiene el servo en la posición centrada. Si el pulso es más corto, por ejemplo, 1 ms, el servo gira a la izquierda; si el pulso es mayor, por ejemplo, 2 ms, el servo gira a la derecha. El movimiento del servo es proporcional al pulso que se le aplica. Otra particularidad que tiene este pulso es su frecuencia de refresco, que en este caso es de 50 Hz, lo que equivale a mandar un pulso de control cada 20 ms.

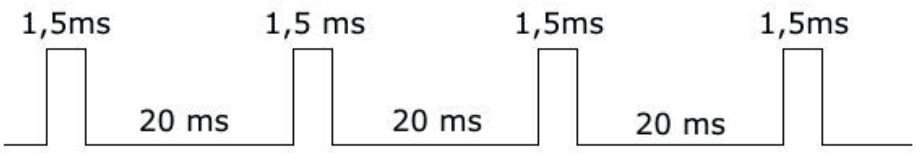


Figura 5.2

Los servos modernos dejan de controlar el motor, tan pronto como se dejan de mandar los pulsos de control. Por eso, para controlar un servo hay que mandar los pulsos de control unas 50 veces

por segundo, con un margen del 20 % aproximadamente. Si pasa más tiempo sin mandar los pulsos, el servo entra en reposo y, como consecuencia, su consumo baja a apenas 8 mA, lo que puede ser muy interesante para algunos proyectos.

Otra particularidad que tienen los servos de radio control es que su movimiento está limitado en la mayoría de los casos a 180 grados. En los sistemas originales controlados vía radio, el rango de movimiento es de 90 grados, es decir, 45 grados hacia cada lado desde la posición central, ya que el ancho del pulso va desde los 900 a los 2.100 milisegundos (figura 5.3). Esto es suficiente para mover los diferentes mandos de los modelos, como son el timón, la dirección, el acelerador, etc. En la práctica, el 95 % de los servos trabajan con pulsos entre los 500 y los 2.500 milisegundos, consiguiendo movimientos de 180-190 grados, aunque todo esto varía ligeramente por arriba y por abajo según diferentes modelos y fabricantes. Esta limitación en el movimiento tuvo como consecuencia que el sistema sensor de la posición se pudiera reducir a un simple potenciómetro, lo que simplificaba el servo desde el punto de vista eléctrico, y además abarata su coste y su peso. El potenciómetro se encuentra conectado mecánicamente al eje de salida del servo, de forma que los dos se mueven a la vez. De ahí la dificultad de conseguir que un servo gire más de 270 grados, ya que mecánicamente está limitado por el giro del potenciómetro.

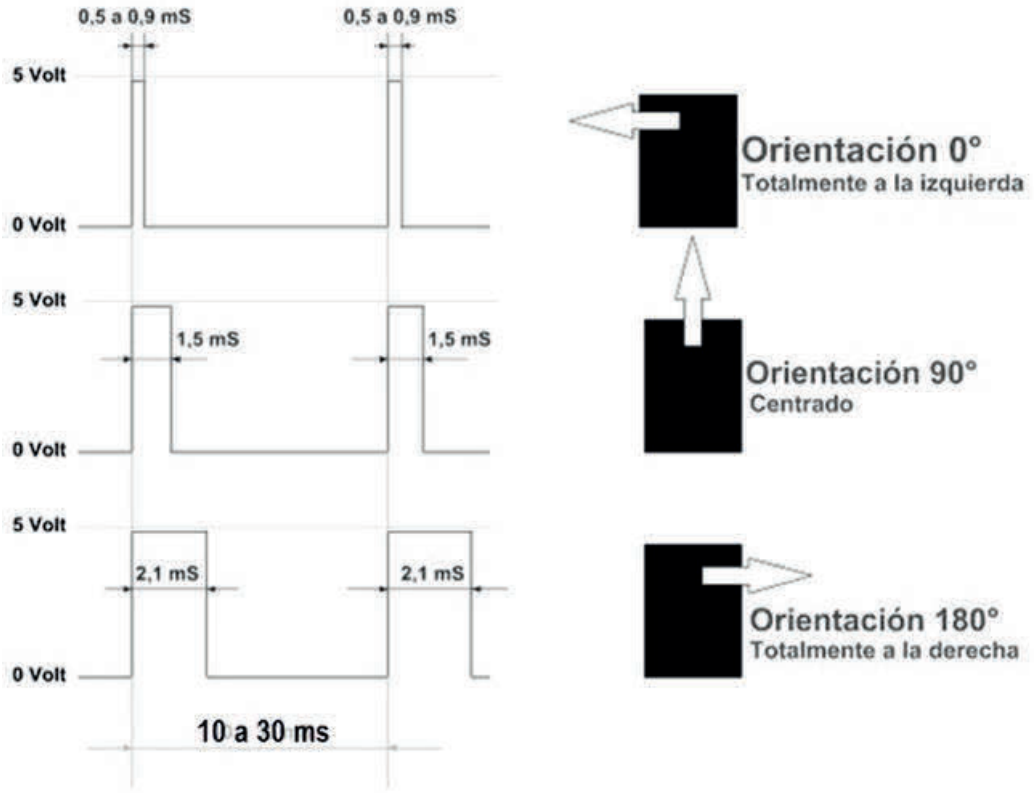


Figura 5.3

Desde el punto de vista eléctrico, el servo es un circuito que se encuentra en equilibrio. El equilibrio se mantiene entre el ancho del pulso de control y la señal que recibe del potenciómetro. Cuando se manda un pulso de control, el equilibrio se descompensa y el circuito mueve el motor y el potenciómetro hasta que consigue que la señal que procede del potenciómetro equilibre de nuevo el circuito. Una vez que el eje alcanza la posición de equilibrio, el motor se para y se mantiene en esa posición. Si se ejerce fuerza externa para intentar mover el eje, el servo reaccionará intentando mantener la posición correspondiente al pulso de control que recibe.

De forma genérica, la mayoría de los servos funcionan con tensiones comprendidas entre los 4,8 V y los 6 V, siendo 6 V la tensión máxima recomendada y en la que se obtiene más potencia, rendimiento y velocidad. Algunos servos admiten 7,2 V, pero no es lo habitual y pueden dañar los servos, por lo que es necesario asegurarse antes de emplear esta tensión.

En cuanto a las conexiones eléctricas (figura 5.4) , casi todos los servos tienen las mismas: negativo, alimentación y señal de control.

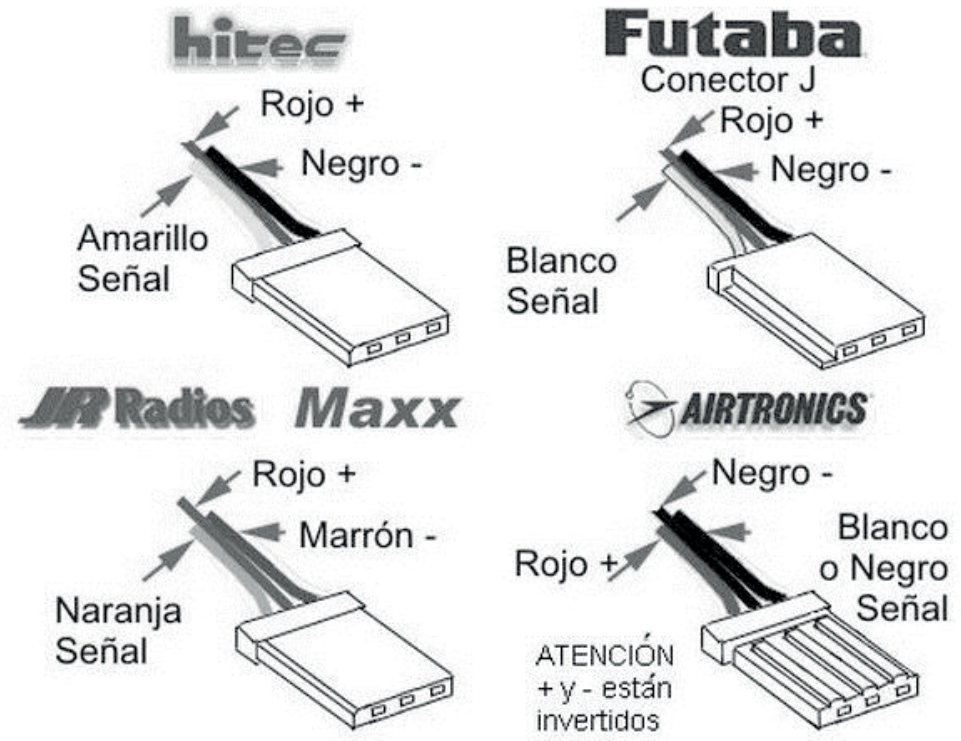


Figura 5.4

Vamos a pasar a la práctica proponiéndonos realizar un control sencillo de la posición de un servo. Utilizamos la técnica PWM para controlar el ancho de impulsos de un motorservo para cambiar

su ángulo de giro. Aunque esto funciona, el PWM generado no es completamente estable, por lo que habrá un poco de fluctuación con el servo. El esquema se muestra en la figura 5.5.

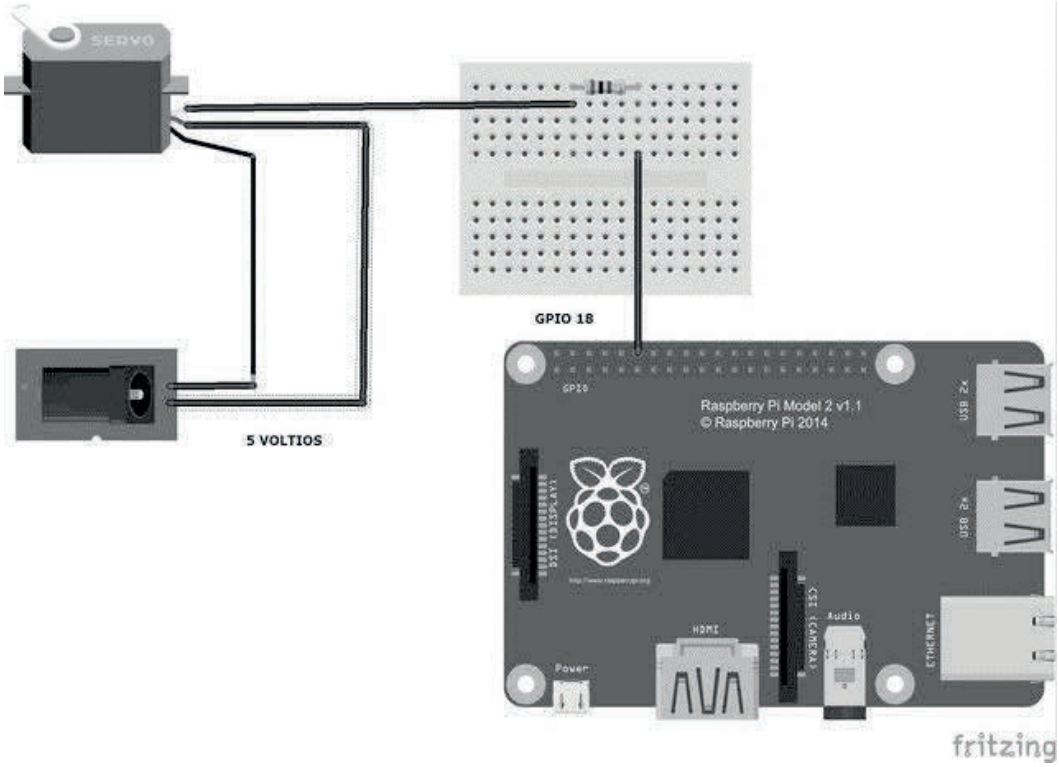


Figura 5.5

La resistencia 1 KΩ no es esencial, pero sí conveniente para proteger el pin GPIO de inesperadas corrientes elevadas de pico en la señal de control. Es necesario alimentar al servo de una fuente de alimentación de 5 V o de un soporte de cuatro celdas de pilas AA que proporcionen 4,8 V.

La interfaz de usuario para ajustar el ángulo de la servo se basa en el programa **gui_slider.py** (figura 5.6), basado en la librería **Tkinter** que veremos más adelante en profundidad.

Figura 5.6

El script bajo Python se muestra a continuación:

```
from Tkinter import *
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
pwm = GPIO.PWM(18, 100)
pwm.start(5)
class App:
      def __init__(self, master):
           frame = Frame(master)
           frame.pack()
           scale = Scale(frame, from_=0, to=180,
           orient=HORIZONTAL, command=self.update)
           scale.grid(row=0)
      def update(self, angle):
           duty = float(angle) / 10.0 + 2.5
           pwm.ChangeDutyCycle(duty)

root = Tk()
root.wm_title('Servo Control')
app = App(root)
root.geometry("200x50+0+0")
root.mainloop()
```

Es necesario tener en cuenta que este programa utiliza una interfaz gráfica de usuario, por lo que no se puede ejecutar desde SSH. Debemos ejecutarlo utilizando conexión VNC desde el entorno de ventanas IDLE con el entorno gráfico de Python.

La línea `from Tkinter import *` figura al principio del script y es la librería que nos ayudará con la creación de **GUI** (interfaces gráficas de usuario) en Python. Ya se encuentra instalada por defecto, por lo que no debemos preocuparnos por descargarla e instalarla. Es una librería multiplataforma lo que permitirá ejecutar nuestros scripts en diversos sistemas operativos como Windows, Linux o Mac.

Como veremos en el capítulo 9, con ella es fácil crear ventanas y trabajar con interfaces gráficas de usuario en Python. Para crear una ventana sin más, importamos tkinter, llamando al método **tk()**, y obtenemos la ventana principal. Para iniciar el bucle de mensajes de la ventana, usamos **mainloop()**. Ello se hace con el siguiente código de tres líneas:

```
import tkinter as tk
root = tk.Tk()
root.mainloop()
```

El script establece la frecuencia PWM a 100 Hz, lo cual enviará un pulso al servo cada 10 milisegundos. El ángulo se convierte en un ciclo de trabajo entre 0 y 100. Esto produce, en realidad, pulsos más cortos que el mínimo esperado de 1 milisegundo y más de 2 milisegundos.

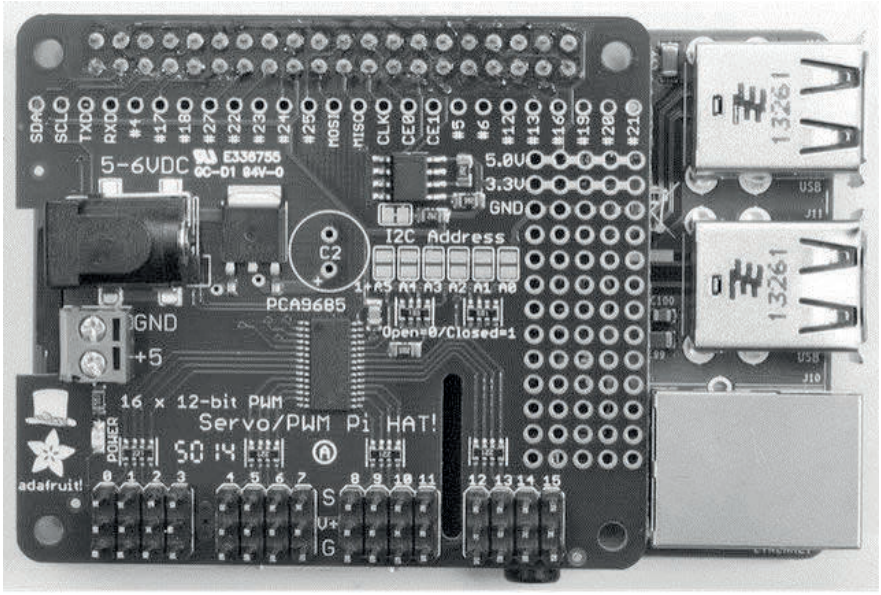
5.2 TRABAJANDO CON LA ADAFRUIT 16-CHANNEL PWM/SERVO HAT

Ya se ha comentado en la introducción a este capítulo de la reciente aparición de una serie de HATS para la Raspberry Pi que nos hacen la vida más fácil. En este caso, vamos a utilizar una plaquita muy versátil que nos permitirá trabajar, ni más ni menos, que con 16 servos a la vez.

Se puede adquirir en el siguiente enlace:

http://tienda.bricogeek.com/expansiones-raspberry-pi/723-adafruit-servo-hat-para-raspberry-pi.html

Esta plaquita de expansión (figura 5.7) para Raspberry Pi permite controlar fácilmente hasta 16 servomotores. La placa tiene 16 salidas y se controla mediante I2C por lo que permite conectar varias placas (u otras) al mismo tiempo a los pines GPIO. Podríamos conectar un total de 62 placas y 992 servos con tan solo 2 pines.



HAT montada sobre la PI2

Figura 5.7

Es compatible con Raspberry Pi2, pero también puede ser utilizada con modelos anteriores de la Rasp. Hay que tener en cuenta que los servos pueden consumir mucha corriente, así que es mejor utilizar una fuente externa para alimentarlos. La placa dispone de una completa librería para Python que te permitirá hacerla funcionar en cuestión de minutos.

Características:

- ✓ Canales: 16.
- ✓ Interfaz: I2C.
- ✓ Alimentación externa.

Esta HAT tiene dos posibles entradas de alimentación. Una de ellas es el pin de 3,3 V de la Pi2, que se utiliza para alimentar el chip de PWM y determina el nivel lógico I2C y el nivel lógico de la señal PWM. Esta posibilidad siempre estará disponible si la Pi2 está alimentada (comprobación del LED rojo). La otra opción es conectar la alimentación a través de un conector tipo jack 2,1 mm para proporcionar 5 o 6 voltios, los cuales son imprescindibles en servos convencionales y no en microservos. Existe una protección de polaridad inversa en caso de que se conectara al revés la alimentación pero, sin embargo, no debemos conectar ambas alimentaciones a la vez, pues corremos el riesgo de dañar el hardware.

Los servos puedes utilizar una gran cantidad de corriente. Por ello no es una buena idea usar el pin de 5 V de la Raspberry Pi para alimentarlos. El ruido eléctrico y las caídas de tensión podrían causar que nuestra Pi2 actué de forma errática o se sobrecaliente.

Regla de oro: Mantened completamente independientes la fuente de alimentación de la Pi y la fuente de alimentación del servo.

Como hemos visto anteriormente la mayoría de los servos vienen con un conector hembra de 3 pines estándar. Este se conecta a las cabeceras de la HAT Servo. Es necesario asegurarse de alinear el enchufe con el cable de tierra (por lo general, de color negro o marrón) con la fila inferior, y el cable de señal (generalmente de color amarillo o blanco) en la parte superior (figura 5.8).

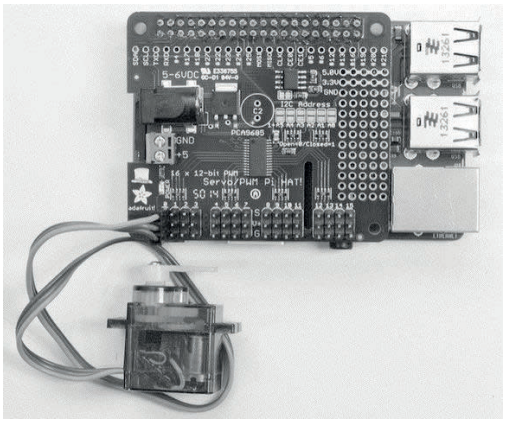


Figura 5.8

Una vez que hemos conectado el servo y lo hemos alimentado, podemos configurar el protocolo I2C para comunicar la HAT con la Pi2.

Seguimos el mismo procedimiento explicado en el apartado 3.4 y que resumimos a continuación:

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
sudo i2cdetect -y 1
```

La última línea del código anterior buscará un dispositivo I2C (/dev/i2c-1) recorriendo todas las direcciones. Si la HAT de servos está conectada correctamente, nos mostrará la dirección por defecto (0x40). Obtenemos la salida por terminal mostrada en la figura 5.9:

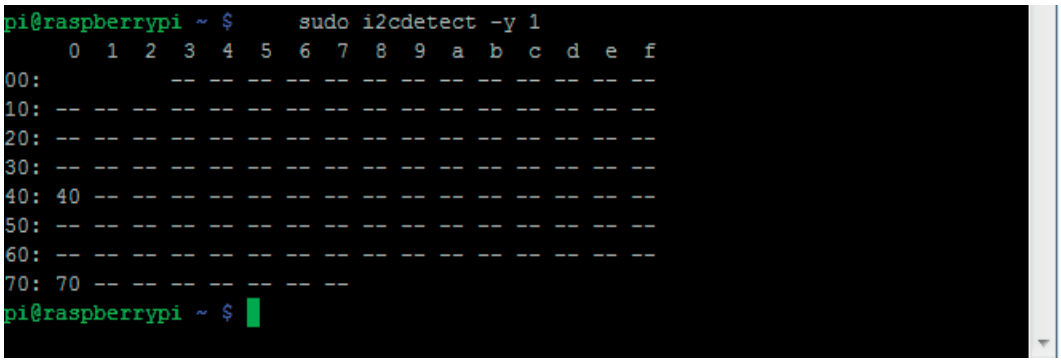


Figura 5.9

Una vez que el hardware está configurado perfectamente, abordamos el tema del software. Primeramente, nos descargamos la librería en Python que nos proporciona el fabricante de forma gratuita. Para ello ejecutamos desde SSH lo siguiente:

```
git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-
Python-Code.git
cd Adafruit-Raspberry-Pi-Python-Code
cd Adafruit_PWM_Servo_Driver
```

Una vez que el código ha sido descargado en una carpeta adecuada y tenemos la HAT Servo con el motor conectados correctamente, podamos probarlo con el siguiente comando:

```
sudo python Servo_Example.py
```

El servo girará en un sentido durante un tiempo y en otro durante ese mismo tiempo. Analicemos el script en Python de este ejemplo. Con la primera definición configuramos el pulso PWM que le enviamos al servo. Después establecemos la frecuencia a 60 Hz y, por último, entramos

en un bucle continuo donde cambiamos la velocidad del motor. Para comprender en profundidad cada una de las funciones, podemos consultar la documentación de la librería.

```
# Importamos la librería propia de la HAT

from Adafruit_PWM_Servo_Driver import PWM
import time

# Inicializamos el dispositivo PWM utilizando la dirección por defecto

pwm = PWM(0x40)
servoMin = 150
servoMax = 600

def setServoPulse(channel, pulse):
        pulseLength = 1000000
        pulseLength /= 60
        pulseLength /= 4096
        pulse *= 1000
        pulse /= pulseLength
        pwm.setPWM(channel, 0, pulse)
        pwm.setPWMFreq(60)

while (True):

 # Cambiamos la velocidad del servo en el canal 0

        pwm.setPWM(0, 0, servoMin)
        time.sleep(1)
        pwm.setPWM(0, 0, servoMax)
        time.sleep(1)
```

El script completo se muestra en las figura 5.10.

5.3 TRABAJANDO CON LA ADAFRUIT DC/STEPPER HAT

Con esta placa de expansión (figura 5.10), podemos controlar hasta 4 motores de corriente continua o 2 motores paso a paso.

Dado que la Raspberry Pi2 no dispone de muchas salidas PWM, esta placa utiliza el chip específico **TB6612** que permite una salida de 1,2 A por canal (3 A de pico), suficiente para la mayoría de motores. Además, el control de la placa se realiza por I2C, lo que permite conectar varias placas de expansión al mismo tiempo, simplemente seleccionando de forma correcta una dirección I2C mediante los jumpers para soldar disponibles en la placa. Los jumpers permiten seleccionar hasta 32 direcciones

distintas, así que podrías conectar hasta 32 placas simultáneamente y controlar 64 motores paso a paso o 128 motores DC.

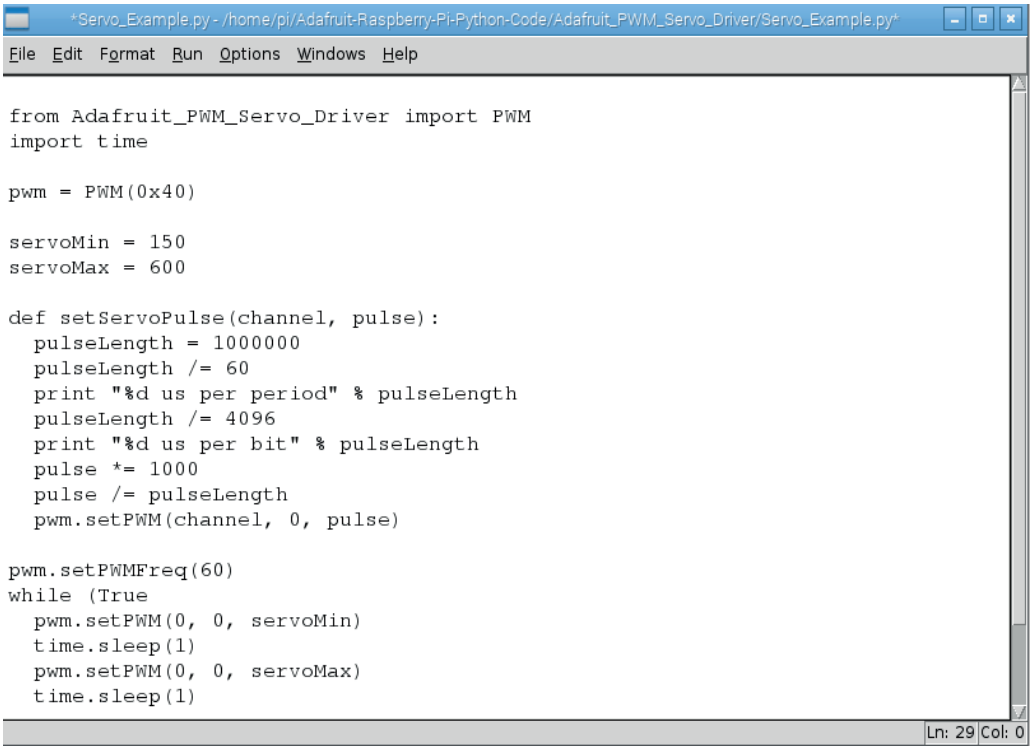


Figura 5.10

Características:

- ✓ Controlador TB6612 con cuatro salidas en puente H. 1,2 A por canal (3 A pico) con protección de temperatura. Funciona con motores de entre 4,5 V y 13,5 V.
- ✓ Control de hasta 4 motores DC con inversión de sentido y control de velocidad digital de 8 bits.
- ✓ Control de hasta 2 motores paso a paso (unipolar o bipolar) con soporte para micro-stepping.
- ✓ Terminales de salida para conexión fácil de motores y alimentación externa.
- ✓ Terminal de entrada de alimentación con protección de polaridad (5-12 V).
- ✓ Dispone de una librería gratuita en Python con ejemplos.

Se puede adquirir en el siguiente enlace:

http://tienda.bricogeek.com/expansiones-raspberry-pi/722-adafruit-dc-stepper-hat-para-raspberry-pi.html

5.3.1 Motores eléctricos de corriente continua (DC) con la Adafruit DC/Stepper Hat

Los motores eléctricos de corriente continua (DC) se pueden encontrar en una amplia gama de dispositivos, incluyendo los coches y barcos de radio control, reproductores de DVD, ventiladores eléctricos, etc. Muchos de estos pueden ser reutilizados para el uso con nuestra Raspberry (decididamente, hay que rebuscar en el trastero…).

Alternativamente, podemos comprarlos por módicos precios en tiendas de electrónica o a través de Internet. Son pequeños motores (figura 5.11) cuyas tensiones de trabajo varían entre 2 V y 30 V. Cada fabricante proporciona una tensión recomendada que si la sobrepasamos (suelen ser bastantes "duros"), podemos quemar el motor. Si por el contrario, aplicamos poca tensión, el motor ni se entera, es decir, no gira.

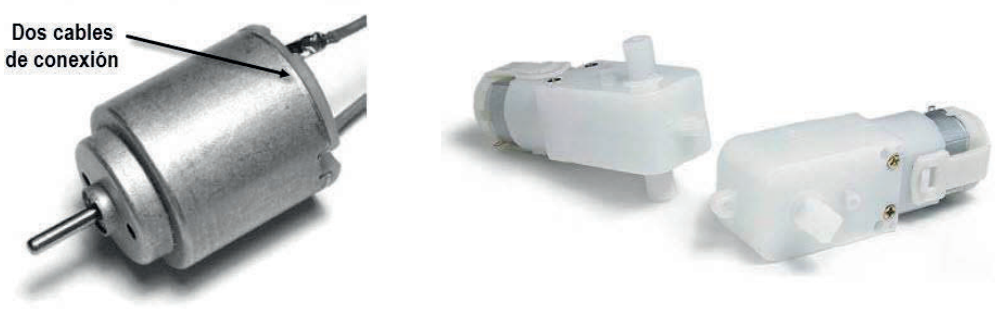


Figura 5.11

El control de estos motores se realiza mediante dos cables. La velocidad del motor se establece con la tensión que se le aplica. Una variación de tensión producirá un cambio proporcional de su velocidad de giro. Para invertir el giro, simplemente se debe invertir la polaridad de los terminales. Si estamos usando el motor DC para accionar un pequeño robot, a menudo se acopla su eje a una caja reductora mecánica (figura 5.11), ya que permite trasformar la alta velocidad de giro a baja velocidad, pero con un par de fuerza más alto de la que por defecto proporciona el motor solo. Esto es adecuado para potenciar la fuerza de los motores, aunque sea a costa de su velocidad de giro. Por otra parte, la Raspberry solo puede proporcionar una pequeña cantidad de corriente, insuficiente para excitar las bobinas del motor DC. Por ello se hace imprescindible aumentar o amplificar dicha corriente si deseamos que nuestro motor se mueva. Podríamos utilizar un puente H como el famoso **L293d** o el **L298**; incluso transistores Mosfet para esto. Sin embargo, desde mi punto de vista, recomiendo utilizar alguna HAT para este propósito.

Las características de un pequeño motor DC típico se pueden resumir en los siguientes puntos:

- ✓ La tensión de trabajo: Esta puede variar desde 3 V a más de 12 V.
- ✓ La corriente sin carga: Es la cantidad de corriente que el motor absorbe cuando gira libremente sin nada conectado.
- ✓ Par de fuerza del eje del motor.

- ✓ La cantidad de corriente que absorbe el motor cuando tiene carga en el eje
- ✓ La velocidad a la tensión de trabajo expresada en revoluciones por minuto (RPM).

Para abordar la utilización de esta HAT (figura 5.12) con los motores DC es necesario tener en cuenta dos aspectos importantes:

- **Requerimientos de voltaje**: La primera cosa importante es averiguar el voltaje del motor que se va a utilizar. Si tienes suerte, el motor posiblemente viene con algún tipo de especificaciones. Algunos motores pequeños solo son destinados para funcionar a 1,5 V, pero es común que funcionen con un rango entre 6-12 V. Los controladores de motor en esta HAT están diseñados para funcionar entre 5 V y 12 V. La mayoría de motores entre 1,5 y 3 V no funcionarán o incluso podrían dañarse con esta mochila.

- **Requerimientos de corriente**: La segunda cuestión a tener en cuenta es la cantidad de corriente que necesitará nuestro motor. Los chips del controlador del motor que vienen con la HAT están diseñados para proporcionar hasta 1,2 A por motor, con picos de corriente de 3 A como máximo.

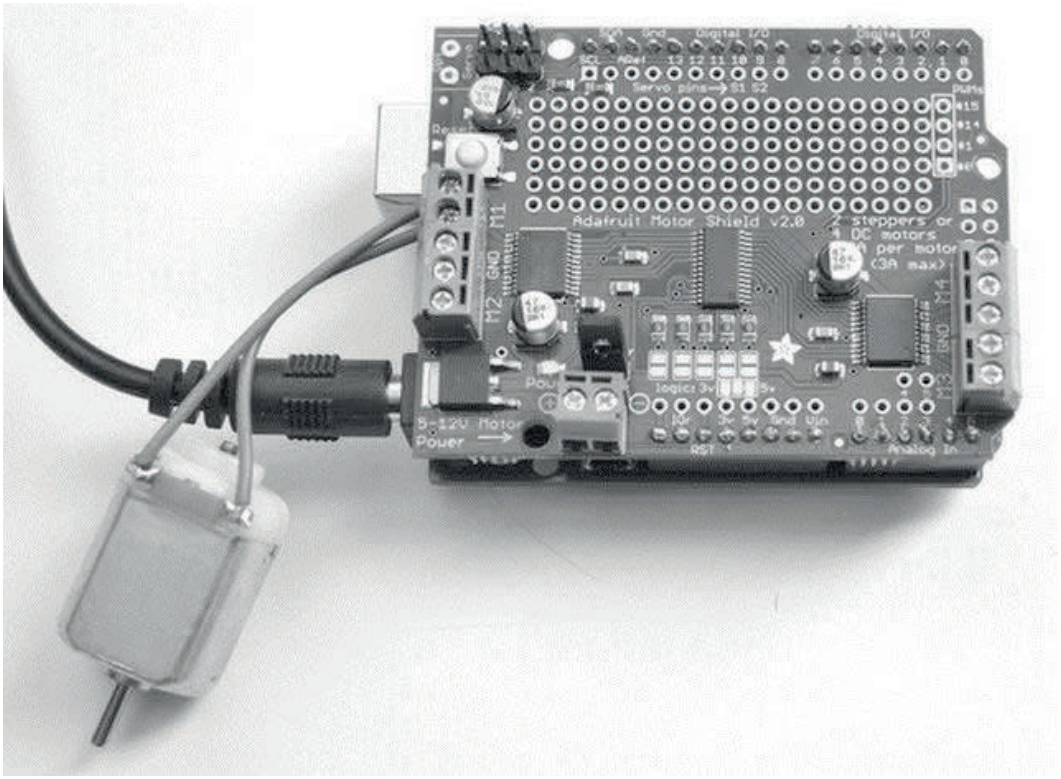


Figura 5.12

Es recomendable tener dos fuentes de alimentación separadas; una para la Pi2 y otra para los motores. Para instalar la librería que proporciona el fabricante debemos descargarla de su propio enlace e instalarla ejecutando estas tres líneas en un terminal:

```
git clone https://github.com/adafruit/Adafruit-Motor-HAT-
Python-Library.git
cd Adafruit-Motor-HAT-Python-Library
sudo python setup.py install
cd examples
```

Vamos a ver un script en la carpeta "examples" recién creada, que muestra todo lo que la biblioteca MotorHAT puede hacer.

Comenzamos con la importación de estas bibliotecas:

```
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_DCMotor
import time
import atexit
```

La biblioteca MotorHAT contiene algunas clases diferentes, uno es la propia clase MotorHAT que es el principal controlador de PWM. Siempre tendremos que crear un objeto y establecer la dirección. Por defecto, la dirección es 0x60.

```
mh = Adafruit_MotorHAT(addr=0x61)
```

El controlador PWM es *free running*, que significa que si el código Python falla, o incluso la Pi2, el controlador PWM aún siguiría trabajando. Esto es muy positivo porque permite que el procesador de la Rasp se ocupe de otras tareas mientras que el driver de PWM se ocupe de su cometido. Aunque esto presenta el inconveniente de que los motores no se detengan cuando el script Python se cierre.

Por esa razón, se recomienda encarecidamente que el código del programa se ocupe de apagar los motores cuando finalicemos su ejecución.

```
def turnOffMotors():
    mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)
    atexit.register(turnOffMotors)
```

Para crear el objeto motor, podemos solicitarlo al objeto MotorHAT creado anteriormente con `getMotor (num)` con un valor entre 1 y 4, según donde lo hayamos conectado físicamente en la HAT (recordad que podemos manejar hasta cuatro motores DC).

```
myMotor = mh.getMotor (3)
```

Los motores de corriente continua son bastante simples; básicamente, solo hay que ajustar su velocidad y su dirección. Para ajustar la velocidad, utilizamos la función **SetSpeed** (velocidad) donde la velocidad varía de 0 (*off*) a 255 (máximo). Este es el ciclo de trabajo PWM del motor:

```
myMotor.setSpeed(150)
```

Para configurar la dirección, utilizamos las opciones siguientes:

- ✓ Adafruit_MotorHAT.FORWARD: motor de corriente continua gira hacia adelante.
- ✓ Adafruit_MotorHAT.BACKWARD: motor de corriente continua gira hacia atrás.
- ✓ Adafruit_MotorHAT.RELEASE: motor de corriente continua "*off*" (desactivado).

En la parte final del script hacemos que el motor gire en un sentido con una velocidad progresiva en ascenso para después hacerlo en descenso. A continuación gira en sentido contrario con la misma progresión de velocidad.

```
while (True):
     print "Sentido Giro Directo "
     myMotor.run(Adafruit_MotorHAT.FORWARD)
     print "Velocidad ascendente"
     for i in range(255):
         myMotor.setSpeed(i)
         time.sleep(0.01)
         print "Velocidad descendente"
     for i in reversed(range(255)):
         myMotor.setSpeed(i)
         time.sleep(0.01)
         print "Sentido de Giro contrario"
         myMotor.run(Adafruit_MotorHAT.BACKWARD)
         print "Velocidad ascendente"
     for i in range(255):
         myMotor.setSpeed(i)
         time.sleep(0.01)
         print "Velocidad descendente"
     for i in reversed(range(255)):
         myMotor.setSpeed(i)
         time.sleep(0.01)
         print "apagado"
          myMotor.run(Adafruit_MotorHAT.RELEASE)
          time.sleep(1.0)
```

El script completo se muestra en la figura 5.13.

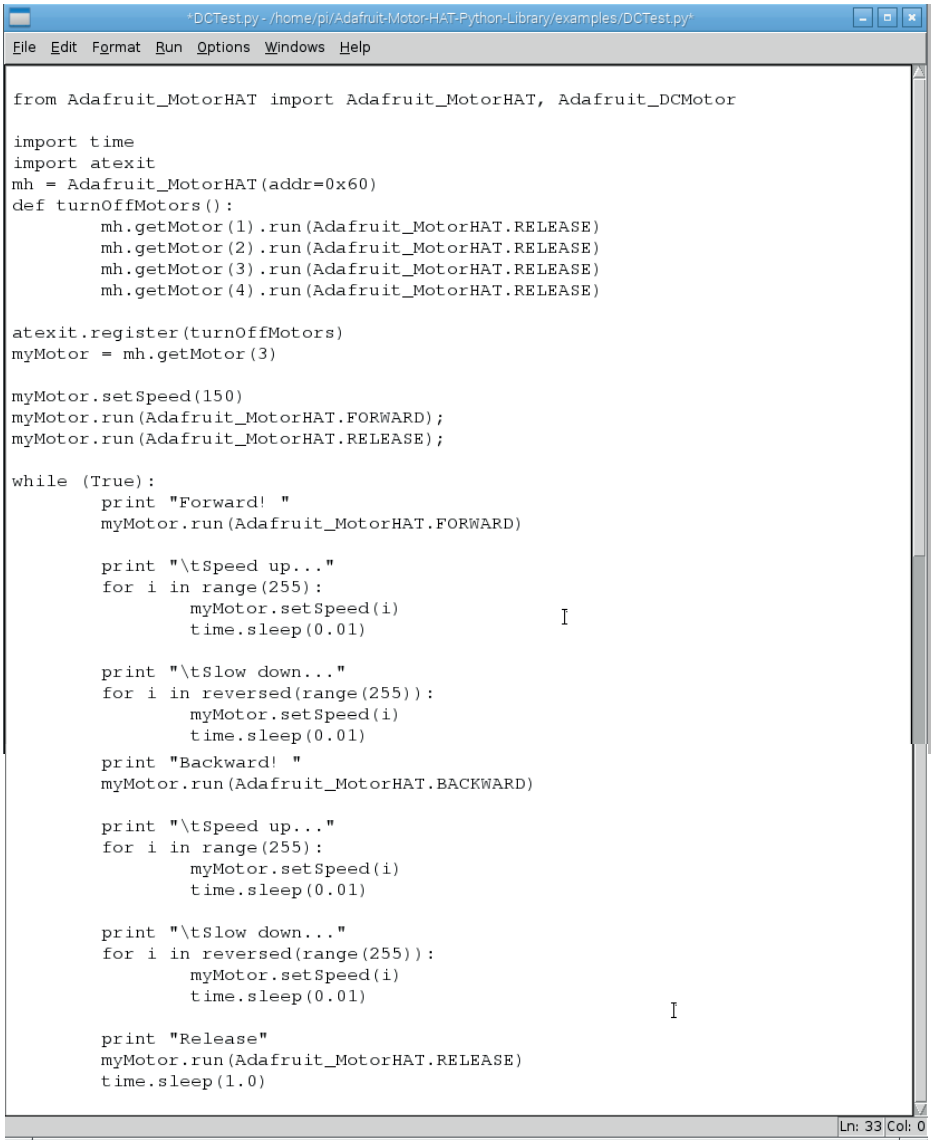


Figura 5.13

5.3.2 Motores paso a paso con la Adafruit DC/Stepper Hat

Un motor paso a paso es un tipo especial de motor que puede moverse en una serie de pasos discretos. Los motores paso a paso son una buena opción para los proyectos que requieren un movimiento controlado y preciso. Los proyectos típicos incluyen las impresoras 3D, sistemas de posicionamiento del telescopio, control numérico por computadora (CNC) de tornos, etc.

Como se comentó anteriormente, se pueden obtener motores paso a paso de viejas impresoras de inyección de tinta o de las impresoras láser. En ellas, estos motores se usan para mover los cabezales de impresión y para controlar la alimentación del papel.

En la figura 5.14 se puede observar el aspecto del motor paso a paso. Los motores paso a paso se pueden clasificar en términos de la torsión que pueden proporcionar. Como el par es proporcional a la longitud del cuerpo, cuanto más largo sea el cuerpo, mayor será el par de torsión que pueden desarrollar.

El ángulo de paso es también una forma de clasificarlos. Un motor paso a paso con un ángulo de 9 grados requerirá 40 pasos para completar una vuelta completa. El par o torque es una medida de la fuerza de rotación que un motor puede proporcionar. A menudo se expresa en onzas-pulgadas o en kg/cm.

El PAP de la figura 5.14 es un tipo NEMA 17 bipolar, que tiene un ángulo de paso de 1,8º (200 pasos por vuelta) y cada bobinado requiere entre 1,2 A y 4 V. Es capaz de cargar con 3,2 kg/cm (44 oz-in).

Figura 5.14

Existen dos tipos principales de motores paso a paso: bipolares y unipolares. Cada uno con sus propias ventajas y desventajas. Echemos un vistazo a las diferencias entre ellos en la tabla 5.1.

La elección de un motor paso a paso puede ser un poco complicada dependiendo de su uso previsto. Para los proyectos que requieran un alto par extremadamente preciso, selecciona uno bipolar; para proyectos más simples, el tipo unipolar es más barato y es una buena opción. Aunque hoy en día, los motores bipolares se están volviendo más populares debido a la reducción en el costo de los integrados necesarios para su control.

Unipolares	Bipolares
Más sencillo de controlar.	Más eficiente.
Menor coste.	Mayor torque.
Cinco o seis conexiones de los cables.	Cuatro conexiones de los cables.
Mayor velocidad de rotación.	Construcción más simple.

Tabla 5.1

Echemos un vistazo a los pasos necesarios para identificar un tipo de motor PAP y las conexiones del mismo. Si observamos el PAP que utilicé en mi impresora 3D, averiguo a través de las características proporcionadas por el fabricante que posee un ángulo de paso de 1,8º y que requiere 1,2 amperios. Como tiene un ángulo de paso de 1,8º, podemos calcular el número de pasos para completar una vuelta al dividir 360º por ese ángulo:

360 / 1,8 = 200 pasos

Sobre la base de este cálculo, el motor paso a paso necesita 200 pasos para completar una revolución. Además el motor cuenta con cuatro cables de conexionado, por lo que deduzco que es de tipo bipolar. Por otra parte, el color de dichos cables es fundamental para conocer a qué bobinas están conectadas. En la figura 5.15 se observa el esquema interno de este motor bipolar y la disposición de sus dos bobinas con los colores asociados a cada terminal.

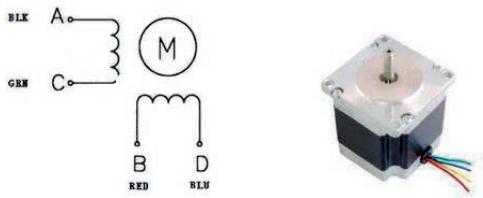


Figura 5.15

En esta estupenda página disponemos de un tutorial muy clarito que explica la teoría de dichos motores PAP: http://www.monografias.com/trabajos37/motores/motores.shtml. En ella se nos muestra una técnica sencilla basada en un polímetro para averiguar la polaridad de los terminales. La usaremos cuando no dispongamos de esta información clave del motor porque la impresora que hemos reciclado es muy vieja.

Pero no siempre es adecuado utilizar PAP bipolares. Podemos usar los PAP unipolares mucho más baratos y más fáciles de encontrar en viejos aparatos olvidados en el trastero. Vamos a verlos.

Los motores paso a paso unipolares básicamente se componen de dos bobinas con una derivación en el centro. Las derivaciones del centro son llevadas fuera del motor como dos cables separados o conectados entre sí internamente. Como resultado de esto, los motores unipolares tienen 5 o 6 cables. Independientemente del número de cables, los motores unipolares son manejados de la misma manera. El cable de toma central (1,2) está ligado a una fuente de alimentación y los extremos de las bobinas son llevados alternativamente a tierra.

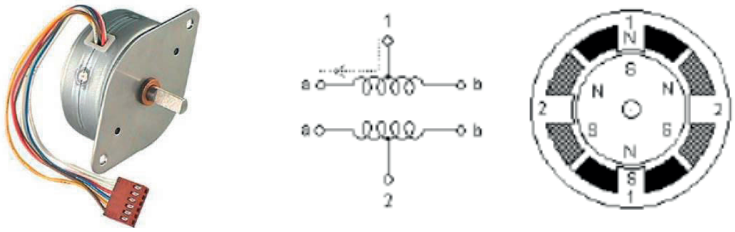


Figura 5.16

La figura 5.16 muestra la sección transversal de un motor paso a paso unipolar de 30º. El bobinado número 1 del motor se distribuye entre los polos de la parte superior e inferior del estátor, mientras que la bobina número 2 del motor se distribuye entre el polo izquierdo y derecho del rotor. El rotor es un imán permanente con seis polos, tres al norte y tres al sur, como se muestra en la figura anterior. Para hacer girar un motor unipolar se aplican impulsos en secuencia a sus devanados. La secuencia de estos impulsos se aplica con un controlador electrónico externo como nuestra Raspberry. Los motores unipolares se pueden hacer avanzar a frecuencias de audio, lo que les permite girar muy velozmente. Este es el motivo por el que se suele decir que un motor "gruñe", debido a la frecuencia a la que se produce la conmutación.

La idea al manejar un motor paso a paso unipolar es ir "encendiendo" y "apagando" las bobinas de a pares siguiendo una determinada secuencia, como se muestra en la tabla extraída de http://www.todorobot.com.ar/informacion/tutorial%20stepper/stepper-tutorial.htm.

Nuestra HAT (figurar 5.17) soporta hasta 2 motores paso a paso. La librería de Python funciona de forma idéntica para motores bipolares que para los unipolares. Su utilización es un poco más compleja de lo que era la de un motor de corriente continua.

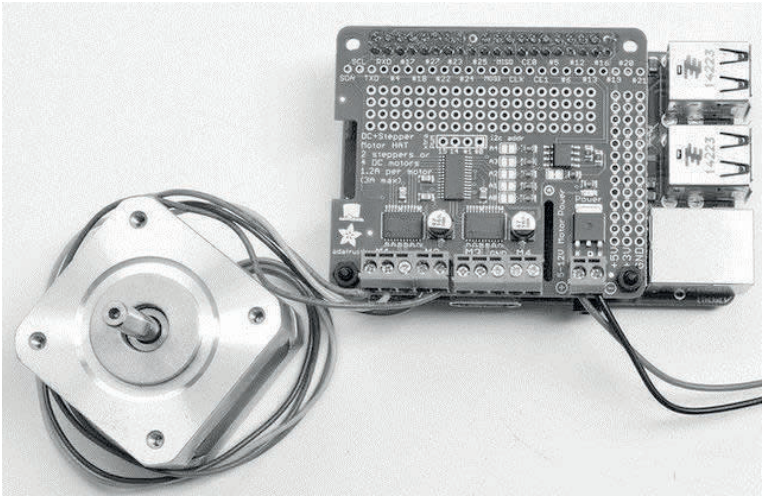


Figura 5.17

A la hora de conectar los motores unipolares a nuestra HAT es necesario, primero, averiguar la correspondencia de sus pines con sus bobinas y qué pines son comunes o de toma central. Como se comentó antes, existen un montón de tutoriales en línea sobre cómo realizar ingeniería inversa para averiguar el conexionado interno del motor. Los pines comunes deben estar conectados entre sí y, además, deben llevarse al terminal GND de la HAT. A continuación, la bobina 1 debe conectarse a un puerto del motor (M1 o M3) y la bobina 2 debe conectarse con el otro puerto del motor (M2 o M4).

Para los motores bipolares se sigue el mismo procedimiento que para los motores unipolares, excepto que no hay que llevar ningún terminal a tierra (cuatro hilos). El código es exactamente el mismo.

Para ejecutar el próximo script debemos conectar una bobina del motor (2 terminales) al puerto **M1-M2** y la otra bobina (2 terminales) al puerto **M3-M4** de la HAT.

Como ya hemos instalado la librería de Ardafuit en el apartado anterior, no hace falta volverlo hacer. Por ello nos vamos a la carpeta "examples" y ejecutamos el script llamado **StepperTest.py**. Observaremos cómo el motor se mueve en un sentido u otro con pasos muy pequeños, solo apreciables si tocamos con los dedos el eje del motor.

Las primeras líneas del código son clásicas: importamos las librerías necesarias.

```
from Adafruit__MotorHAT, import adafruit_MotorHAT,
adafruit_DCMotor, Adafruit_StepperM$
import time
import atexit
```

La biblioteca MotorHAT contiene algunas clases diferentes, una es la propia clase MotorHAT que es la principal del controlador PWM. Siempre tendremos que crear un objeto y establecer su dirección I2C. Por defecto, la dirección es 0x60 (como en el apartado anterior).

```
mh = Adafruit_MotorHAT(addr = 0x60)
```

A pesar de que este código de ejemplo no utiliza motores de corriente continua, todavía es importante tener en cuenta que el controlador PWM es *free running*. Por tanto, es responsabilidad del código "apagar" el motor.

```
def turnOffMotors():
        mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)
atexit.register(turnOffMotors)
```

Para crear el objeto real de motor paso a paso, puede solicitarlo al objeto MotorHAT creado anteriormente con **getStepper** (pasos, númpuerto), donde los pasos es la cantidad de pasos por vuelta del el motor (por lo general, un número entre 35 y 200) y donde númpuerto tiene un valor entre 1 y 2, ya que el puerto 1 tiene como terminales M1-M2, y el puerto 2 los pines M3-M4.

```
myStepper = mh.getStepper(200, 1)
```

A continuación, si deseamos utilizar la función **blocking step()** para recorrer varios pasos a la vez, podemos establecer la velocidad en RPM. Si usamos **OneStep()**, no es necesario configurar la velocidad.

```
myStepper.setSpeed(30)
```

Existen cuatro tipos esenciales de pasos que podemos usar con la HATMotor. Los cuatro tipos trabajarán con cualquier motor paso a paso unipolar o bipolar:

- ✓ **Pasos individuales**: Este es el tipo más simple de paso a paso. Consume menos energía. Utiliza una sola bobina para mantener el motor en marcha.
- ✓ **Pasos dobles**: Es también bastante simple, excepto que en lugar de una sola bobina, trabajan dos bobinas a la vez. Esto consume más energía (aproximadamente 2x), pero presenta mayor torque en el giro (fuerza) que un solo paso a paso.
- ✓ **Pasos intercalados**: Es una mezcla del simple y doble paso a paso. Tiene un poco más de fuerza que un solo escalonamiento, pero con una transición más suave entre los pasos.
- ✓ **Micropasos**: Utilizamos una mezcla de un solo paso a paso con PWM para una transición lenta entre los pasos. Es más lento que el tipo de un único paso a paso, pero tiene mucha más precisión.

```
while (True):
        print("pasos simples")
        myStepper.step(100, Adafruit_MotorHAT.FORWARD,
        Adafruit_MotorHAT.SINGLE)
        myStepper.step(100,Adafruit_MotorHAT.BACKWARD,
        Adafruit_MotorHAT.SINGLE)
        print("Pasos dobles")
        myStepper.step(100,Adafruit_MotorHAT.FORWARD,
        Adafruit_MotorHAT.DOUBLE)
        myStepper.step(100,Adafruit_MotorHAT.BACKWARD,
        adafruit_MotorHAT.DOUBLE)
        print("Pasos intercalados")
        myStepper.step(100,Adafruit_MotorHAT.FORWARD,Adafru
        it_MotorHAT.INTERLEAVE)
        myStepper.step(100,Adafruit_MotorHAT.BACKWARD,Adafr
        uit_MotorHAT.INTERLEAVE)
        print("Micropasos”)
        myStepper.step(100,Adafruit_MotorHAT.FORWARD,
        Adafruit_MotorHAT.MICROSTEP)
        myStepper.step(100,Adafruit_MotorHAT.BACKWARD,
        Adafruit_MotorHAT.MICROSTEP)
```

Como se observa en las líneas del script, se llaman a las distintas funciones de la librería HAT para producir los cuatro distintos modos de trabajo que se expusieron antes. En la figura 5.18 se muestra el resultado por la consola:

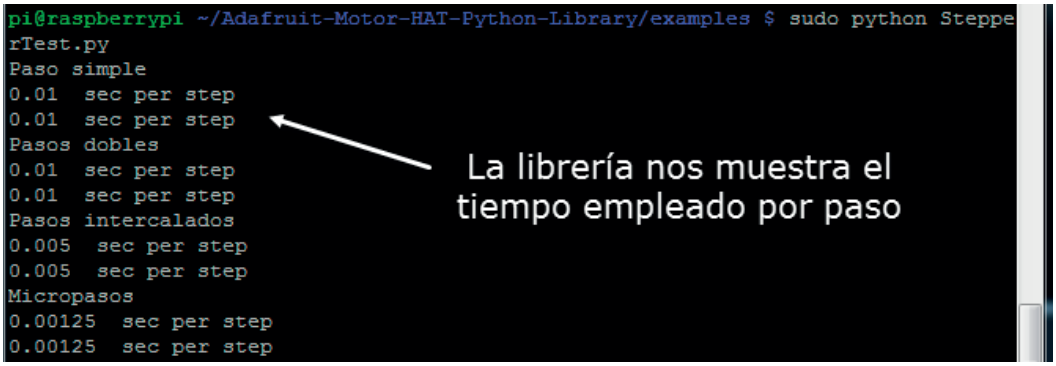


Figura 5.18

El listado completo del script **Stepper.Test.py** se muestra a continuación en la figura 5.19:

StepperTest.py - /home/pi/Adafruit-Motor-HAT-Python-Library/examples/StepperTest.py

File Edit Format Run Options Windows Help

```
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_DCMotor, Adafruit_StepperMotor

import time
import atexit
mh = Adafruit_MotorHAT()

def turnOffMotors():
        mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
        mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)

atexit.register(turnOffMotors)
myStepper = mh.getStepper(200, 1)
myStepper.setSpeed(30)
while (True):
        print("Paso simple")
        myStepper.step(100, Adafruit_MotorHAT.FORWARD,  Adafruit_MotorHAT.SINGLE)
        myStepper.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.SINGLE)

        print("Pasos dobles")
        myStepper.step(100, Adafruit_MotorHAT.FORWARD,  Adafruit_MotorHAT.DOUBLE)
        myStepper.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.DOUBLE)

        print("Pasos intercalados")
        myStepper.step(100, Adafruit_MotorHAT.FORWARD,  Adafruit_MotorHAT.INTERLEAVE)
        myStepper.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.INTERLEAVE)

        print("Micropasos")
        myStepper.step(100, Adafruit_MotorHAT.FORWARD,  Adafruit_MotorHAT.MICROSTEP)
        myStepper.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.MICROSTEP)
```

Figura 5.19

Por otra parte, podemos recorrer varios pasos a la vez con la función **step()**.

```
step(númeropasos, dirección, tipo)
```

Donde **númeropasos** es el número de pasos a completar, la dirección en un sentido o en el otro y el tipo de paso puede ser simple, doble, intercalado o micropaso.

Si tenemos dos motores podemos utilizar las dos líneas siguientes:

```
stepper1.step(100, Adafruit_MotorHAT.FORWARD,
Adafruit_MotorHAT.SINGLE)
stepper2.step(100, Adafruit_MotorHAT.BACKWARD,
Adafruit_MotorHAT.SINGLE)
```

Entonces, el primer motor se moverá 100 pasos, para, y luego el segundo motor comenzará a moverse.

CAPÍTULO

6 INTERNET DE LAS COSAS

En este capítulo, aprenderemos con un par de sencillos proyectos, como utilizar nuestra Raspberry Pi2 con el llamado **Internet de las Cosas**. A partir de ahora nos referiremos a ello con la abreviatura **IoT** (*Internet of Things*).

Básicamente se trata de que todas las cosas del mundo estén conectadas a Internet. Hoy en día tenemos smartphones, tablets, ordenadores portátiles, dispositivos multimedia en el salón, e incluso las propias televisiones que se conectan a Internet. A esto habría que añadir las videoconsolas, e incluso los coches. Sin embargo, eso no es nada en realidad si pensamos en la gran cantidad de cosas que hay en el mundo. No solo los dispositivos electrónicos pueden conectarse a Internet.

El Internet de las Cosas va mucho más allá. Algunos ejemplos de cosas conectadas a la red que podrían considerarse como parte de ese Internet de las Cosas serían los electrodomésticos que están conectados. Ya existen frigoríficos, hornos y lavadoras que pueden ser controladas desde un smartphone gracias a la conexión a Internet con la que cuentan. Ese es solo el primer paso de lo que está por llegar. Tanto a nivel doméstico como a nivel profesional, el Internet de las Cosas podría cambiar el mundo tal y como lo conocemos hoy. Pensemos solo en algunas de las aplicaciones que podrían llegar a tener lugar. Un agricultor debe conocer en todo momento las condiciones del campo que cultiva. Su trabajo consistiría en comprobar regularmente la temperatura y humedad del campo y registrar esos datos en un ordenador. Pero supongamos que todos esos datos fueran monitorizados de manera automática y registrados en un servicio *online*, de manera que el agricultor tuviera pleno conocimiento de las condiciones en que se encuentra el campo de cultivo, e incluso pudiera conocer cómo está en tiempo real. Y todavía hay más, con sensores lo suficientemente baratos, podría llegar a monitorizar absolutamente todas las plantas que está cultivando, conociendo cómo crecen y si algunas de ellas están teniendo problemas.

Las aplicaciones domésticas podrían ser igual de importantes. Por ejemplo, podríamos disponer de sensores y controladores en diversos elementos de la casa. Seguro que, en alguna ocasión, al irnos de viaje nos ha entrado la duda de si hemos cerrado el paso del gas de la cocina, o de si las persianas, ventanas o luces han quedado en la posición o estado que queríamos. Sería tan sencillo poder acceder al servicio con el que controlamos nuestra casa, comprobar que todos los elementos funcionan correctamente, e incluso modificar el estado de los mismos. O incluso, por ejemplo, si tenenos que volver en pocas horas a casa, podemos programar cuándo queremos que comience a prepararse la comida. Cosas como regular la temperatura del hogar cuando estemos nosotros allí, o encender las luces de manera automática, podrían ser hechos cotidianos de la vida. Es el Internet de las Cosas, las cosas que nos rodean, que pasarían a estar permanentemente conectadas.

6.1 ALMACENANDO EN LA NUBE DATOS DE HUMEDAD Y TEMPERATURA

En este primer proyecto del capítulo, vamos a aprender cómo utilizar la Raspberry Pi2 para monitorear algunos datos de forma remota, desde cualquier lugar del mundo. Vamos a empezar conectando un sensor de temperatura sencilla de humedad (DHT22) a nuestra Raspberry Pi. A continuación, vamos a utilizar un servicio llamado **Dweet.io** para almacenar datos en la nube. Por último, vamos a integrar la medición de datos con otro servicio llamado **Freeboard.io**. Este nos permitirá monitorizar gráficamente los datos que vienen de nuestra Raspberry Pi2.

6.1.1 El sensor de humedad/temperatura DHT22

El sensor de humedad/temperatura DHT22 (figura 6.1) se presenta en un cuerpo de plástico ideal para montar sobre un panel o similar. Utiliza un sensor capacitivo que devuelve una señal digital con la medición (no se necesitan pines analógicos). Es muy sencillo de utilizar aunque solo permite una lectura cada 2 segundos.

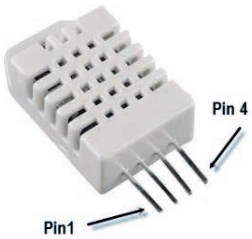


Figura 6.1

Simplemente es necesario alimentarlo de 3 a 5 V; el cable amarillo a un pin de entrada y el cable negro a GND. Usamos un protocolo de un solo cable pero no es directamente compatible con la librería de Dallas: One-Wire. Si necesitamos utilizar varios sensores al mismo tiempo, cada uno necesita su propio pin de datos. Es bastante sensible y preciso y funciona en un amplio rango de humedades.

Características:

- ✓ Alimentación: 3 a 5 V.
- ✓ Consumo: 2,5 mA máx. (en captura).
- ✓ Rango: 0-100 % (±2-5 %).
- ✓ Capaz de leer temperatura de −40 a 80 °C (±0.5 °C).
- ✓ Tiempo desensado: 1 vez cada 2 segundos.

Del montaje realizado que se muestra en la figura 6.2, destacamos que no utilizamos el pin 3, y que la resistencia de 10 KΩ tiene que conectarse entre la alimentación y el pin de salida de datos. También es interesante ver que, aunque el sensor entrega señales de naturaleza analógica, como son la humedad y la temperatura, estas señales las leemos por un pin digital de la Pi2.

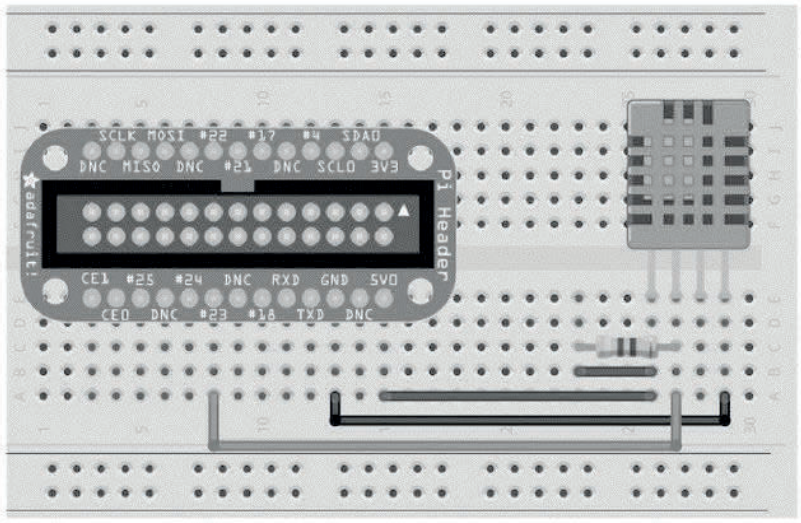


Figura 6.2

Este hecho revela que con un solo pin somos capaces de obtener dos lecturas. Por ello nos damos cuenta de la ventaja que aportan las señales digitales frente a las analógicas: menos cables y más simplicidad en los montajes.

El pin de la señal del sensor de DHT (pin número 2 del sensor) tiene que estar conectado en el pin GPIO 23 de la Raspberry. También debemos conectar el pin VCC (pin número 1 del sensor) al pin de 5 V de las Rasp y GND a GND. Por último, insertamos una resistencia de 1 KΩ entre el pin número 1 y 2.

En la figura 6.3 se muestra el montaje y conexionado real utilizando el conector Cobber de la Pi2 y el breadboard.

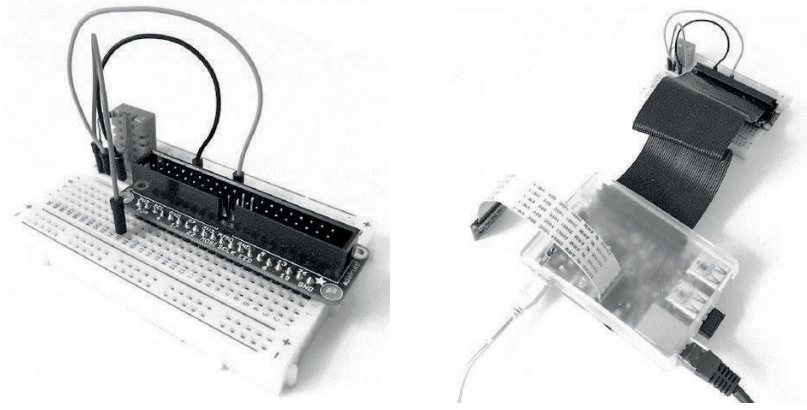


Figura 6.3

6.1.2 Probando el sensor DHT22

Lo primero que debemos hacer es bajarnos la librería **Adafruit_Python_DHT** e instalarla en nuestra Pi2 (tenemos la versión actualizada para este modelo de Rasp).

```
git clone https://github.com/adafruit/ Adafruit_Python_DHT.git
cd Adafruit_Python_DHT
```

Para instalar la librería de Python es necesario solventar las dependencias:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-openssl
```

Instalamos:

```
sudo python setup.py install
```

Para probar la librería de Python ejecutamos el script **simpletest.py** en la carpeta "examples". Este ejemplo es sencillo y debemos modificar antes un par de líneas para configurarlo según nuestras especificaciones de conexionado tal y como se observa en la figura 6.4.

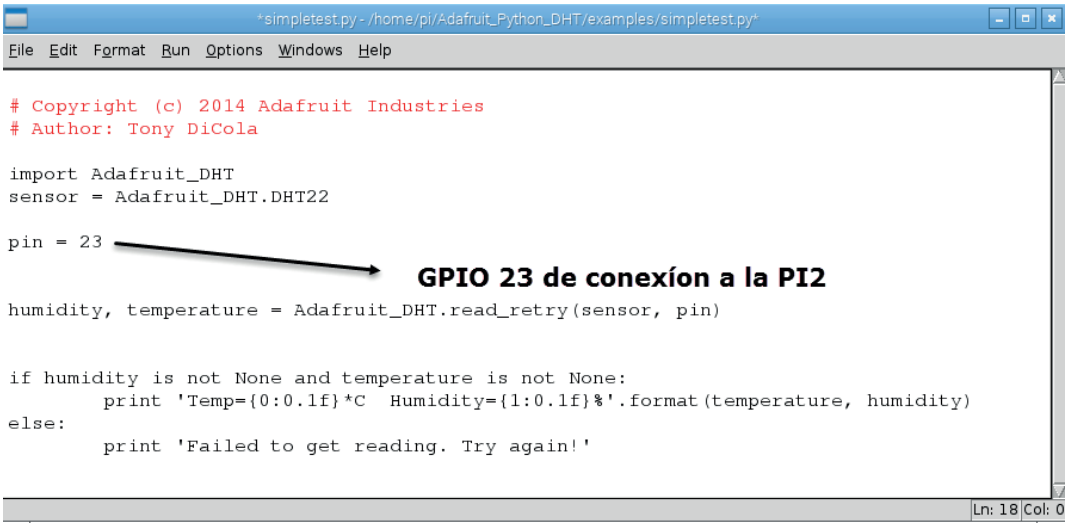


Figura 6.4

Ejecutamos el script y observamos el resultado en la consola (figura 6.5) mostrando la temperatura y humedad.

```
sudo python simpletest.py
```

```
pi@raspberrypi ~/Adafruit_Python_DHT/examples $ sudo python simpletest.py
Temp=24.4*C  Humidity=55.4%
pi@raspberrypi ~/Adafruit_Python_DHT/examples $ sudo python simpletest.py
Temp=24.3*C  Humidity=55.4%
```

Figura 6.5

6.1.3 Enviando datos a la nube

Ahora vamos a configurar el proyecto para enviar estos datos a la nube. Por eso vamos a utilizar un servicio en la nube llamado **dweet.io**. Esta es una manera muy simple para almacenar datos en la nube, ya que no requiere ningún tipo de registro. Podemos encontrar más información acerca de dweet.io (figura 6.6) en: http://dweet.io/

Las librerías disponibles para comunicarnos como clientes a Dweet.io se pueden observar en la figura 6.7.

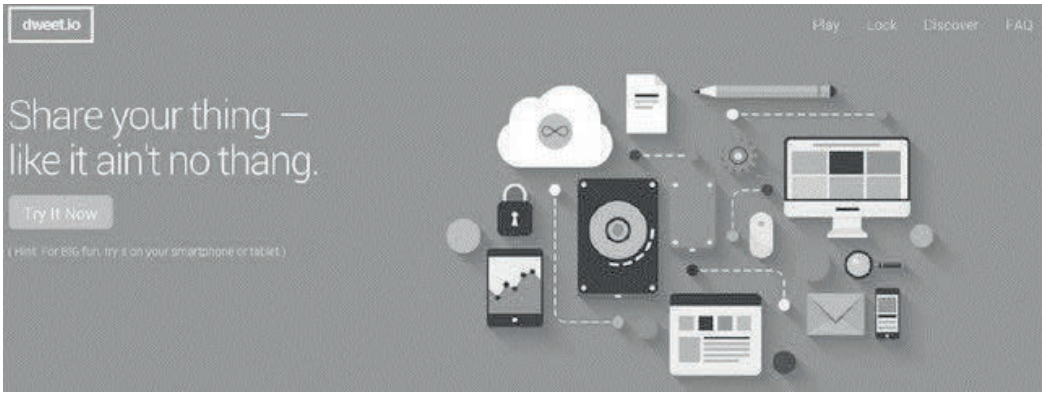


Figura 6.6

Node.js	https://github.com/buglabs/node-dweetio (Official)
Javascript	https://github.com/buglabs/dweetio-client (Official)
Python	https://github.com/paddycarey/dweepy (Unofficial) https://github.com/bliti/pydweet (Unofficial)
Ruby	https://github.com/vannell/ruby-dweetio (Unofficial)

Figura 6.7

Evidentemente optamos por la librería en Python. Instalamos **dweepy**, que es un sencillo cliente de Python para dweet.io. Lo descargamos e instalamos:

```
git clone git://github.com/paddycarey/dweepy.git
cd dweepy
python setup.py install
```

Dweepy tiene como objetivo proporcionar una interfaz sencilla de Python a Dweet.io. Ha sido diseñado para ser fácil de usar, y tiene como objetivo cubrir la API Dweet.io completamente. Para usarlo, primero tendremos que importar dweepy:

```
import dweepy
```

Podemos enviar un dweet, sin especificar un nombre para el parámetro que queremos enviar.

```
dweepy.dweet({'alguna clave': 'algún valor'})
   {
     u'content': {u': alguna clave': u'algún valor'},
     u'created': u'2014-03-19T10:35:59.504Z',
     u'thing': u'unequaled-start'
   }
```

Si no especificamos un nombre del parámetro, dweet.io le asignará un nombre. Es más práctico enviar un dweet de un parámetro con un nombre especificado como se muestra a continuación:

```
dweepy.dweet_for('sensor', {'alguna clave': 'algún valor'})
   {
    u'content': {u'alguna clave': u'algún valor'},
    u'created': u'2014-03-19T10:38:46.010Z',
    u'thing': u'dht22'
   }
```

Para leer los dweet subidos debemos utilizar el siguiente procedimiento:

```
dweepy.get_latest_dweet_for('sensor')
[
    {
     u'content': {u'alguna clave': u'algún valor'},
     u'created': u'2014-03-19T10:38:46.010Z',
     u'thing': u'sensor'
    }
]
```

Es necesario tener en cuenta que dweet.io solamente guarda los últimos 500 dweets durante un período de 24 horas. Si no se envían dweets en las últimas 24 horas, se eliminará el histórico de los datos subidos. Ahora añadimos al script básico de funcionamiento del sensor DHT22 la parte del código que nos permite "dweetutear" los datos de temperatura y humedad (figura 6.8).

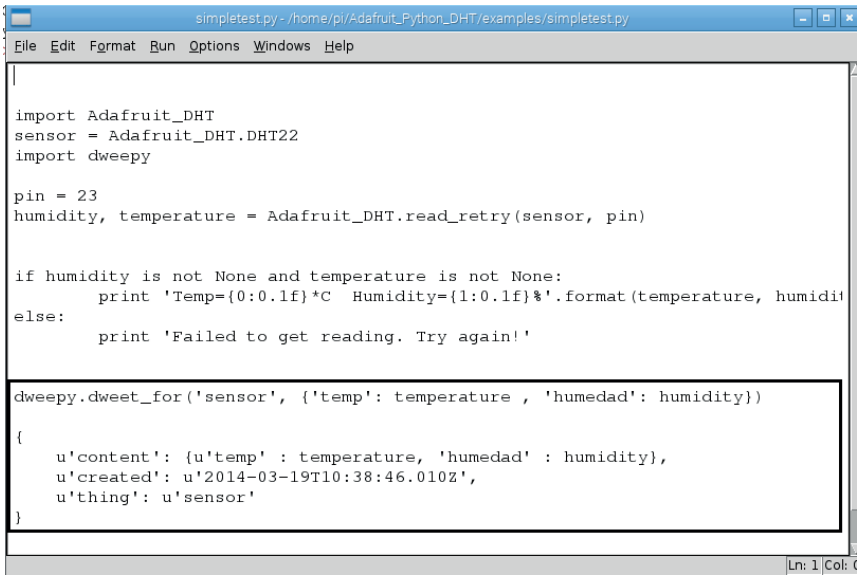
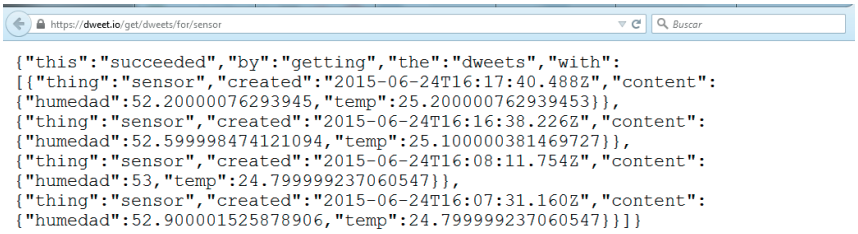


Figura 6.8

Como vemos, es muy sencillo el envío de datos a la nube utilizando este método. De hecho, casi es como tuitear. Para ver que los datos están realmente subidos, accedemos a la URL https://dweet.io/get/dweets/for/sensor donde "sensor" es el nombre que le hemos dado a nuestra conexión o "thing" cuando nos conectamos a dweet.io.

En la figura 6.9 se muestra el resultado de ejecutar nuestro proyecto llamado **simpletest.py**.

```
sudo python simpletest.py
```



Se muestran tres dweets,ya hemos ejecutado el script simpletest.py tres veces.

Figura 6.9

Para realizar continuamente subidas procedentes del sensor, solo tendríamos que añadir una estructura de bucle tipo **while true**.

De todas maneras observamos que los datos que nos interesan (temperatura y humedad) están incrustados en otra información irrelevante. Además no presentan, que digamos, una visualización entendible y clara para el usuario. Por ello, vamos a integrar otra aplicación (figura 6.10) que nos va a hacer la vida más fácil. Es decir, nos gustaría ver los datos gráficamente. Para ello, vamos a utilizar **freeboard.io** (https://www.freeboard.io/). El primer paso es crear una cuenta gratuita.

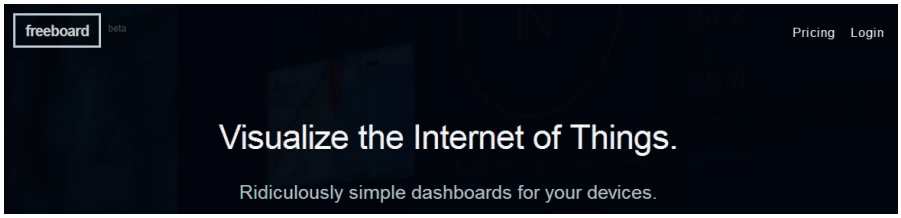


Figura 6.10

A continuación, creamos un nuevo tablero de instrumentos y una nueva fuente de datos. Aquí es donde debemos introducir el nombre de "Dweet.io" (figura 6.11).

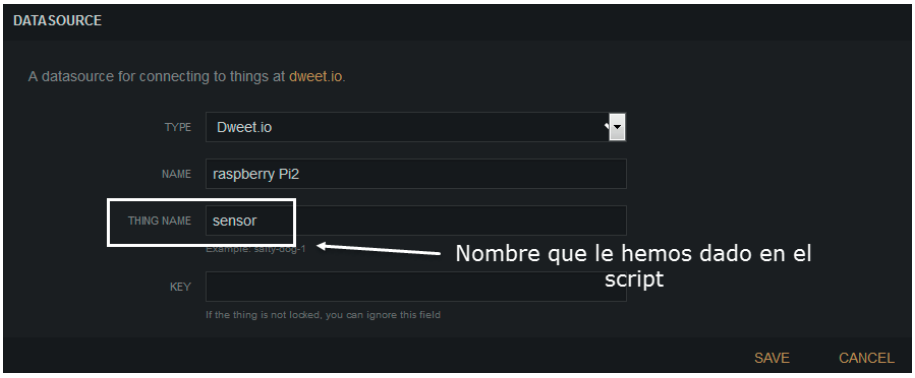


Figura 6.11

Después de eso, debemos ver que los datos provienen del tablero (figura 6.12):

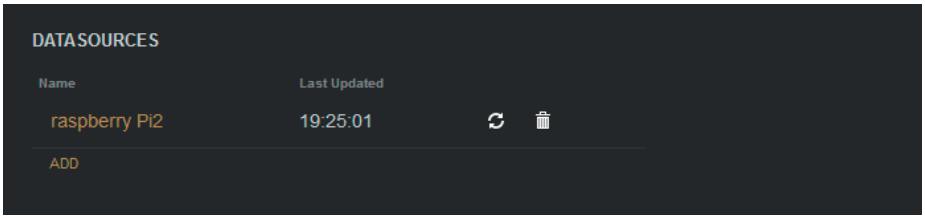


Figura 6.12

A continuación, creamos un nuevo panel y luego un nuevo widget. Vamos a crear un widget tipo Gauge para visualizar la temperatura actual procedente de la Pi2, como se muestra en la figura 6.13.

Figura 6.13

Estos ajustes permiten vincular el widget a la fuente de datos que hemos creado antes. Entonces, podemos hacer lo mismo para la humedad (figura 6.14).

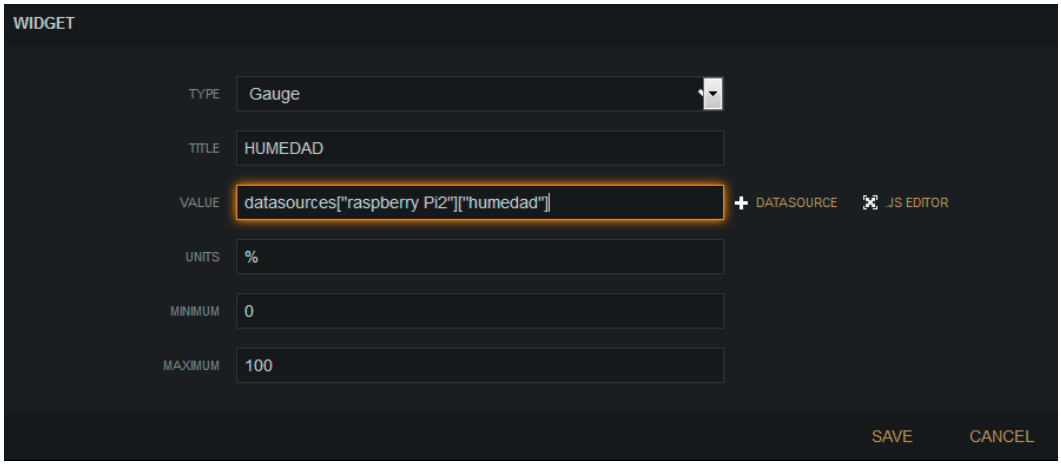


Figura 6.14

En la figura 6.15 observamos el resultado final. Si los valores de temperatura y humedad presentan demasiados decimales, siempre podemos ajustarlos con la función **round()** en el script.

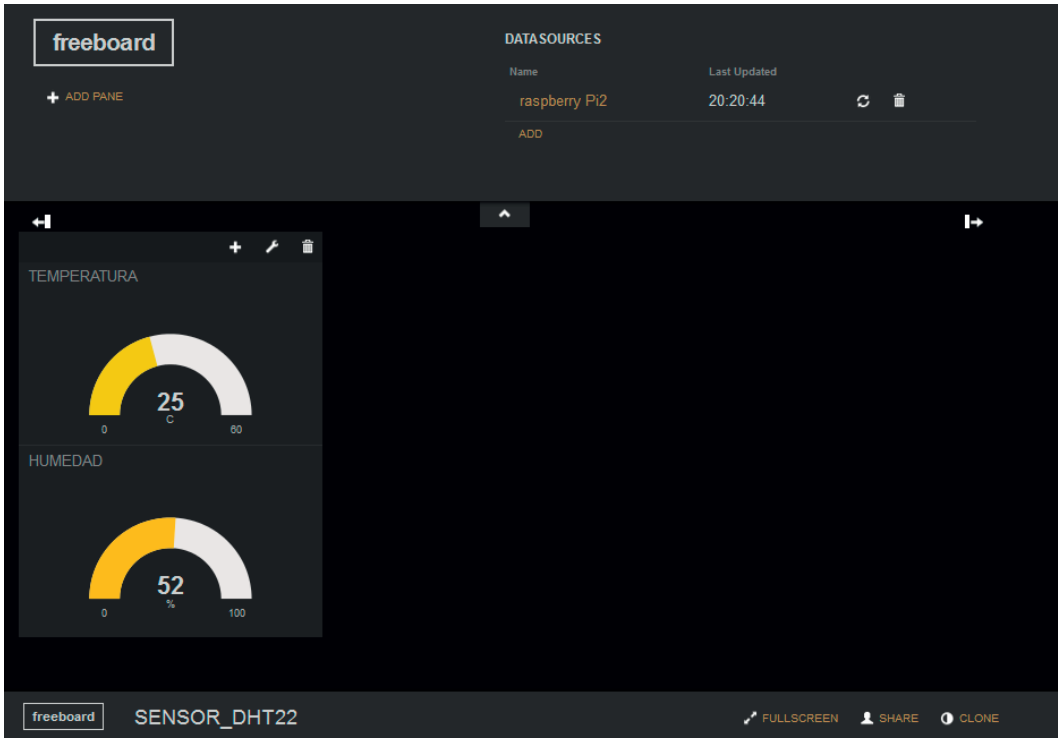


Figura 6.15

En la figura 6.15 tenemos una opción llamada **share** que nos mostrará un enlace para que podamos compartirlo públicamente, y, de esta manera, cualquiera podrá acceder a la monitorización de los datos de nuestro sensor DHT22.

6.2 SISTEMA DE ALARMA CON CARRIOTS

En este proyecto vamos a construir un sistema de alerta para su hogar u oficina. Para ello, utilizamos una Raspberry Pi2 capaz de detectar presencia con un sensor **PIR** y **Carriots** como motor para enviar alertas por SMS.

6.2.1 Probando el sensor PIR

Un sensor PIR (sensor pasivo de infrarrojos) se basa en la idea que todos los objetos emiten energía en forma de radiación a causa de tener un calor corporal por encima del cero absoluto. Los sensores PIR (figura 6.16) están compuestos por dos ranuras, cada una de ellas sensible a los infrarrojos. Cuando un cuerpo caliente pasa por delante del campo de detección del sensor, una de las dos mitades detecta la diferencia de calor y provoca un diferencial entre las dos mitades de las ranuras. Ocurre lo mismo cuando el cuerpo sale de la zona de detección; la otra mitad detecta un cambio y provoca otra diferencia de potencial igual pero de sentido contrario.

De esta manera el sensor es capaz de distinguir si ha habido movimiento en la habitación. Son sensores de infrarrojo pasivos porque, por un lado, capturan los infrarrojos y por el otro, como no irradian ninguna energía sobre los objetos, son pasivos. La clave son las lentes de Fresnel que juegan un papel decisivo en los sensores PIR ya que consiguen ampliar su campo de detección. Una lente de Fresnel es una lente plano-convexa que se utiliza para conseguir focalizar una mayor cantidad de radiación sobre el sensor.

La prueba que vamos a implementar consiste en detectar movimiento y encender un diodo LED. Así de simple, ya que solo nos interesa conocer su funcionamiento básico. El tipo de PIR que utilizaremos es el que podemos adquirir en www.bricogeek.com. El fabricante nos proporciona el conexionado que es el siguiente:

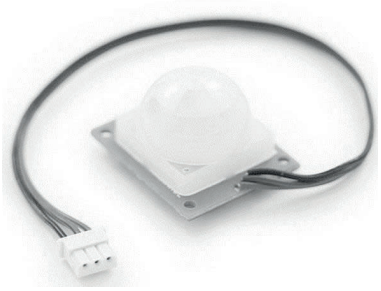


Figura 6.16

- ✓ Cable rojo: VCC (5 a 12 V).
- ✓ Cable negro: Pin "Alarma" en colector abierto.
- ✓ Cable marrón: GND.

El conexionado se muestra en la figura 6.17:

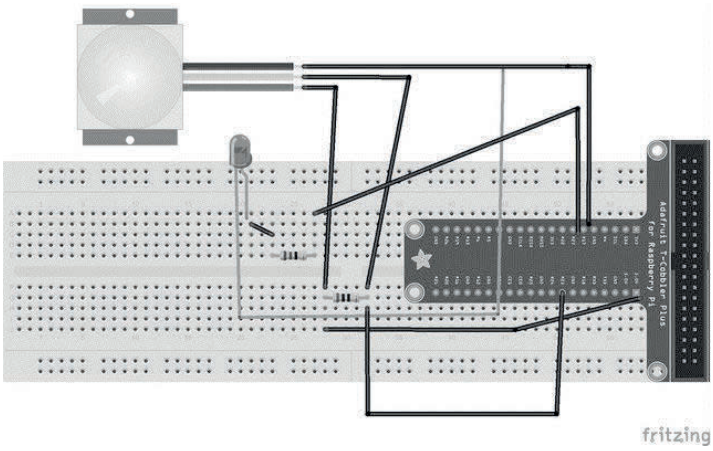


Figura 6.17

Utilizamos el pin GPIO 23 como salida de señal de presencia y una resistencia de 10 KΩ como pull-up. Cuando se detecte movimiento haremos parpadear un LED a través del GPIO 17. El script se muestra en la figura 6.18. Es preciso destacar que he ajustado por el método de prueba y error, los tiempos utilizando la función time.sleep(x) para calibrar inicialmente el sensor y para sincronizarlo con la Pi2.

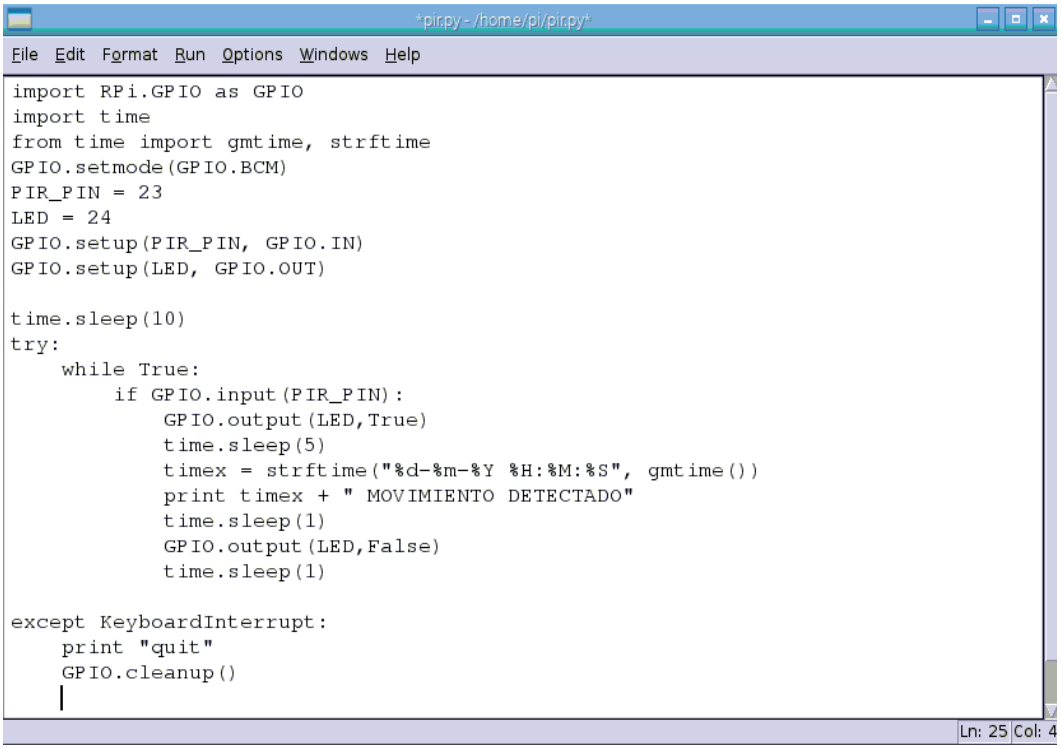


Figura 6.18

También he añadido una línea de código para mostrar la fecha y la hora y así tener constancia de cuándo se ha producido el movimiento.

El resultado por terminal se muestra en la figura 6.19.

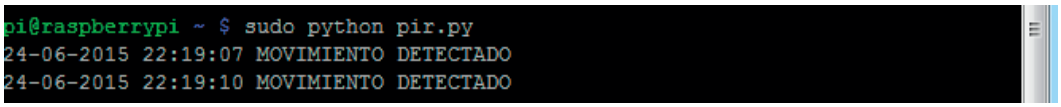


Figura 6.19

6.2.2 Utilizando Carriots para construir un sistema de alarma

Carriots (www.carriots.com) (figura 6.20) es una Plataforma como Servicio (**PaaS** en sus siglas en inglés) diseñada para proyectos del Internet de las Cosas (**IoT**) y de Máquina a Máquina (**M2M**).

Figura 6.20

Qué se puede hacer con Carriots:

- ✓ Recopilar y almacenar todo tipo de datos de sus dispositivos.
- ✓ Construir potentes aplicaciones con nuestro motor SDK.
- ✓ Desplegar y escalar desde modestos prototipos hasta miles de dispositivos.

Los proyectos IoT se realizan de una forma más rápida, barata, sencilla, fiable y escalable en la nube. Se siguen los siguientes pasos en todos los casos:

- ✓ Conectar cualquier dispositivo a Carriots: solo se necesita un acceso a Internet.
- ✓ Enviar sus datos a la REST API de Carriots: es sencillo, seguro y estándar.
- ✓ Añadir inteligencia con el potente código SDK Java: simplificado gracias a Groovy.
- ✓ Es posible acceder a todo ello, integrarlo, gestionarlo y hacer que sea realmente potente combinándolo todo.

Lo primero que tenemos que hacer es crear de forma gratuita una cuenta en Carriots.

La Raspberry Pi2 debe estar programada para enviar un flujo de datos a Carriots indicando si se ha activado el sensor PIR. Todos los flujos de datos enviados por la Raspberry Pi2 se recogen y almacenan en Carriots. Carriots es una enorme base de datos que recoge toda la información que nuestros sensores envían. Además de almacenamiento de datos, el verdadero poder de Carriots es construir aplicaciones muy rápidamente con pocas líneas de código Groovy. En este escenario vamos a construir una simple "Alerta" que envíe un SMS en caso de que la Pi2 detecte movimiento.

Ahora que estamos registrados en Carriots, accedemos a nuestro panel de control y observamos que tenemos ya creado un dispositivo nuevo que tiene el formato **defaultDevice@ miusuario** (figura 6.21) con un id_developer que necesitaremos más tarde.

Figura 6.21

Vamos ahora a la pestaña **MI CUENTA** y copiamos la **Api_key** que es la llave que nos dará acceso completo a la base de datos de Carriots para subir datos (figura 6.22).

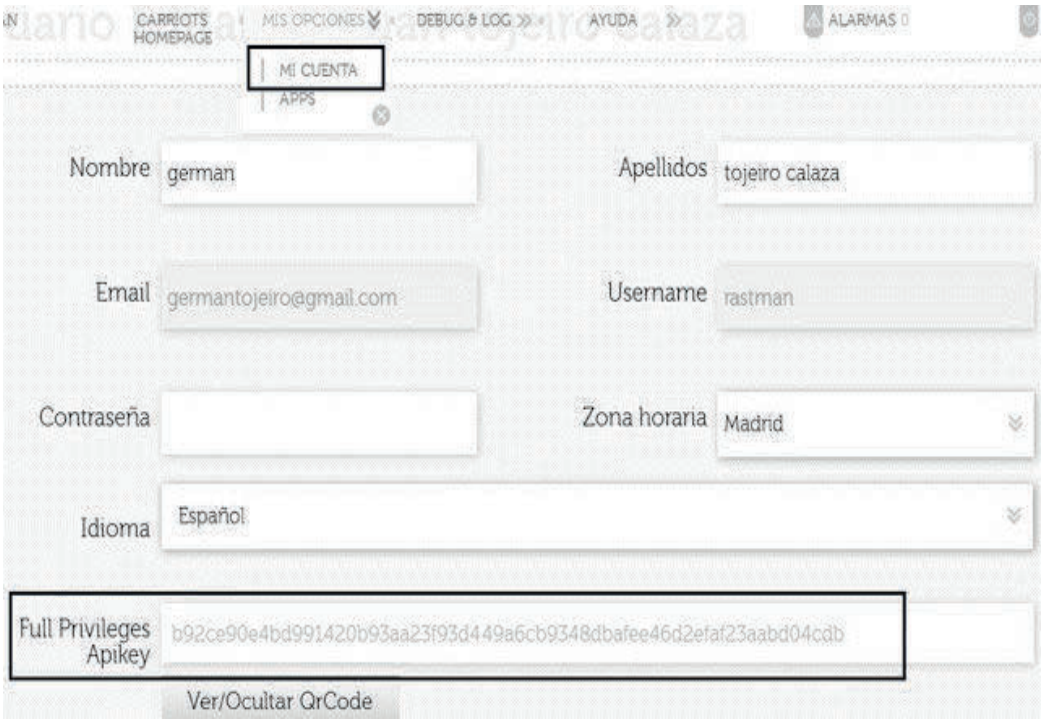


Figura 6.22

Con los datos: **id_developer** y la Api_key ya podemos empezar a diseñar nuestro script. De hecho, es lo necesario para conectar apropiadamente con Carriots. El script completo se muestra en la figura 6.23. Básicamente es el mismo que el del apartado anterior salvo que hemos añadido las reglas del protocolo para poder comunicarnos con este servidor de datos.

Se pueden observar las dos líneas de código donde se añaden los datos personales a los que nos hemos referido en el párrafo anterior. En realidad se trata de una plantilla ya diseñada, que hemos modificado para adecuarla al objetivo propio, que no es, ni más ni menos, que detectar un movimiento en el sensor PIR y, en consecuencia, subir un aviso a la base de datos que refleje esa detección.

```
import RPi.GPIO as GPIO
import time
from urllib2 import urlopen, Request
from time import mktime, sleep, gmtime, strftime
from datetime import datetime
from json import dumps

GPIO.setmode(GPIO.BCM)
PIR_PIN = 23
LED = 24
GPIO.setup(PIR_PIN, GPIO.IN)
GPIO.setup(LED, GPIO.OUT)

class Client (object):
    api_url = "http://api.carriots.com/streams"

    def __init__(self, api_key=None, client_type='json'):
        self.client_type = client_type
        self.api_key = api_key
        self.content_type = "application/vnd.carriots.api.v2+%s" % self.client_type
        self.headers = {'User-Agent': 'Raspberry-Carriots',
                        'Content-Type': self.content_type,
                        'Accept': self.content_type,
                        'Carriots.apikey': self.api_key}
        self.data = None
        self.response = None

    def send(self, data):
        self.data = dumps(data)
        request = Request(Client.api_url, self.data, self.headers)
        self.response = urlopen(request)
        return self.response

    def rc_time(PIR_PIN):
        measurement = 1

        while GPIO.input(PIR_PIN) == GPIO.HIGH:
          measurement = 0
        return measurement

    def main():

        on = 1
        off = 2

        device = "defaultDevice@rastman"
        apikey = "b92ce90e4bd991420b93aa23f93d449a6cb9348dbafee46d2efaf23aabd04cdb"

        lights = off
        client_carriots = Client(apikey)

        while True:
            timestamp = int(mktime(datetime.utcnow().timetuple()))
            if GPIO.input(PIR_PIN):
                GPIO.output(LED,True)
                time.sleep(1)
                timex = strftime("%d-%m-%Y %H:%M:%S", gmtime())
                print timex + " MOVIMIENTO NO DETECTADO"
                time.sleep(1)
                GPIO.output(LED,False)
                time.sleep(1)

            if rc_time(PIR_PIN) == GPIO.HIGH:
                timex = strftime("%d-%m-%Y %H:%M:%S", gmtime())
                print timex + " MOVIMIENTO  DETECTADO"
```

Figura 6.23

El resultado de ejecutar **pir1.py** se muestra en la figura 6.24 donde vemos que cuando se detecta movimiento se sube un aviso al servidor y este responde con un “OK”.

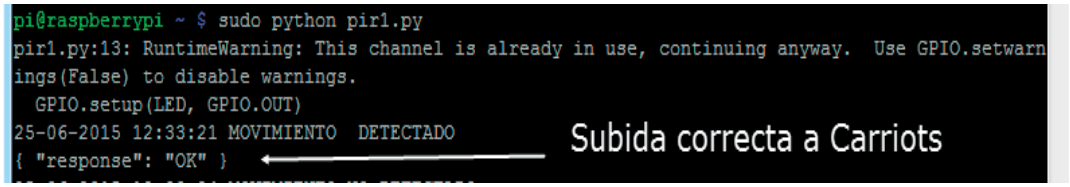


Figura 6.24

Accediendo a la ventana de trama de datos en nuestro panel de control observamos los movimientos detectados (figura 6.25).

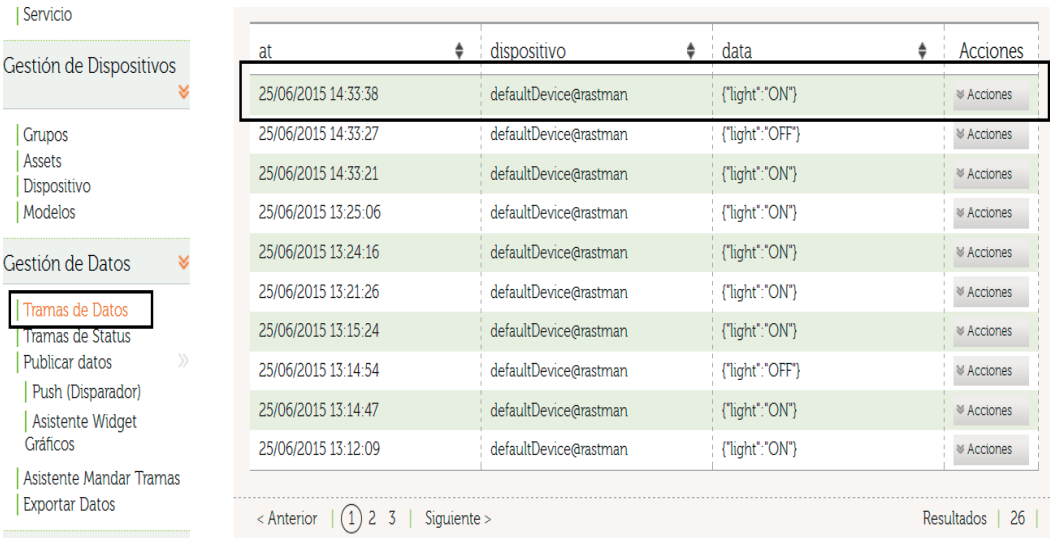


Figura 6.25

Como punto final vamos a crear y configurar un procedimiento para enviarnos automáticamente un SMS a nuestro móvil si un movimiento es detectado. Para realizar esto, accedemos al panel de control, y luego, en la gestión de dispositivos, localizamos nuestro dispositivo y seleccionamos su nombre. A continuación hacemos clic en el botón **NUEVO** en la pestaña de **Listener** (figura 6.26).

```
            new_lights = on


        else:
            new_lights = off


        if lights is not new_lights:
            lights = new_lights
            data = {"protocol": "v2", "device": device, "at": timestamp, "data": dict
            light=("ON" if new_lights is on else "OFF")}
            carriots_response = client_carriots.send(data)
            print carriots_response.read()
        time.sleep(1)

if __name__ == '__main__':
    main()
                            I
GPIO.cleanup()
```

Figura 6.26

Rellenamos los campos que se indican en la figura 6.27 con los datos requeridos. Cada vez que se detecte un movimiento se enviará el mensaje indicado al teléfono móvil.

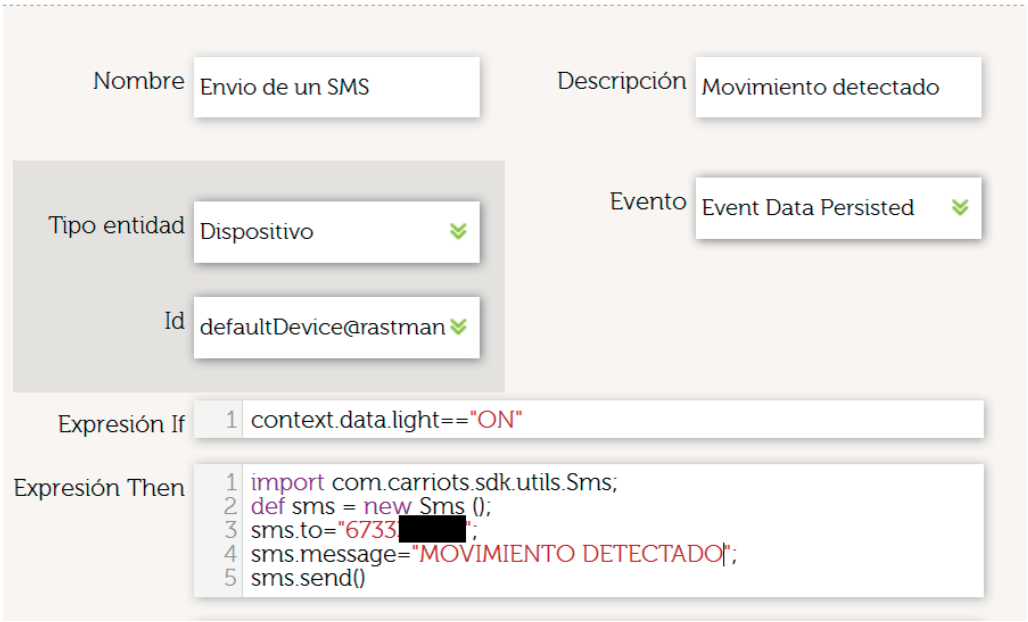


Figura 6.27

CAPÍTULO

7 RASPBERRY Pi2 Y ARDUINO

La Raspberry Pi2 es una plataforma a tener en cuenta para el desarrollo de aplicaciones domóticas y de control. Sin embargo, la Raspberry Pi2 tiene una pequeña limitación en cuanto al número de entradas/salidas disponibles, así como problemas relacionados con los tiempos de respuestas que se consiguen programando en Python.

Una posible solución a esta limitación puede ser combinar la Pi2 con Arduino y relegar todo el control del hardware al Arduino y utilizar la Raspberry como controlador maestro. En éste capítulo vamos a abordar diferentes técnicas para comunicar ambos dispositivos a través del cable USB.

Evidentemente, es un tema interesante para aquellos de los lectores que tengan cierta experiencia en la programación de Arduino y empiecen con la Raspberry. Proyectos que impliquen la comunicación wifi o la adaptación de niveles de voltajes no les van a suponer un coste adicional en shields.

7.1. PROGRAMANDO EL ARDUINO DESDE LA Pi2

Vamos a instalar el IDE de Arduino en la Raspberry y ejecutar un programa simple desde allí para que se ejecute en el propio Arduino. El **Arduino IDE** está disponible para la Raspberry Pi2. Es un poco lento, pero factible. Los comandos para instalarlo se muestran a continuación:

```
sudo apt-get update
sudo apt-get install arduino
```

Arduino IDE se conecta a la Raspberry Pi2 a través de su cable USB para programarlo. Es muy útil entrar en el entorno gráfico LXDE utilizando la conexión remota por VNC. Ahora veremos el IDE de Arduino perfectamente instalado (figura 7.1).

Ahora podemos conectar el Arduino a nuestra Raspberry Pi2. En el menú Herramientas, seleccionamos el tipo de tarjeta de Arduino Uno. Luego, desde la opción de puerto serie, seleccionamos la opción **/dev/ttyACM0**. Para cargar un programa de prueba que hará que el LED en el Arduino parpadee, seleccionamos en el menú archivo el programa: Blink. Hacemos clic en la derecha de la flecha en la barra de herramientas para iniciar el proceso de compilación y subida al Arduino. Si todo

está bien, debería ver un mensaje: “Carga terminada” en el área de estado en la parte inferior del IDE ventana (figura 7.2).

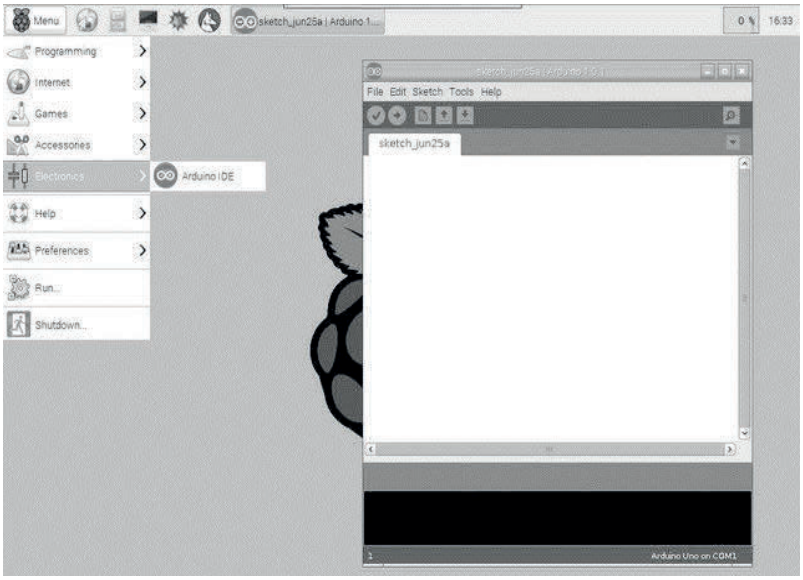


Figura 7.1

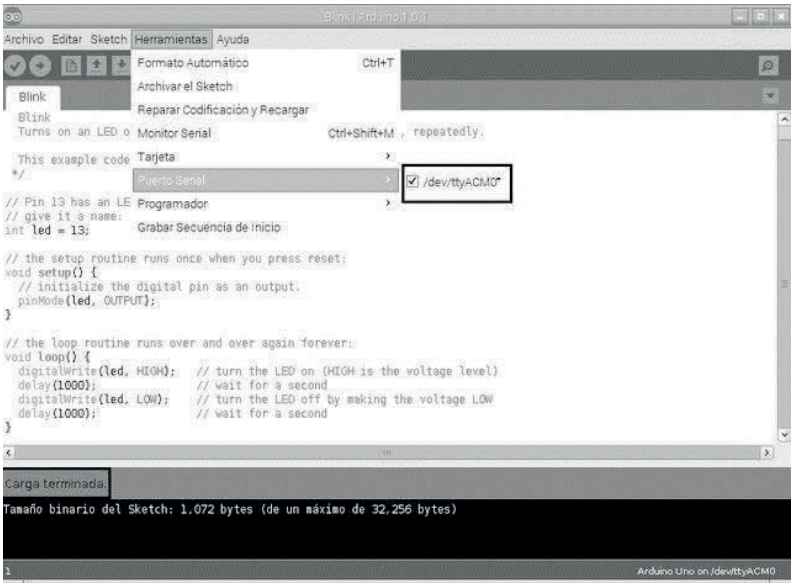


Figura 7.2

Si todo va bien, veremos parpadear el LED que tiene el Arduino en el pin 13.

7.2 UTILIZANDO EL MONITOR SERIE

El IDE de Arduino incluye una característica llamada **el monitor de serie**, que nos permite tanto enviar mensajes de texto al Arduino como verlos. Para probar esto, primero escribiremos un programa corto. Este sketch repetirá el envío del mensaje cada segundo (figura 7.3).

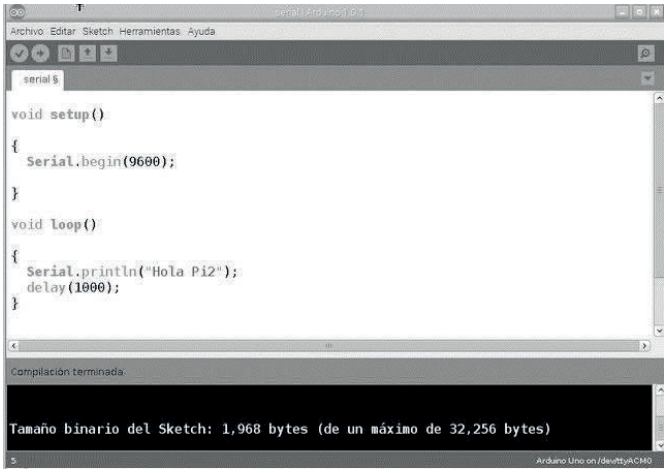


Figura 7.3

Tan pronto como el sketch se carga en el Arduino, se comenzará a enviar el mensaje "Hola Pi2" en el terminal serie. No veremos esto hasta que abramos el monitor serie (figura 7.4) haciendo clic en el icono que parece una lupa, en la derecha de la barra de herramientas.

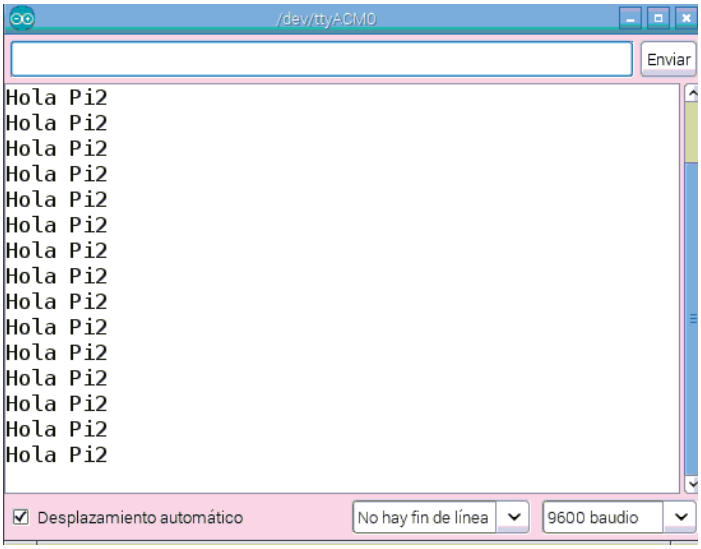


Figura 7.4

El monitor de serie tiene una lista desplegable en la esquina inferior derecha, donde podemos seleccionar la velocidad de transmisión (velocidad de la comunicación). Si esto no está ya establecido en 9600, es necesario cambiarlo a ese valor.

Más adelante en este capítulo, nos centraremos en la escritura de nuestro propio código personalizado para comunicarnos con los programas en Python que se ejecutan en la Pi2 de modo que no necesitaremos tener el IDE de Arduino ejecutándose. Un enfoque más genérico y práctico es utilizar algo llamado **PyFirmata**.

7.3 CONFIGURACIÓN DE PYFIRMATA

Primero descargamos e instalamos el sketch **Firmata** para Arduino y, por otro lado, hacemos lo propio con el script **PyFirmata** en nuestra Pi2. El Arduino IDE incluye Firmata, así que todo lo que tenemos que hacer para instalarlo en Arduino es compilar y cargar el sketch **StandardFirmata** que viene por defecto incluido en los ejemplos del IDE. Una vez que **Firmata** está instalado, el Arduino esperará la comunicación con la Raspberry.

Ahora, lo que necesitamos es instalar el script **PyFirmata** (la otra mitad del enlace) . Esto requiere el uso de la librería **PySerial**. La librería **PySerial** permite el uso del puerto serie (pines **Rx** y **Tx**) de la Raspberry usando Python. Instalamos la librería:

```
sudo apt-get install python-serial
```

Ahora bajamos e instalamos la librería **PyFirmata**:

```
sudo git clone https://github.com/tino/pyFirmata.git
cd pyFirmata
sudo python setup.py install
```

Un Arduino Uno por sí solo utiliza alrededor de 50 mA, por lo que es perfectamente que sea alimentado por la Pi2 a través de la conexión USB. Sin embargo, si tenemos gran cantidad de electrónica externa al Arduino, no nos quedará más remedio que alimentarlo desde nuestro propio adaptador de corriente DC.

El único inconveniente de la utilización de **Firmata** es que todas las instrucciones tienen que venir de la Raspberry Pi2.

7.4 CONTROL DE LA SALIDAS DIGITALES EN UN ARDUINO DESDE UNA Pi2

Abordaremos el uso de la librería **PyFirmata** para escribir un script en Python en la Pi2 que haga parpadear un LED conectado en el pin 10 del Arduino. Lo observamos en la figura 7.5 y lo hemos escrito utilizando el procesador de textos nano.

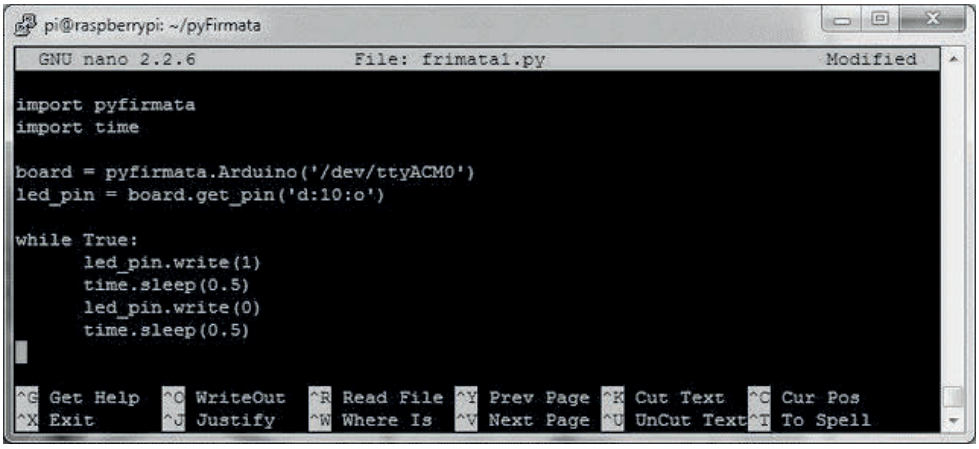


Figura 7.5

Hay que mencionar que después de importar las librerías pertinentes, configuramos el pin 10 del Arduino como salida utilizando la línea de código siguiente:

```
led_pin = board.get_pin('d:10:o')
```

Ahora vamos a leer una entrada digital procedente del Arduino. La siguiente secuencia de comandos de Python imprime un mensaje cada vez que se accionamos el pulsador. Abrimos un editor (nano o IDLE) y escribimos el siguiente código mostrado en la figura 7.6.

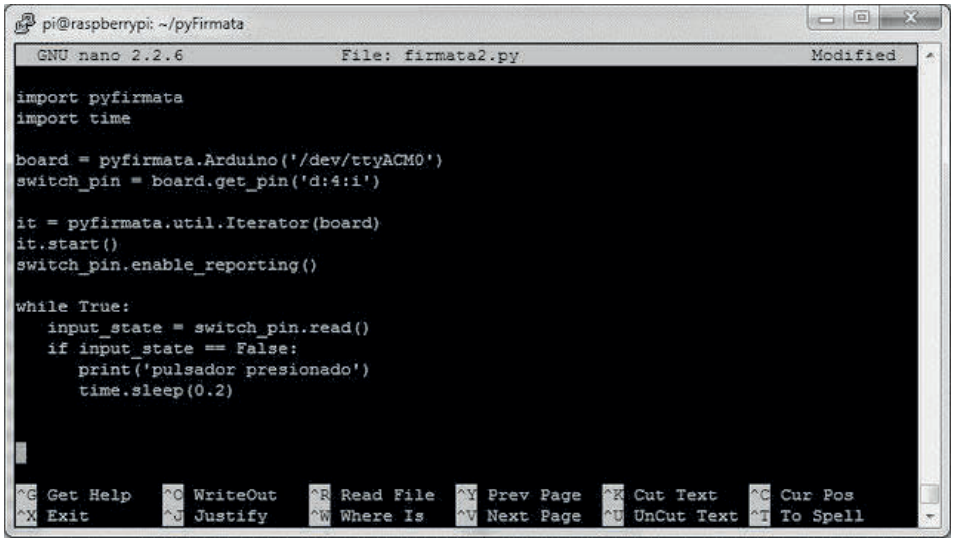


Figura 7.6

El pulsador se encuentra en el pin 4. Para evitar que el Arduino nos esté mandando continuamente datos en los pines de lectura, se utiliza un **Iterator**. El Iterator es un **thread**, por lo que hay que matarlo al salir. Lo más fácil es romper la conexión con el puerto serie, el thread se suicida el solito.

```
it = pyfirmata.util.Iterator(board)
it.start()
```

Esto es muy similar a la conexión de un interruptor directamente a una Pi2, y si solo tenemos un pulsador, no hay un beneficio real y práctico en el uso de un Arduino para hacer esto.

7.5 ENTRADAS ANALÓGICAS DE ARDUINO CON PYFIRMATA

En este apartado vamos a leer las entradas analógicas del Arduino utilizando PyFirmata. Abrimos un editor (nano o IDLE) y escribimos el siguiente código mostrado en la figura 7.7.

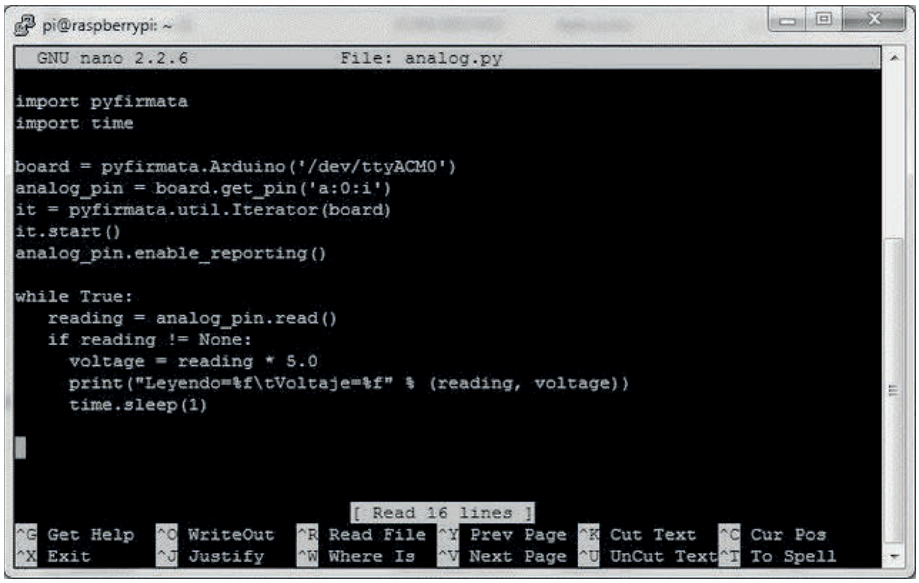


Figura 7.7

La secuencia de comandos en Python mostrará la tensión de la primera entrada analógica. Utilizamos un **trimmer** para variar dicha tensión y poder observar los cambios de lectura en la consola. Tecleamos el siguiente comando para ejecutar el script. El resultado lo vemos en la figura 7.8.

```
sudo python analog.py
```

```
pi@raspberrypi ~ $ sudo python analog.py
Leyendo=0.471200        Voltaje=2.356000
Leyendo=0.480900        Voltaje=2.404500
Leyendo=0.260000        Voltaje=1.300000
Leyendo=0.219000        Voltaje=1.095000
Leyendo=0.396900        Voltaje=1.984500
Leyendo=0.421300        Voltaje=2.106500
Leyendo=0.207200        Voltaje=1.036000
Leyendo=0.252200        Voltaje=1.261000
Leyendo=0.432100        Voltaje=2.160500
```

Figura 7.8

7.6 SALIDAS PWM CON PYFIRMATA

Controlaremos el brillo de un LED mediante PWM utilizando un Arduino. Usamos **PyFirmata** para enviar comandos a un Arduino para generar una señal PWM en su salida digital en el pin 10 donde está conectado el LED. Abrimos un editor (nano o IDLE) y escribimos el código mostrado en la figura 7.9. Hay que recordar que en nuestro Arduino es necesario tener cargado el sketch **standadFirmata**.

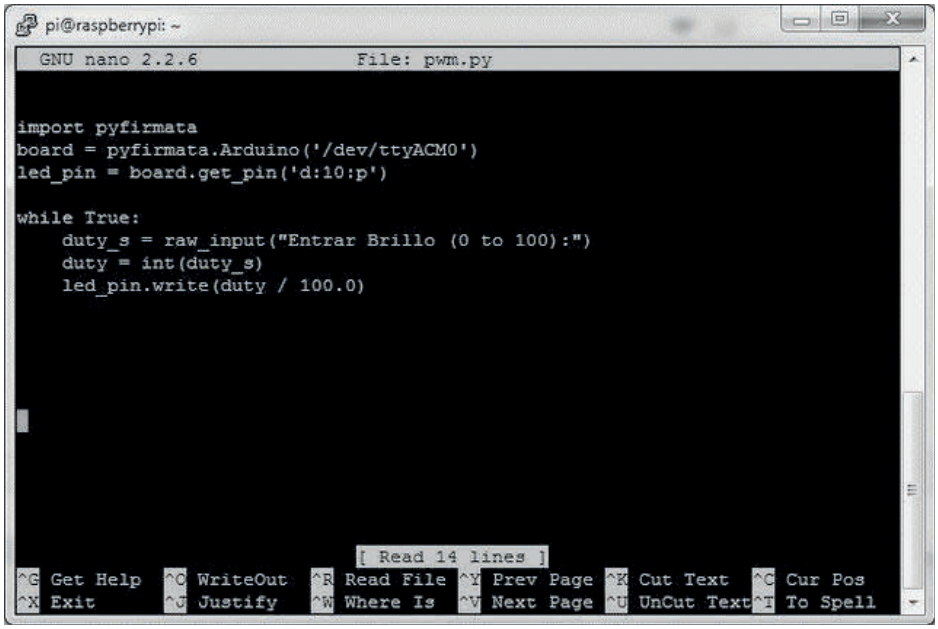


Figura 7.9

El esquema de conexionado del Arduino se observa en la figura 7.10.

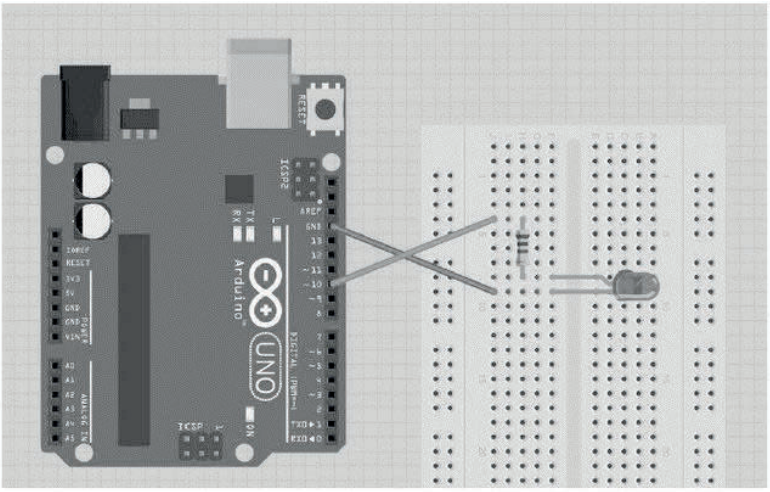


Figura 7.10

El resultado por consola al ejecutar `sudo python pwm.py` es el mostrado en la figura 7.11. La entrada del brillo es por teclado y veremos cómo cambia sobre el LED en tiempo real.

```
pi@raspberrypi ~ $ sudo python pwm.py
Entrar Brillo (0 to 100):50
Entrar Brillo (0 to 100):100
Entrar Brillo (0 to 100):45
Entrar Brillo (0 to 100):10
Entrar Brillo (0 to 100):1
Entrar Brillo (0 to 100):5
Entrar Brillo (0 to 100):89
Entrar Brillo (0 to 100):100
Entrar Brillo (0 to 100):25
Entrar Brillo (0 to 100):2
Entrar Brillo (0 to 100):
```

Figura 7.11

7.7 SHIELD DE EXPANSIÓN DE ARDUINO PARA LA Pi2

A estas alturas ya sabemos que la Raspberry es similar a un mini ordenador que funciona con el sistema operativo Linux, mientras que Arduino es un microcontrolador ejecutando un programa sencillo con acceso a sus pines IO. Ambos se centran en diferentes áreas. En comparación con Arduino, la Raspberry consigue mucho mejor rendimiento a nivel de cálculo pudiendo ejecutar programas complejos.

Un ejemplo sencillo sería cuando en la construcción de un robot móvil se utiliza una Raspberry para ampliar su visión y analizar la imagen con algún algoritmo o librería pesada junto con

una webcam. Mientras tanto, Arduino maneja muy bien la parte de los motores y proporciona respuestas rápidas como, por ejemplo, para la evasión de obstáculos. Porque, sin el sistema operativo, el tiempo de respuesta a los cambios rápidos en el Arduino es mucho más corto. Por otra parte, el alto rendimiento de la Pi2, ofrece más rendimiento y sencillez para las comunicaciones inalámbricas y para la ejecución de algoritmos complejos de procesamiento de imágenes.

Aquí es donde entra la placa expansión (http://tienda.bricogeek.com/expansiones-raspberry-pi/667-arduino-expansion-shield-para-raspberry-pi-b.html) que combina un Arduino Leonardo (ATmega32U4) en formato shield para Raspberry. La placa permite conectar perfectamente la shield de Arduino, ya que incorpora sus zócalos hembra, y además ofrece la posibilidad de comunicación directa por USB con Pi (figura 7.12).

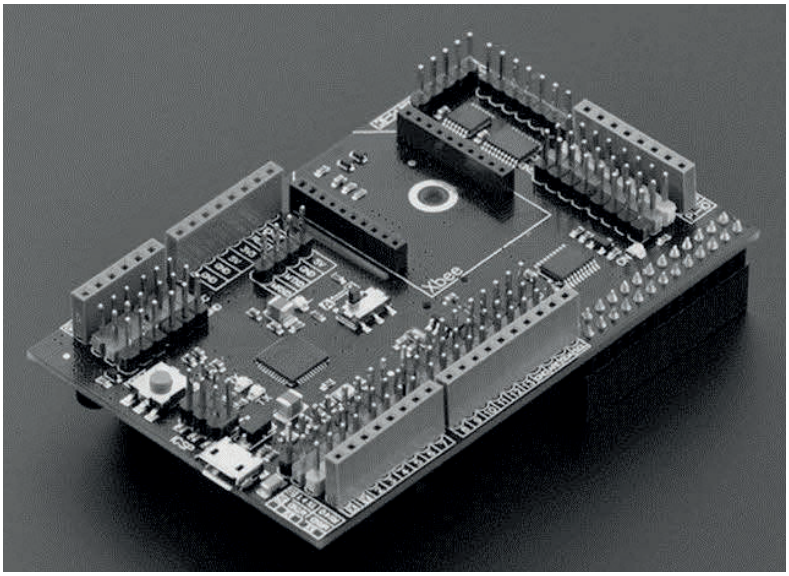


Figura 7.12

También dispone de cosas muy interesantes como un zócalo para módulos en formato XBee (válido para los BLE con ese formato o incluso Bluetooth), un reloj RTC DS1307 con batería incluida para conservar la hora y fecha, que no trae originalmente la Raspberry. También dispone de un conversor de niveles de 3,3 a 5 V para poder utilizar las shields o accesorios de Arduino que funcionen a 5 V. El diagrama de pines se muestra en la figura 7.13.

Características:

- ✓ Microcontrolador: ATmega32u4.
- ✓ Compatible con Arduino Leonardo.
- ✓ Pines hembra para shields de Arduino.
- ✓ Compatible con la mayoría de shields para Arduino (5 V).
- ✓ Tensión de funcionamiento: 5 V.
- ✓ RTC incorporada (DS1307) para que la Rasp guarde hora y fecha después de apagado.

- ✓ Zócalo extra con 8 pines GPIO de la Raspberry Pi.
- ✓ Puerto SPI, I2C y UART de la Raspberry Pi.
- ✓ Dimensiones: 88 × 56 × 17 mm.
- ✓ Cable USB.

Como observamos en la figura 7.13, podemos trabajar directamente con los pines GPIO de la Pi2, así como con los pines de Arduino, mediante su IDE corriendo bajo el entorno gráfico LXDE.

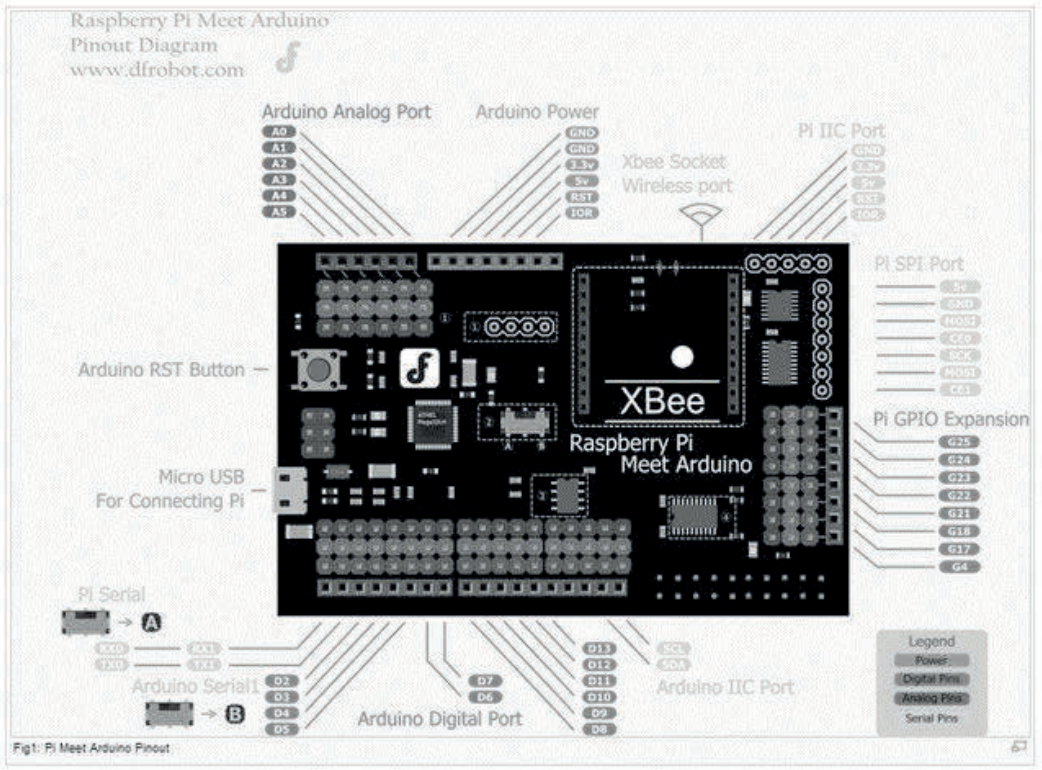


Figura 7.13

CAPÍTULO

8 SCRATCH Y RASPBERRY Pi2

Si entramos en el entorno gráfico de la Raspberry Pi2, habremos visto, aunque de pasada, el icono de un pequeño gatito en el menú de **Programing**, tal y como se ve en la figura 8.1:

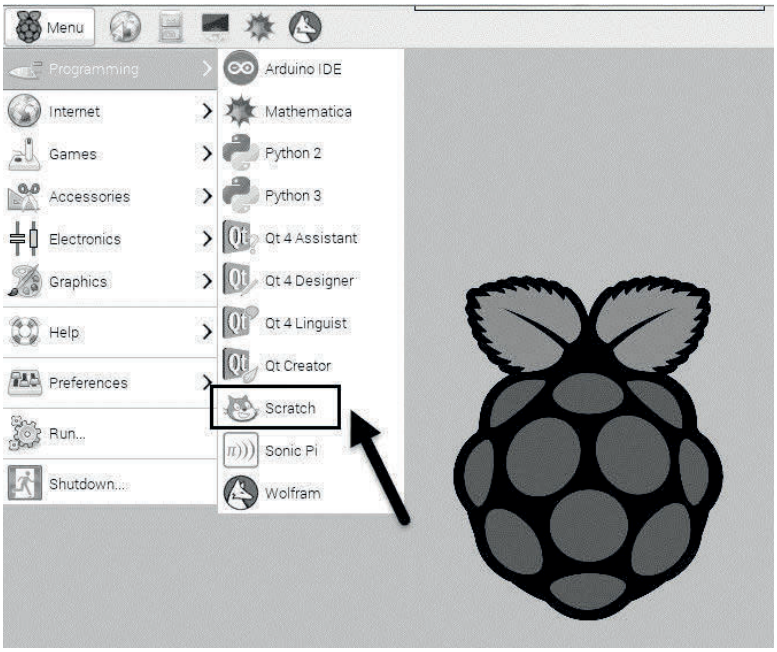


Figura 8.1

Se trata de un entorno de programación denominado **Scratch**. Scratch es un programa informático especialmente destinado a niños y niñas, que les permita investigar e introducirse en la programación de ordenadores utilizando una interfaz gráfica muy sencilla. Scratch está basado en el lenguaje de programación LOGO. Fue desarrollado por el Lifelong Kindergarten Group, viendo la luz por primera vez en 2007. Scratch es software libre y, por lo tanto, se puede redistribuir libremente e instalar en cualquier ordenador que tenga Windows, Mac OS X o Linux.

Scratch se utiliza en lugares muy diferentes y es válido para cualquiera de ellos, como en casa, la escuela, museos, etc. Está recomendado para niños/as entre 6 y 16 años, pero pueden utilizarlo personas de cualquier edad. A diferencia de un lenguaje de programación tradicional, Scratch no espera que el usuario memorice los nombres de las instruccione,s tales como **print** o **inkey$**. En vez de eso, casi todo se hace arrastrando y colocando bloques de código y arreglándolos de un modo lógico.

Las **características** más importantes de Scratch son:

- ✓ Este programa está basado en bloques gráficos y la interfaz que tiene es muy sencilla e intuitiva.
- ✓ Tiene un entorno colaborativo mediante el cual se pueden compartir proyectos, scripts y personajes en la web.
- ✓ El trabajo en Scratch se realiza mediante la unión de bloques que pueden ser eventos, movimientos de gráficos y sonidos.
- ✓ Los programas pueden ser ejecutados directamente sobre el navegador de Internet.

Sus **ventajas** son varias:

- ✓ Es un programa gratuito, de software libre.
- ✓ Es perfecto para enseñar y aprender a programar.
- ✓ Está disponible para varios sistemas operativos, Windows, Mac OS X y Linux.
- ✓ Permite compartir los proyectos a través de Internet, pudiendo ser descargados y utilizados por otras personas.
- ✓ Es multilenguaje.

La pregunta que nos asalta de inmediato es por qué está preinstalada en la Raspbian. La respuesta está relacionada con los orígenes de la Raspberry. Su objetivo inicial era eminentemente educativo. Se trataba de poner en manos de los niños y niñas una herramienta barata que les permitiera aprender programación de ordenadores con un lenguaje sencillo y visual.

8.1 SCRATCH Y LA ELECTRÓNICA

Vamos a particularizar el uso de Scratch en el terreno de la electrónica, y más concretamente en la utilización de los pines GPIO para el control de algunos dispositivos. Lo primero que tenemos que hacer es descargar e instalar una librería especializada en adecuar estos pines a Scratch. Se trata de la librería denominada **ScratchGPIO7**. Su descarga e instalación la realizamos desde la línea de comandos:

```
sudo wget http://bit.ly/1wxrqdp -O isgh7.sh
sudo bash isgh7.sh
```

Si ahora accedemos en el entorno gráfico de la Pi2, veremos los iconos de Scratch accesibles tal y como se muestra en la figura 8.2. Ahora entramos en el icono **ScratchGPIO7** y abrimos el ejemplo **blink11** con la opción **Open** bajo el menú archivo. La figura 8.3 muestra lo que nos aparece en pantalla.

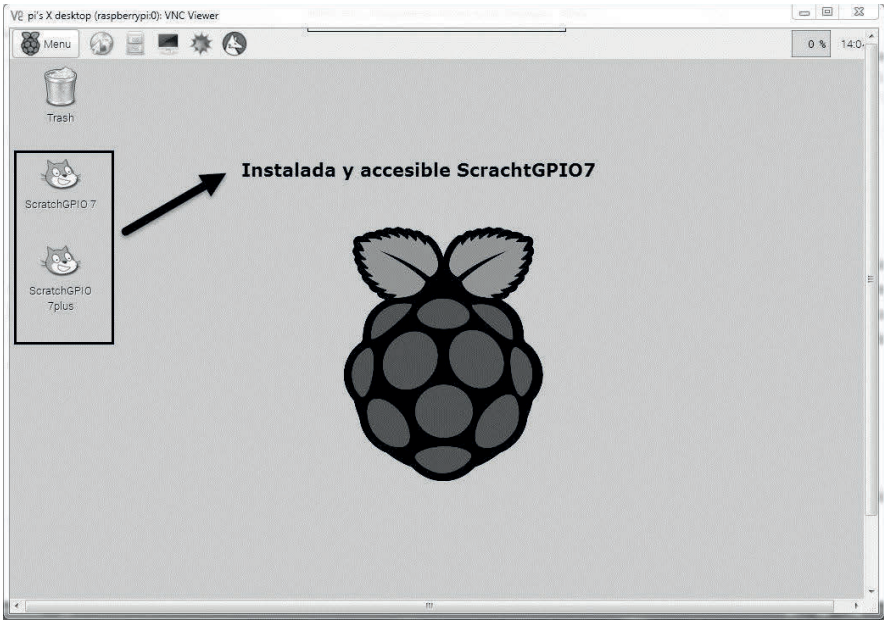


Figura 8.2

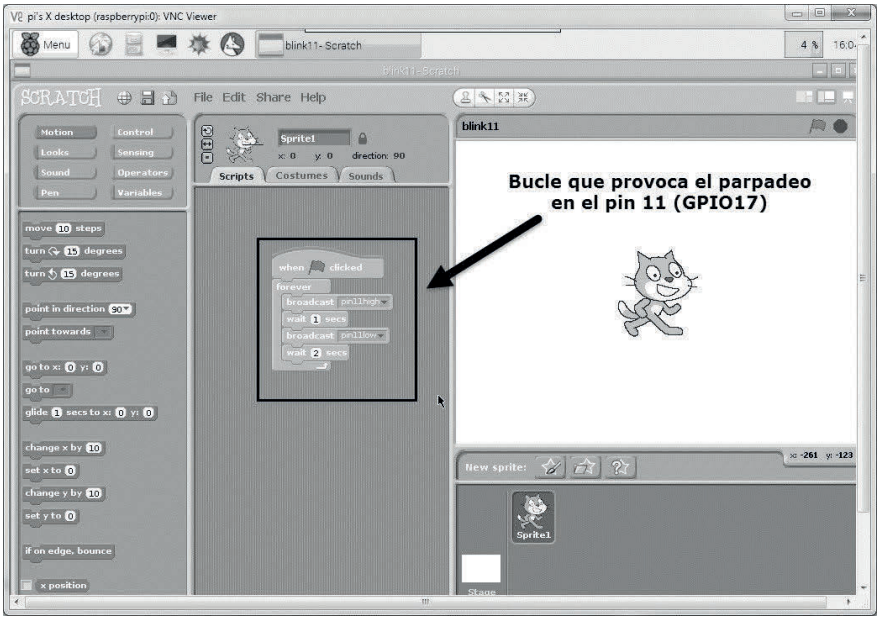


Figura 8.3

Si todo es correcto y tras una ventanita que nos avisa que el sensor remoto está habilitado, podemos pasar a conectar el hardware relativo a este ejemplo. Por supuesto, apagamos la Pi2 y conectamos un LED con una resistencia de 220 Ω al pin GPIO 17 que se corresponde con el pin 11 de Scratch. Observamos que no tenemos una correspondencia entre el número de GPIO y los pines de la placa. La equivalencia se muestra en la figura 8.4. Lo que hace este ejemplo **blink11** es simplemente hacer parpadear un LED. A continuación abordaremos algunos proyectos que a modo de ejemplo nos mostrarán el uso de la Pi2 con Scratch. Evidentemente suponemos que ya tenemos conocimientos básicos de la programación de Scratch. Existe amplia información en la red respecto a ello.

Los pines GPIO disponibles en Scratch son los siguientes:

- Salidas (21,18,16,15,13,12,11)
- Entradas (26,24,22,19,10,7)

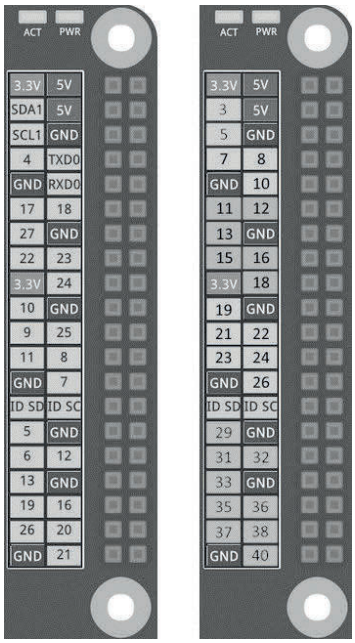


Figura 8.4

8.2 CONTROL SIMPLE DE UN LED

En este apartado vamos a controlar el encendido y apagado de un LED utilizando varias teclas del ordenador. Cuando presionemos la barra espaciadora o la tecla de la flecha hacia arriba del teclado, el LED se encenderá, y cuando lo hagamos con la flecha hacia abajo o la letra “a”, se apagará.

Scratch posee una función **BROADCAST** que permite enviar mensajes a los pines de la Raspberry de la siguiente manera tan simple:

```
BROADCAST PIN11HIGH // (pone alto el pin 11).
BROADCAST PIN11LOW // (pone bajo el pin 11).
```

Creamos un proyecto nuevo en Scratch y añadimos los bloques tal y como se muestran en la figura 8.5. Comprobamos el funcionamiento sobre el LED conectado al pin 11 de la Pi2.

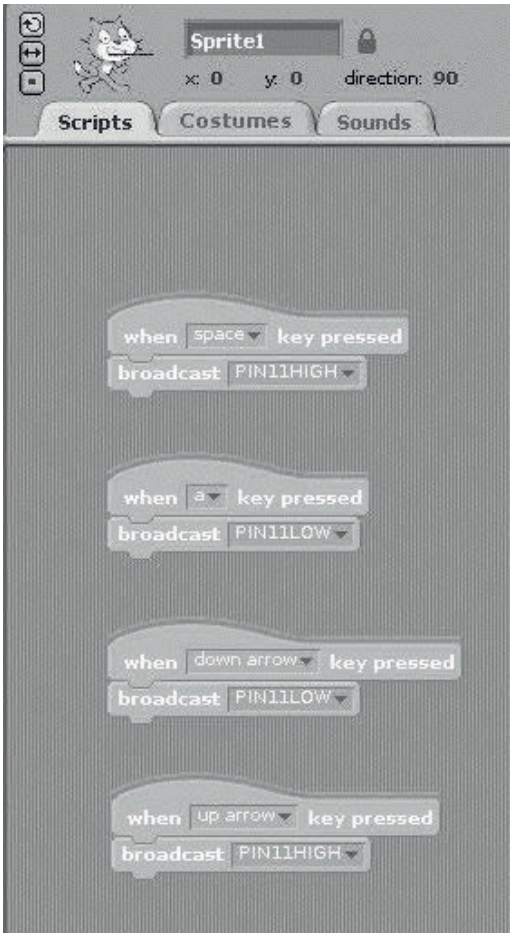


Figura 8.5

8.3 CONTROL DEL BRILLO POR PWM DE UN LED

De una forma sencilla podemos variar el brillo sobre un LED utilizando una variable denominada **power11**. Definimos esa variable e insertamos varias instancias de ella con tres valores a modo de ejemplo (25, 50 y 100). En la figura 8.6 observamos el conjunto de bloques y la creación de la variable.

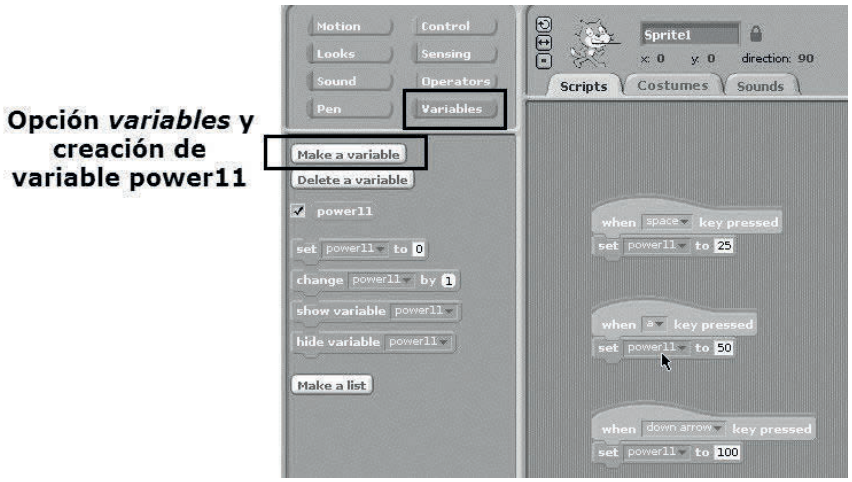


Figura 8.6

Cada vez que presionemos una de las tres teclas variaremos el brillo del LED de una manera similar a como lo haríamos mediante PWM (**Pulse Width Modulation**). La diferencia es que con la variable **power11** es mucho más fácil.

8.4 MANEJO DE UN INTERRUPTOR SIMPLE

Para comprobar una entrada (en nuestro caso un interruptor), hay que ir al bloque de detección y hacer clic en la palabra *slider* en la parte inferior y nos daremos cuenta de que tiene pines seleccionables (22, 7, 3, 5, 24, 26, 19, 21, 23, 8 y 10). Si conectamos un interruptor a uno de estos pines, entonces podremos detectar cuándo el interruptor está abierto o cerrado. En la figura 8.7 utilizamos el pin 10 (RXD en GPIO BCM).

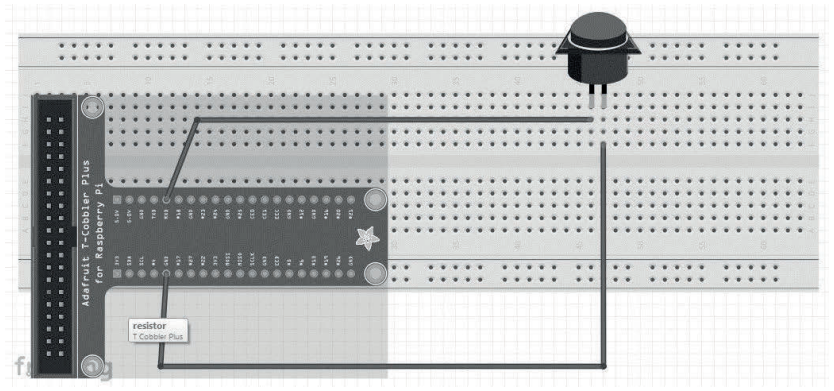


Figura 8.7

En las figuras 8.8 y 8.9 se puede observar el diagrama de bloques en el que le hacemos decir (*say*) al gatito que muestre un estado alto o un estado bajo en función de si el pulsador está abierto o cerrado.

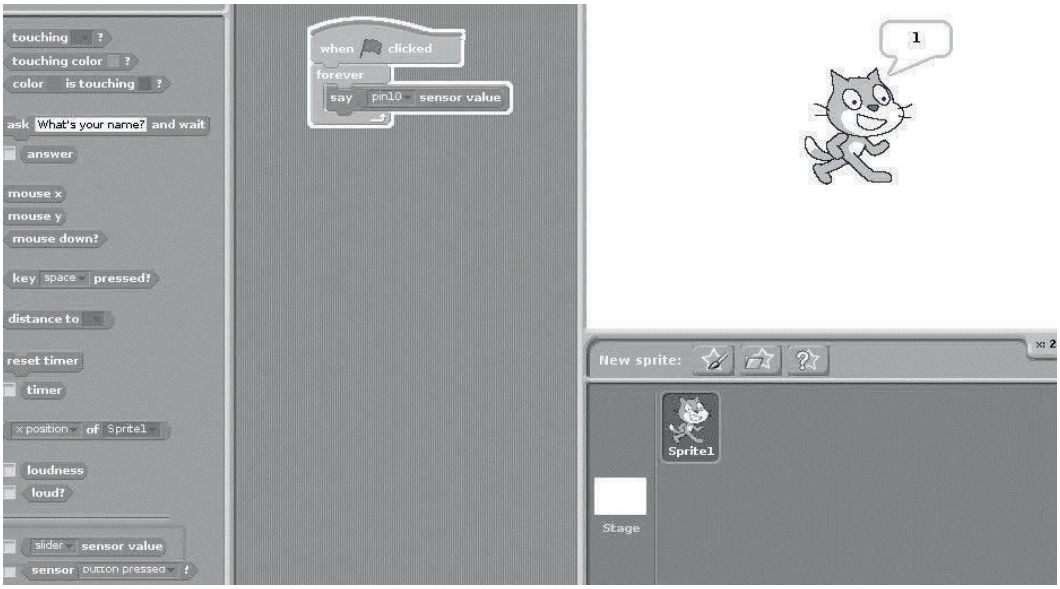


Figura 8.8

Figura 8.9

8.5 CONTROL DE UN MOTOR DC

Vamos a controlar la velocidad de un motor DC utilizando un driver de corriente típico como el ULN2003. Las conexiones se pueden observar en la figura 8.10 donde el motor es controlado por el pin 11 (GPIO17 BCM).

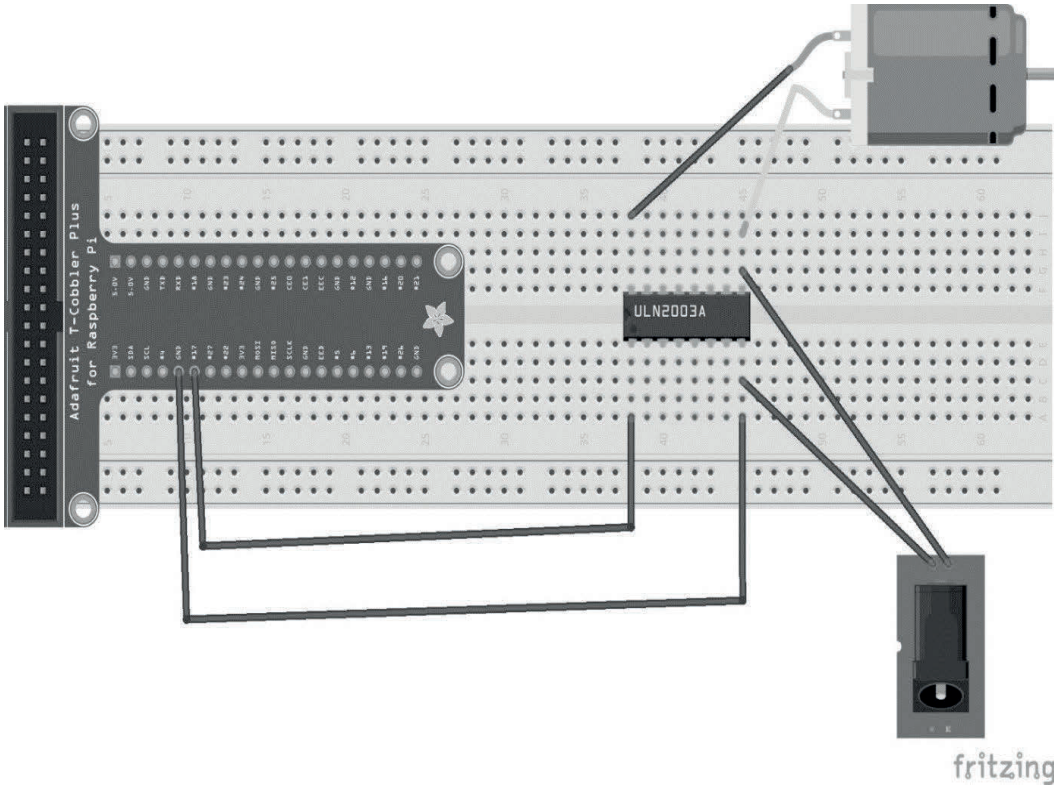


Figura 8.10

Utilizamos un alimentación externa de 6 V teniendo cuidado en su conexión. Aunque el motor se controla por el pin 11 se podría haber utilizado una variable llamada **Motor11**. De hecho, todos los pines en el Raspberry Pi2 se pueden controlar de esta manera. Para utilizar un motor en el pin 13 lo que haríamos simplemente es definir una variable llamada **Motor13**.

Para definir la velocidad del motor establecemos en la variable un valor de 0 a 100. El diagrama de bloques se puede observar en la figura 8.11.

El bucle va incrementando la velocidad en pasos de 1 segundo, desde 0 hasta la velocidad máxima.

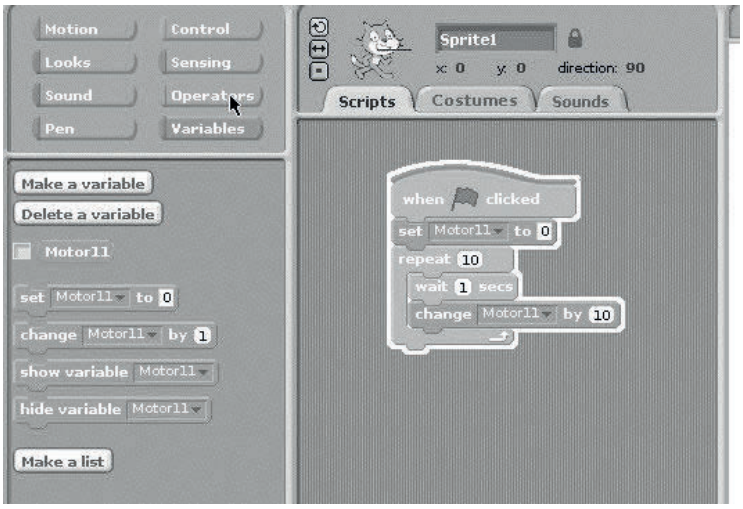


Figura 8.11

8.6 CONTROL DE SERVOS UTILIZANDO LA ARDAFUIT 16 SERVO/PWM HAT

Para usar un servo basta con crear una variable llamada **SERVOX** (donde X es el pin que ha conectado el cable de señal). Los valores servo son −90 y 90, con el valor 0 que lleva el servo a su posición central.

Para utilizar la HAT de Ardafruit (su funcionamiento lo hemos visto anteriormente) debemos definir las variables desde **AdaServo0** hasta **AdaServo15**. En el diagrama de bloques que se expone en la figura 8.12 utilizamos solo un servo creando para él la variable **AdaServo0**. Se muestran tres posibilidades de funcionamiento. Con la barra espaciadora, el servo se posiciona en el centro. Con la tecla de la flecha hacia arriba, gira en un sentido, y con la flecha hacia abajo, en sentido contario. Es necesario, desde luego, ajustar los valores para calibrar cuánto gira.

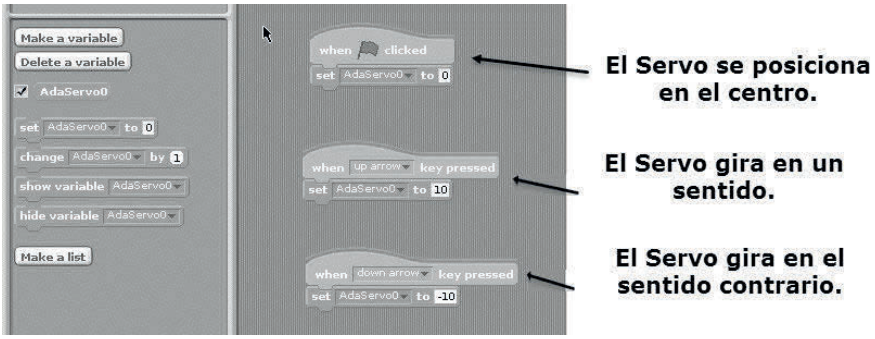


Figura 8.12

Evidentemente el propósito de este libro no es exponer detalladamente todas las posibilidades del uso de Scratch con la Pi2, sino solo mostrar al lector sus enormes posibilidades, sobre todo, en el campo de la educación. La completa compatibilidad y flexibilidad de los pines GPIO es, sin duda, una de las tareas que quedan pendientes a los desarrolladores y que deberán afrontar en un futuro inmediato.

CAPÍTULO

9

INTERFAZ GRÁFICA (GUI) CON TKINTER

En este capítulo aprenderemos a construir GUIS para que nuestros proyectos posean una apariencia amigable y cómoda. Hasta ahora la interfaz con el usuario era a través de la consola o terminal. Llegó el momento de abordar la elaboración de entornos gráficos de ventanas que dotan a nuestras aplicaciones de un aspecto profesional.

La interfaz gráfica de usuario, conocida también como GUI (**Graphical User Interface**), es una aplicación que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

En este capítulo aprenderemos a crear aplicaciones gráficas (GUI) paso a paso con Python, utilizando como herramienta gráfica **Tkinter**. Existen otras opciones; la más popular es el entorno de programación gráfica en Python **PyQt** basada en la librería Qt. Sin embargo, es también más complicada de utilizar y requiere conocimientos más avanzadas.

Tkinter es un **binding** de la biblioteca gráfica **Tcl/Tk** para el lenguaje de programación Python. Un **binding** es una adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquel en el que ha sido escrita.

Evidentemente, el presente libro no va cubrir, ni mucho menos, todos los entresijos de la programación en Tkinter. El objetivo es sentar las bases de su filosofía e introducirnos a través de varios ejemplos en su metodología. Existen en la red amplios y variados tutoriales que nos profesionalizan en su programación.

9.1 INSTALACIÓN DE PYTHON 3.4.X

Antes de nada es de vital importancia reseñar que vamos a utilizar la nueva versión de **Python 3.4** ya que supone importantes mejoras con respecto a la versión hasta ahora utilizada a lo largo del presente libro. No todas las librerías han sido migradas a esta versión, pero es el presente y el futuro de Python, razón por la cual es la que vamos a emplear en este capítulo. Las diferencias entre ambas, aunque pocas, se pueden encontrar en el siguiente enlace:

https://www.raspberrypi.org/documentation/usage/python/more.md

La versión 3.4 ya viene preinstalada con la Raspbian, pero si tenemos una versión antigua de esta imagen, para descargarla e instalarla en nuestra Raspberry Pi2, abrimos un terminal y tecleamos lo siguiente:

```
sudo apt-get install python3
```

9.2 INTRODUCCIÓN A TKINTER

La capacidad de programar una aplicación de interfaz gráfica de usuario (en oposición a una simple aplicación de consola) abre todo un mundo de posibilidades para un programador. Se cambia el enfoque del programador y se focaliza en el usuario final, lo que le permite llegar a un público más amplio. Tkinter es la forma más fácil y rápida de hacer este trabajo. Tkinter es una gran herramienta para la programación de aplicaciones GUI en Python.

Las características que hacen Tkinter una gran opción para la programación GUI incluyen:

- ✓ Es fácil de aprender (más simple que cualquier otro paquete de GUI para Python).
- ✓ Relativamente con poco código puede producir aplicaciones GUI poderosas.
- ✓ El diseño en capas asegura que es fácil de entender.
- ✓ Es portable a través de todos los sistemas operativos.
- ✓ Es de fácil acceso, ya que viene pre-instalado con la distribución estándar de Python.
- ✓ Ninguno de los otros conjuntos de herramientas GUI tiene todas estas características al mismo tiempo.

Ahora vamos a empezar creando una simple ventana vacía y de paso comprobamos que Tkinter está completamente operativo en nuestra Pi2. Para ello abrimos Python3 IDLE en el entorno gráfico LXDE y tecleamos los siguientes comandos:

```
from tkinter import *
root = Tk()
root.mainloop()
```

Guardamos esto con la extensión de archivo .py. Ejecutamos el programa desde el menú **Ejecutar** (F5 en IDLE). La ejecución de este programa debe generar una ventana raíz en blanco como se muestra en la figura 9.1. Esta ventana está construida con las funciones de minimizar, maximizar y cerrar como cualquier ventana tipo Windows.

La primera línea importa todos (*): clases, atributos y métodos de Tkinter en el espacio de trabajo actual. La segunda línea crea una instancia de la clase **Tkinter.Tk**. Esto genera lo que se llama la ventana **raíz** que se ve en la figura. Por convención, la ventana raíz en Tkinter, generalmente se llama **root**, pero somos libres de llamarla con cualquier otro nombre. La tercera línea ejecuta el **Mainloop** (es decir, el bucle de eventos), método de la raíz objeto. El método Mainloop es lo que mantiene la ventana raíz visible. Si quitamos la tercera línea, la ventana creada en la línea 2 desaparecerá inmediatamente. Esto sucederá tan rápido que ni siquiera veremos la ventana que aparece en la pantalla. Mantener el Mainloop en funcionamiento nos permite mantener el programa que se ejecuta hasta que se pulsa el botón de cierre, que sale del bucle principal.

Figura 9.1

Como programadores GUI, por lo general, somos responsables de resolver tres aspectos de nuestro programa:

- ¿Qué componentes deben aparecer en la pantalla? Se trata de la elección de los componentes que conforman la interfaz de usuario. Los componentes típicos incluyen cosas tales como botones, campos de entrada, casillas de verificación, botones de radio, barras de desplazamiento, y similares. En Tkinter, los componentes que se agregan a su interfaz gráfica de usuario se llaman **widgets**.

- ¿Dónde deben ir los componentes? Se trata de decidir el posicionamiento o la colocación de cada componente en la estructura general de diseño. Esto incluye que se tomen decisiones en temas de posicionamiento y el diseño estructural de varios componentes. En Tkinter, esto se conoce como la gestión de la geometría.

- ¿Cómo interactúan los componentes y se comportan? Esto implica agregar funcionalidad a cada componente. Cada componente o widget hace algún trabajo. Por ejemplo, al hacer clic sobre un botoón se produce una respuesta; una barra de desplazamiento se encarga del desplazamiento; y casillas de verificación y botones de radio permiten al usuario tomar algunas decisiones.

Vamos a profundizar más en cada uno de estos tres componentes en el contexto de Tkinter. Ahora que tenemos nuestra ventana principal preparada, es tiempo de pensar sobre los componentes que deben aparecer en la ventana, es decir, sobre los widgets.

9.3 PRINCIPALES WIDGETS EN TKINTER

La mayoría de los lenguajes gráficos de programación proporcionan una biblioteca de widgets para que podamos utilizarlos en nuestros programas. Aunque no es una norma oficial, existe un conjunto común de widgets disponibles en casi todos los entornos de programación gráfica. La tabla 9.1 enumera los widgets más comunes en la programación Python GUI.

Frame	Proporciona un área principal de ventana.
Label	Coloca texto en el área de ventana.
Button	Proporciona un evento determinado al presionar sobre él.
CheckButton	Permite al usuario seleccionar o deseleccionar un elemento.
Entry	Proporciona al usuario un área para entrar o visualizar una línea de texto.
ListBox	Muestra varios valores para seleccionar.
Menu	Crea un menú de barra de herramientas en la parte superior de la ventana.
RadioButton	Permite al usuario selecciona una opción de varias disponibles.
Scrollbar	Permite visualizar varios elementos de una lista.
Separator	Coloca una línea o barra vertical u horizontal en una ventana.
Text	Permite introducir varias líneas de texto en un área determinada.

Tabla 9.1

Cada widget tiene su propio conjunto de propiedades que definen la forma en que aparece en la ventana del programa y establece cómo manejar cualquier dato o acciones que tengan lugar mientras el usuario interactúa con esa ventana.

La programación para un entorno GUI es un poco diferente de la programación de línea de comandos en la manera que Python maneja el código del programa. En un programa de líneas de comandos, el orden del código del programa controla qué sucede después. Por ejemplo, el programa pide al usuario una entrada, procesa dicha entrada y luego muestra los resultados en la línea de comandos. El usuario del programa solo puede responder a las peticiones de entrada del programa.

Por el contrario, un programa de interfaz gráfica de usuario muestra un conjunto completo de interacción con todos los widgets a la vez; todos en la misma ventana. El usuario del programa puede decidir cuál de los widgets se procesa a continuación. Debido a que el código no sabe a priori qué widget activará el usuario en un momento dado, tiene que utilizar una función denominada **event driven programing** (programación orientada a eventos) para procesar código. En la programación orientada a eventos, Python llama a diferentes métodos dentro el programa, en base a qué evento (o acción) ocurre en la ventana de interfaz gráfica de usuario. No hay un flujo conjunto de código de programa; consiste en un conjunto de métodos que se ejecutan de forma individual en respuesta a un evento.

Por ejemplo, un usuario puede introducir datos en un widget de texto, pero no pasa nada hasta que dicho usuario pulsa un botón en la ventana del programa que visualizará el texto. El botón activa

un evento y su código de programa debe detectar ese evento y luego ejecutar el método de código para leer el texto en el campo de texto y procesarlo. La clave para la programación orientada a eventos es la vinculación de los widgets en la ventana para eventos para luego unir los eventos a los módulos de código en el programa. Un controlador de eventos está a cargo de este proceso. Para que el programa funcione, debemos crear módulos separados en Python. Los controladores de eventos hacen la mayor parte del trabajo en los programas de interfaz gráfica de usuario. Ellos recuperan los datos de los widgets, procesan los datos y, a continuación, muestran los resultados en la ventana, usando otros widgets. Esto puede parecer un poco complicado al principio, pero una vez que nos acostumbramos a la codificación con eventos, todo se vuelve más fácil a la hora de trabajar en un entorno gráfico.

9.4 UTILIZANDO TKINTER

Es necesario seguir tres pasos básicos para crear una aplicación GUI utilizando el paquete tkinter:

- ✓ Crear una ventana.
- ✓ Añadir widgets a la ventana.
- ✓ Definir los controladores de eventos para los widgets.

En los siguientes apartados veremos cada uno de estos pasos para mostrar cómo se construye una aplicación de interfaz gráfica de usuario utilizando Tkinter con los scripts de Python.

9.4.1 Creación de una ventana

En un entorno gráfico, todo gira en torno a una ventana. El primer paso para la creación de un programa de interfaz gráfica de usuario es la creación de la ventana principal de la aplicación, llamada **la ventana raíz**. Lo hacemos mediante la creación de un objeto de **Tk**, que controla todos los aspectos de nuestra ventana. Para crear un objeto Tk, primero tenemos que importar la biblioteca Tkinter para luego crear instancias de ese objeto Tk como el que se muestra:

```
from Tkinter import *
root = Tk()
```

Esto crea un objeto de la ventana principal y la asigna a la variable llamada **raíz**. Sin embargo, este valor predeterminado de ventana no tiene tamaño, título o característica alguna. Necesitamos ejecutar algunos métodos del objeto Tk para la ventana y configurar algunas de sus características como ventana que es. Existen dos métodos comunes para esto que son: el método del **título()** para establecer el título de la ventana (aparecerá en la barra de título, en la parte superior de la ventana) y el método de la **geometría()** (define el tamaño de la ventana).

```
root.title('Esto es una prueba de ventana')
root.geometry('300x100')
```

Después de configurar estos métodos, es necesario utilizar el método **mainloop ()**, lo que pone la ventana en bucle como vimos anteriormente. Ahora esperamos que se desencadene un evento.

Cuando se produzcan eventos en esta ventana, Python los intercepta y los pasa a su código de programa. Por ejemplo, si hacemos clic en la X (en la esquina superior derecha de la ventana), Python

capta ese evento y sabe cerrar la ventana. El listado completo se muestra a continuación. El resultado de ejecutarlo en el entorno gráfico de nuestra Pi2 se observa en la figura 9.2.

```
from tkinter import *
root= Tk()
root.title('Esto es una prueba de ventana')
root.geometry('300x100')
root.mainloop()
```

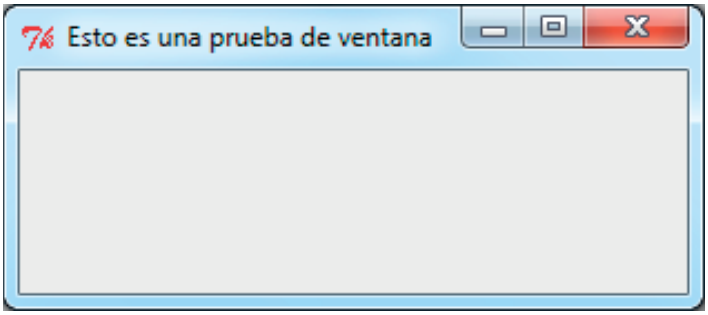


Figura 9.2

9.4.2 Añadiendo widgets a la ventana

Después de crear la ventana raíz, ya estamos listos para empezar a trabajar con los widgets para nuestra interfaz. Existen tres pasos a seguir para añadir widgets a una ventana:

- ✓ Creamos una plantilla de marco en la ventana raíz.
- ✓ Definimos un método de posicionamiento para colocar los widgets en ese marco.

Colocamos los widgets en el marco utilizando el método de posicionamiento elegido.

El primer paso en el proceso de añadir widgets a la ventana es crear una plantilla para el diseño de los widgets de esa ventana. El paquete Tkinter utiliza el objeto **Frame** para generar un área para que coloquemos widgets en la ventana. Sin embargo, no se utiliza el objeto Frame directamente en el código de ventana; en su lugar debemos crear, en base a la clase Frame, una clase "hija" para definir todos los métodos y atributos de la ventana.

```
class Application(Frame):
```

Después de establecer la clase "hija", es necesario crear un constructor para ello. Si le echamos un vistazo a algún manual básico de programación orientada a objetos, veremos que se define un constructor con el método: **__init__ ().** Este método utiliza la palabra clave **self** como el primer parámetro y se necesita la raíz **Tk** del objeto que se creó como segundo parámetro. Ahora tenemos una plantilla básica que se puede utilizar para crear una clase del marco o Frame de la ventana:

```
class Application(Frame):
    """Mi aplicación de ventana"""
    def __init__(self, master):
         super(Application, self).__init__(master)
         self.grid()
```

La definición de clase para crear la ventana y el marco no es muy larga, pero es algo complicada. El constructor que generamos para la clase de aplicaciones contiene dos sentencias. La sentencia **super()** importa el método constructor de la clase principal **Frame** para la clase de aplicaciones, pasando el objeto ventana raíz. La última declaración en el constructor define el método de posicionamiento utilizado para el marco. En este ejemplo se utiliza el método de la cuadrícula **tkinter()**. Ahora tenemos nuestra plantilla general de clase de aplicaciones que podemos utilizar para crear una ventana (figura 9.3).

```
from tkinter import *
class Application(Frame):
      def __init__(self, master):
      super(Application, self).__init__(master)
      self.grid()
root = Tk()
root.title('Test de la ventana de aplicacion')
root.geometry('300x100')
app = Application(root)
app.mainloop()
```

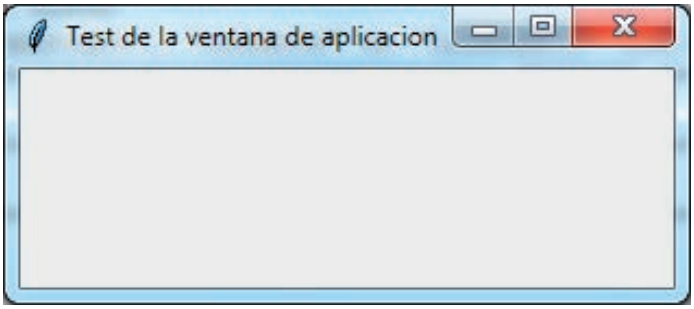


Figura 9.3

La clave para una aplicación con interfaz gráfica fácil de usar es la colocación de los widgets en el área de la ventana. Por el contrario, muchos widgets agrupados pueden hacer que la interfaz de usuario sea confusa. En el ejemplo de la sección anterior se utilizó el método de la **rejilla()** (*grid*) para colocar los widgets en el marco. El paquete Tkinter ofrece tres maneras de colocar los widgets en la ventana:

- El uso de un sistema de rejilla.
- La colocación los widgets en espacios libres.
- El uso de valores posicionales.

El último método, utilizando los valores de posición, requiere que se defina la ubicación precisa de cada widget, usando coordenadas **x** e **y** dentro de la ventana. Si bien esto proporciona el control más preciso sobre donde aparecen nuestros widgets, puede ser un poco difícil cuando se está empezando a programar.

El método de colocación más o menos hace lo que dice: Se trata de empacar u organizar widgets en una ventana de la mejor manera que puede en el espacio disponible. Cuando elegimos este método, Python coloca los widgets de la ventana por nosotros, comenzando en la parte superior izquierda y moviéndose a lo largo del próximo espacio disponible, ya sea a la derecha o a continuación del widget anterior. El método de "embalaje" funciona bien para las pequeñas ventanas con solo algunos widgets, pero si tenemos una ventana más grande, las cosas se pueden volver rápidamente desordenadas y desalineadas.

El compromiso entre el método posicional y el método de colocación es el método de la cuadrícula. El método de la cuadrícula crea un sistema de rejilla en la ventana, con filas y columnas, algo así como una hoja de cálculo. Colocamos cada widget de la ventana en un lugar específico de filas y columnas. Podemos definir un widget para abarcar varias filas o columnas, por lo que tiene una cierta flexibilidad en la forma en que aparecen estos.

El método de la cuadrícula (**grid()**) define tres parámetros para colocar el widget en la ventana:

```
object.grid(row = x, column = y, sticky = n)
```

Los valores de fila (**row**) y columna (**column**) se refieren a la ubicación de la celda en la disposición, entendiendo la celda superior izquierda de la ventana con los valores asignados de fila 0 y columna 0. El parámetro **sticky** le dice a Python cómo alinear el widget en el interior la celda. Hay nueve posibles valores de sticky:

- **N**: Coloca el widget en la parte superior de la celda.
- **S**: Coloca el widget en la parte inferior de la celda.
- **E**: Alinea a la derecha el widget en la celda.
- **W**: Alinea a la izquierda el widget en la celda.
- **NE**: Coloca el widget en la esquina superior derecha de la celda.
- **NW**: Coloca el widget en la esquina superior izquierda de la celda.
- **SE**: Coloca el widget en la esquina inferior derecha de la celda.
- **SW**: Coloca el widget en la esquina inferior izquierda de la celda.
- **CENTER**: Centra el widget en la celda.

Ahora que tenemos un objeto marco o Frame y un método de posicionamiento, ya estamos listos para comenzar a colocar algunos widgets dentro de nuestra ventana. Podemos definir los widgets directamente en el constructor de la clase para la clase de Aplicación; pero, por otra parte, se ha convertido en algo habitual en los círculos de Python crear un método especial llamado

create_widgets(), y luego colocar las declaraciones para crear los widgets dentro de ese método. Solo llamamos al método create_widgets() desde el interior del constructor de la clase. Al agregar el create_widgets() para la clase de Aplicación, el constructor tiene una apariencia parecida a esto:

```
def __init__(self, master):
      super(Application, self).__init__(master)
      self.grid()
      self.create_widgets()
```

El método create_widgets() contiene todas las declaraciones para construir los objetos de widgets que deseemos que aparezcan en la ventana. Un ejemplo que resume todo lo anterior se muestra a continuación:

```
from tkinter import *
class Application(Frame):
      def __init__(self, master):
            super(Application, self).__init__(master)
            self.grid()
            self.create_widgets()
      def create_widgets(self):
            self.label1 = Label(self, text='Hola mi Ventana')
            self.label1.grid(row=0, column=0, sticky= W)
root = Tk()
root.title('Test de la ventana de aplicacion')
root.geometry('300x100')
app = Application(root)
app.mainloop()
```

El resultado de su ejecución en la figura 9.4:

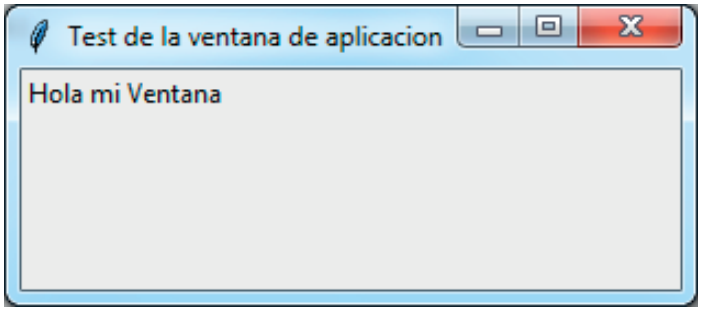


Figura 9.4

El método create_widgets() contiene dos líneas de código para definir un objeto widget de Etiqueta o **Label** para la ventana. Por una parte definimos el objeto **Label** real y por la otra se aplica el método de la cuadrícula para posicionar el widget Label alineado a la izquierda en la ventana.

El siguiente paso en la construcción de una aplicación GUI es definir los eventos que utiliza la ventana. Los widgets que pueden generar eventos (por ejemplo, cuando el usuario hace clic en un botón de la aplicación) utilizan el parámetro de comando para definir el nombre de un método que Python llama cuando detecta dicho evento. Por ejemplo, para enlazar un botón a un método de evento, podemos escribir un código como este:

```
def create_widgets(self):
     self.button1 = Button(self, text="Submit", command =
     self.display)
     self.button1.grid(row=1, column=0, sticky = W)
def display(self):
      print("El Botón ha sido presionado")
```

El método create_widgets() crea un solo botón para mostrarse en el área de la ventana. El constructor de la clase **Button** establece el parámetro de comando para **self.display** que apunta al método de visualización en dicha clase. Por ahora, el método de visualización de prueba (**display**) solo utiliza una sentencia **print()** para mostrar un mensaje en la línea de comandos donde se inició el programa. Ahora las cosas están empezando a verse más como una interfaz gráfica de usuario que como un programa en línea. El listado completo se muestra a continuación:

```
from tkinter import *
class Application(Frame):
      def __init__(self, master):
            super(Application, self).__init__(master)
            self.grid()
            self.create_widgets()
      def create_widgets(self):
           self.label1 = Label(self, text='Welcome to my
            window!')
           self.label1.grid(row=0, column=0, sticky=W)
           self.button1 = Button(self, text='Presioname',
           command=self.display)
           self.button1.grid(row=1, column=0, sticky=W)
      def display(self):
           print('El botón has sido presionado')

root = Tk()
root.title('Evento del Botón')
root.geometry('300x100')
```

```
app = Application(root)
app.mainloop()
```

El resultado de su ejecución, así como el terminal de comandos, se pueden observar en las figuras 9.5 y 9.6:

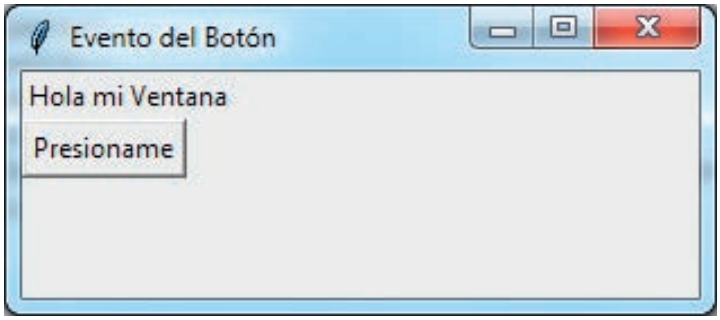


Figura 9.5

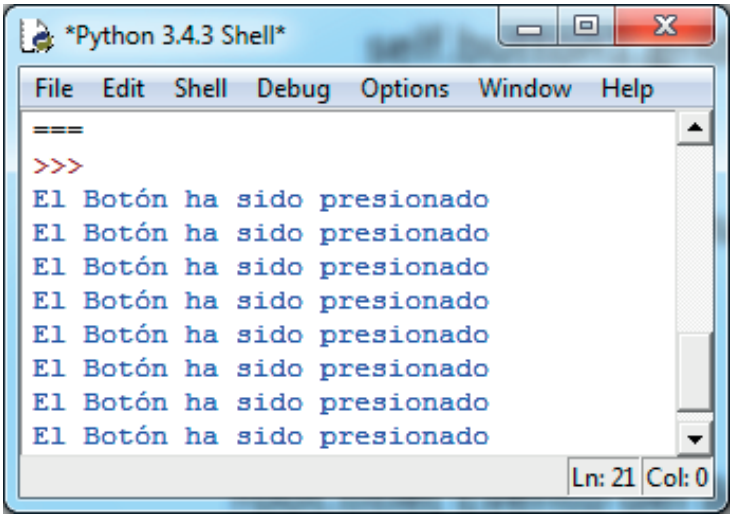


Figura 9.6

El widget de Etiqueta nos permite colocar texto dentro de una ventana. Este widget se utiliza a menudo para identificar otros widgets, como las zonas de entrada o cuadros de texto. Para agregar un widget de Etiquetas a la ventana, se define el texto que se mostrará con el parámetro que se muestra a continuación:

```
self.label1 = Label(self, text='Esto es una etiqueta')
```

Los botones proporcionan una forma para que los usuarios activen los eventos en una aplicación. Este es el formato básico para la creación de un widget de Botón.

```
self.button1=Button(self,text='Enviar', command=self.calculate)
```

Debemos asignar el widget **Button** a un nombre de variable única dentro de la clase de Aplicación. Con el widget de Botón, nos aseguraremos de señalar con el parámetro de comando el método asociado al controlador de eventos. Si no se especifica el parámetro de comando, el botón no realiza ninguna acción cuando se hace clic. Además, es necesario utilizar el método de la cuadrícula (*grid*) para situar el botón donde se desee en el área del marco o frame de la ventana.

El widget de **CheckButton** proporciona un tipo ON u OFF en la interfaz. Si el widget CheckButton es seleccionado, devuelve un valor 1, y si el widget no está marcado, se devuelve un valor 0. Los widget de CheckButton son comúnmente utilizados para hacer las selecciones de uno o más elementos de una lista (tales como la selección de los ingredientes en una pizza). El trabajo con el widget CheckButton es un poco complicado. No se puede acceder directamente al widget de CheckButton para averiguar si se ha seleccionado. En lugar de ello, es necesario crear una variable especial que puede contener un valor que representa el estado de la casilla de verificación. Esto se conoce como una variable de control. Se puede crear una variable de control mediante el uso de uno de los cuatro métodos especiales:

- **BooleanVar ()**: Para valores booleanos 0 y 1.
- **DoubleVar ()**: Para los valores de punto flotante.
- **IntVar ()**: Para valores enteros.
- **VarCadena ()**: Para los valores de texto.

Porque el widget CheckButton devuelve un valor booleano, se debe utilizar el método de control de variable **BooleanVar()**.

```
self.varCheck1 = BooleanVar()
```

Después de vincular la variable de control para el widget CheckButton, se utiliza el parámetro variable cuando se define el widget CheckButton.

```
self.check1=Checkbutton(self,text='Opcion1',
variable=self.varCheck1)
```

Así que con el widget CheckButton, en lugar de utilizar un controlador de eventos, se vincula el widget a una variable de control. El parámetro de texto en el objeto CheckButton define el texto que aparece a continuación de la casilla de verificación en la ventana. Para recuperar el estado del widget CheckButton en su código, es necesario utilizar el método **get()** para la variable de control de la siguiente manera:

```
opcion1 = self.varCheck1.get()

if (opcion1):
```

```
        print('El checkboton ha sido seleccionado')
    else:
        print('El checkbutton no ha sido seleccionado')
```

Resumimos lo explicado hasta ahora con un ejemplo más completo. El listado se muestra a continuación. No hace falta decir que es recomendable probarlo.

```
from tkinter import *
class Application(Frame):

def __init__(self, master):
        super(Application, self).__init__(master)
        self.grid()
        self.varSalchicha = IntVar()
        self.varPepp = IntVar()
        self.create_widgets()

def create_widgets(self):
        self.label1=Label(self, text='¿Qué quieres en tú
pizza?')
        self.label1.grid(row=0)
        self.check1=Checkbutton(self, text='Salchicha',
        variable =self.varSalchicha)

        self.check2=Checkbutton(self, text='Pepperoni',
        variable =self.varPepp)

        self.check1.grid(row=1)
        self.check2.grid(row=2)
        self.button1=Button(self,text='Ordenar',
        command=self.display)

        self.button1.grid(row=3)

def display(self):
        if (self.varSalchicha.get()):
                print('Tu deseas salchicha')
        if (self.varPepp.get()):
                print('Tu deseas pepperoni')
        if(notself.varSalchicha.get()andnot
self.varPepp.get()):
```

```
            print("Tu no quieres nada pizza?")
            print('----------')

root = Tk()
root.title('Testo de eventos Checkbutton')
root.geometry('300x100')
app = Application(root)
app.mainloop()
```

El resultado de su ejecución, así como el terminal de comandos, se pueden observar en las figuras 9.7 y 9.8:

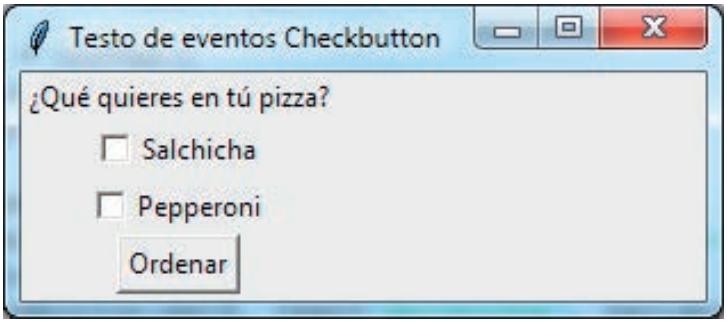


Figura 9.7

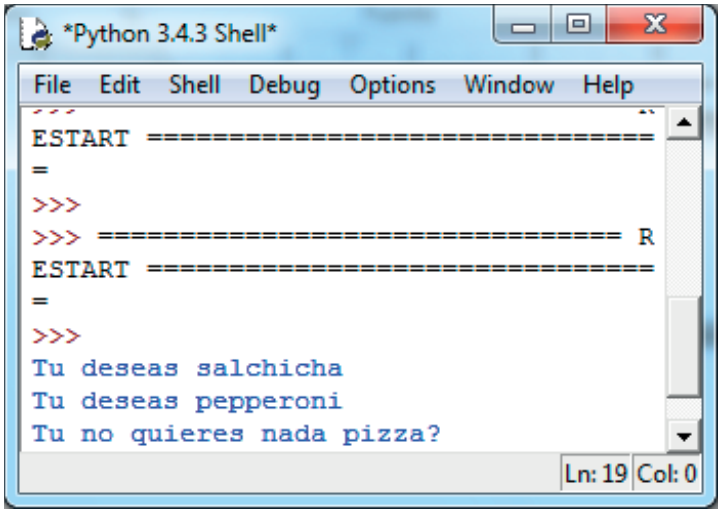


Figura 9.8

El control **Entry** es uno de los widgets más versátiles que utilizaremos en nuestras aplicaciones.

Se crea un campo de formulario de una sola línea. Un usuario del programa puede utilizar este campo para introducir texto para enviarlo al programa, o su programa puede emplearlo para mostrar el texto de forma dinámica en la ventana. La creación de un widget de entrada no es muy complicado.

```
self.entry1 = Entry(self)
```

El control Entry normalmente se vincula a otro widget, como un botón, un evento que recupera el texto que está en dicho control o muestra un nuevo texto. Para ello, es necesario utilizar **get()** del widget **Entry()** para recuperar el texto en el campo de formulario o usar el método **insert()** para mostrar texto en el campo de formulario.

Completamos el anterior listado del programa con este nuevo widget:

```
from tkinter import *

class Application(Frame):

      def __init__(self, master):
            super(Application, self).__init__(master)
            self.grid()
            self.create_widgets()

     def create_widgets(self):
           self.label1= abel(self,text='Entrar un texto en
            minúsculas'´)

            self.label1.grid(row=0)
            self.text1 = Entry(self)
            self.text1.grid(row=2)
            self.button1=Button(self,text='Convertir el
texto',
            command=self.convert)
            self.button1.grid(row=6, column=0)
            self.button2=Button(self, text='Limpiar
            resultado',
            command=self.clear)
            self.button2.grid(row=6, column=1)
            self.text1.focus_set()

     def convert(self):
```

```
            varText = self.text1.get()
            varReplaced = varText.upper()
            self.text1.delete(0, END)
            self.text1.insert(END, varReplaced)

    def clear(self):
            self.text1.delete(0,END)
            self.text1.focus_set()

root = Tk()
root.title('Testeo de Entry widget')
root.geometry('500x200')
app = Application(root)
app.mainloop()
```

El resultado de su ejecución, así como el terminal de comandos, se pueden observar en las figuras 9.9 y 9.10:

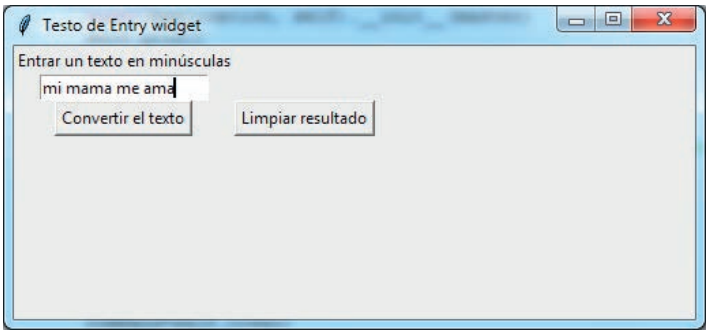


Figura 9.9

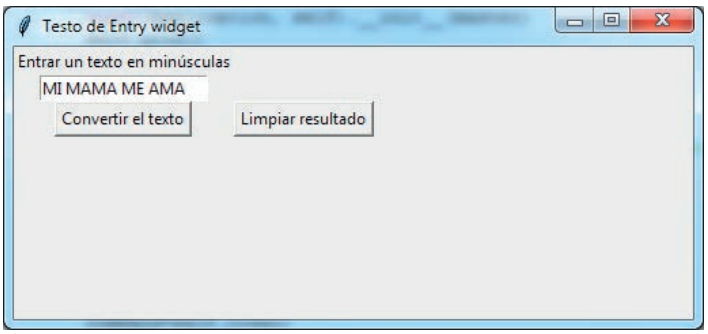


Figura 9.10

Como podemos observar, se trata de un programa que convierte un texto escrito en minúsculas en el mismo texto en mayúsculas. El método **focus_set()** es una herramienta muy útil. Permite que le digamos a la ventana que el widget obtenga el control del cursor, obligando al usuario tener que hacer clic en el primer widget.

Para introducir grandes cantidades de texto, puede utilizar el widget de texto si prevé la entrada de texto de varias líneas o se presentan varias líneas de texto. El widget de texto tiene la siguiente sintaxis:

```
self.text1 = Text(self, options)
```

Podemos utilizar un buen número de opciones para controlar el tamaño del widget de texto en la ventana y saber cómo formateará el texto contenido en el área de visualización. Las opciones más utilizadas son la anchura y la altura. Al igual que con el control Entry, recuperamos el texto de un widget de texto utilizando el método **get()**. Tambień podemos quitar el texto desde el widget utilizando el método **delete(),** y añadir texto al widget usando el método **insert()**. Sin embargo, existe una pequeña diferencia con respectos a esos métodos en el widget de texto porque éste trabaja con varias líneas de texto, los valores del índice que se especifican para los métodos get(), delete() e insertar() no constan de un solo valor numérico sino que, en realidad, son dos partes: "**x.y**".

En este caso, **x** es la ubicación de la fila (a partir de 1), e **y** es la ubicación de la columna (a partir de 0). Así que, para hacer referencia al primer carácter en el widget de texto, se utiliza el valor del índice de "1.0". En el siguiente programa se puede observar cómo disponemos de varias líneas en la caja de texto para convertir de minúsculas a mayúsculas.

```
from tkinter import *

class Application(Frame):
      def __init__(self, master):
           super(Application, self).__init__(master)
           self.grid()
           self.create_widgets()
           def create_widgets(self):
           self.label1=Label(self, text='Entrar el texto a
           convertir:')

           self.label1.grid(row=0, column=0, sticky =W)
           self.text1 = Text(self, width=20, height=10)
           self.text1.grid(row=1, column=0)
           self.text1.focus_set()

           self.button1=Button(self,text='Convert',
           command=self.convert)

           self.button1.grid(row=2, column=0)
```

```
            self.button2=Button(self,text='Clear',
            command=self.clear)
            self.button2.grid(row=2, column=1)

    def convert(self):
        varText = self.text1.get("1.0", END)
        varReplaced = varText.upper()
        self.text1.delete("1.0", END)
        self.text1.insert(END, varReplaced)

    def clear(self):
        self.text1.delete("1.0", END)
        self.text1.focus_set()

root = Tk()
root.title = 'Testo de widget text'
root.geometry('300x250')
app = Application(root)
app.mainloop()
```

El resultado de su ejecución, así como el terminal de comandos, se pueden observar en las figuras 9.11 y 9.12:

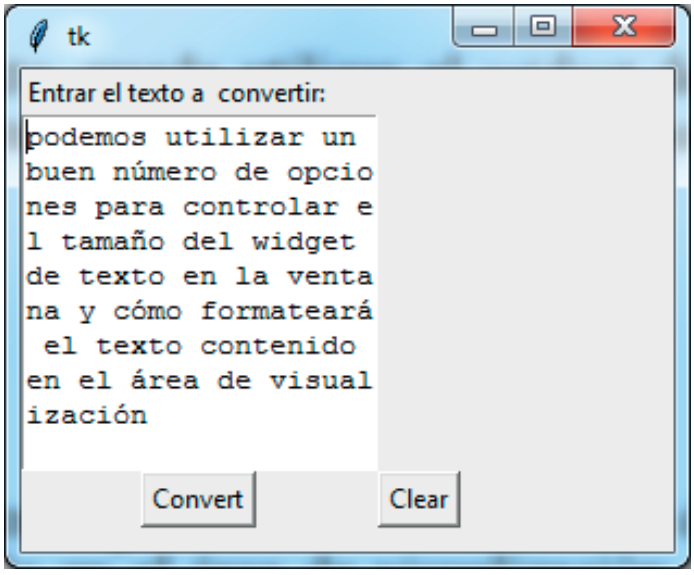


Figura 9.11

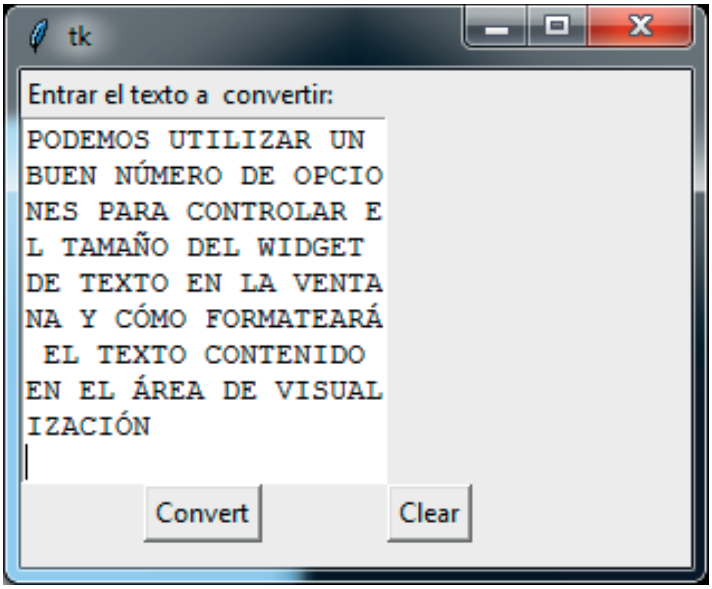


Figura 9.12

El widget **Listbox** (cuadro de lista) proporciona una lista de varios valores a elegir por parte del usuario. Al crear el widget puede especificar cómo el usuario selecciona los elementos de la lista. Para ello se debe usar el parámetro **SelectMode** como se muestra aquí:

```
self.listbox1 = Listbox(self, selectmode=SINGLE)
```

Estas opciones están disponibles para el parámetro **SelectMode**:

- **SINGLE**: Selección de solo un elemento a la vez.
- **BROWSE**: Selección de un solo elemento, pero los elementos se pueden mover en la lista.
- **MÚLTIPLE**: Selección de varios elementos haciendo clic en ellos a la vez.
- **EXTENDED**: Selección de varios elementos utilizando las teclas MAYÚS y CONTROL mientras hacemos clic en los artículos deseados en dicha selección.

Después de crear el widget de cuadro de lista, es necesario agregar elementos a la lista. Lo hacemos mediante el método **insert()** como se muestra aquí:

```
self.listbox1.insert(END, 'element uno')
```

El primer parámetro define la ubicación del índice en la lista de donde se debe insertar el nuevo elemento. Podemos utilizar la palabra clave **END** para colocar el nuevo elemento al final de la lista. Si tenemos una gran cantidad de elementos a añadir al widget, es posible colocarlos en un objeto de la lista y utilizar un bucle **for** para insertar todos a la vez, como en el siguiente ejemplo:

```
elementos = ['elemento uno', 'elemento dos', 'elemento tres']
for elemento in elementos:
            self.listbox1.insert(END, elemento)
```

Para recuperar los elementos seleccionados desde el widget, se necesitan dos pasos. En primer lugar, se utiliza el método **curselection()** para recuperar una tupla que contiene el índice de los elementos seleccionados (a partir de 0).

```
items = self.listbox1.curselection()
```

Una vez que tengamos la tupla que contiene los valores de índice, utilizamos el método **get()** para recuperar el valor del texto del elemento en ese lugar de índice.

```
for elemento in elementos:
      strItem = self.listbox1.get(elemento)
```

En el siguiente programa se expone su uso:

```
from tkinter import *

class Application(Frame):

      def init__(self, master):
            super(Application, self).__init__(master)
            self.grid()
            self.create_widgets()
            def create_widgets(self):
            self.label1 = Label(self, text='Select your
            items')
            self.label1.grid(row=0)
            self.listbox1 = Listbox(self,
            selectmode=EXTENDED)

            elementos=['elemento uno','elemento dos',
            'elemento tres']

            for elemento in elementos:
            self.listbox1.insert(END, elemento)
            self.listbox1.grid(row=1)
            self.button1=Button(self,text='Enviar',
            command=self.display)
            self.button1.grid(row=2)
```

```
            def display(self):

            elementos = self.listbox1.curselection()

            for elemento in elementos:
                  strItem = self.listbox1.get(elemento)
                  print(strItem)
                  print('----------')

root = Tk()
root.title('Testeo Listbox widget ')
root.geometry('300x200')
app = Application(root)
app.mainloop()
```

El resultado de su ejecución, así como el terminal de comandos, se pueden observar en las figuras 9.13 y 9.14.

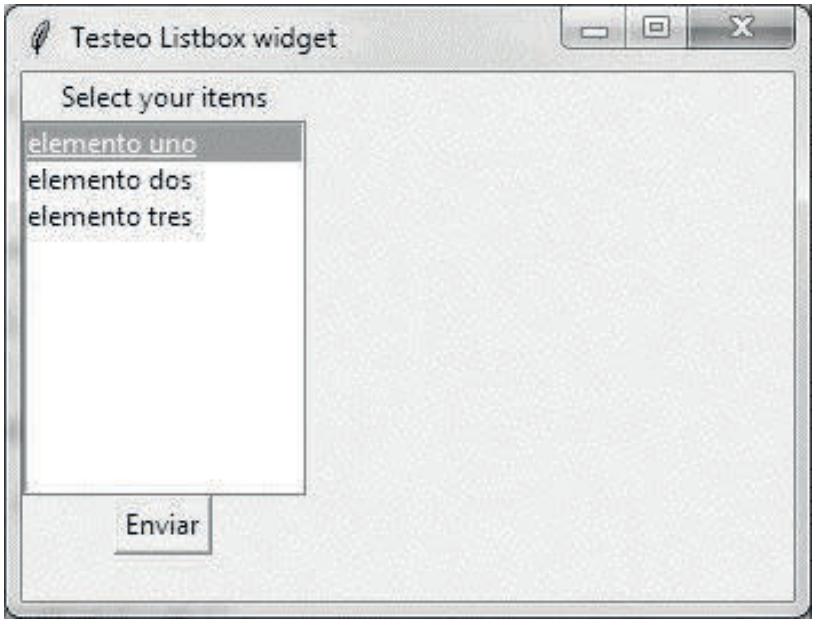


Figura 9.13

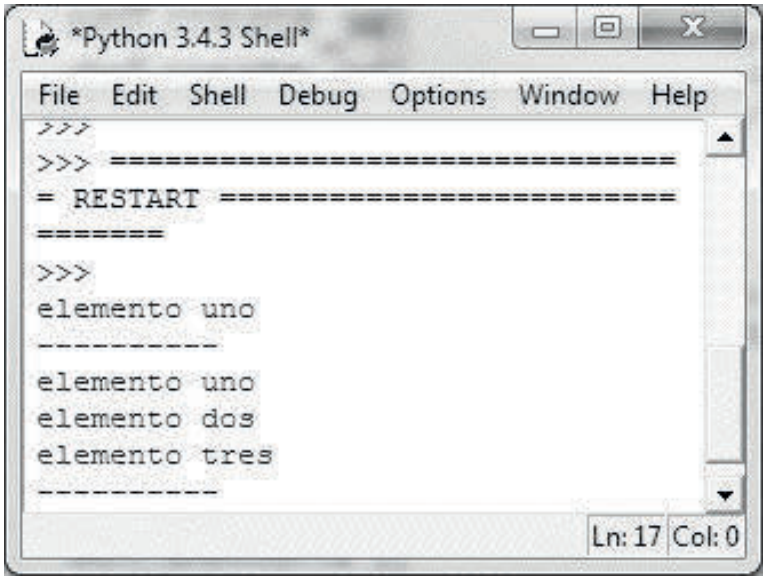


Figura 9.14

Un elemento básico de los programas de interfaz gráfica de usuario es la barra de menús en la parte superior de la ventana. La barra de menús proporciona menús desplegables para que los usuarios del programa puedan hacer rápidamente selecciones. Podemos crear barras de menús en nuestras ventanas Tkinter utilizando el **widget Menu**. Para crear la barra de menú principal se vincula el widget directamente al objeto **Frame**. Luego se utiliza el método **add_command()** para añadir entradas de menú individuales. Cada método add_command () especifica un parámetro **Label** para definir qué texto aparece por la entrada del menú y un parámetro **command** para definir el método a ejecutar cuando se selecciona la entrada del menú. Así es como se ve:

```
menubar = Menu(self)
menubar.add_command(label='Ayuda', command=self.help)
menubar.add_command(label='Salir', command=self.exit)
```

Esta parte de código crea una única barra de menú en la parte superior de la ventana con dos selecciones: **Ayuda** y **Salir**. Finalmente se necesita conectar la barra de menú para el objeto raíz **Tk** añadiendo el siguiente comando:

```
root.config(menu=self.menubar)
```

Ahora, cuando se muestra su aplicación, tendrá una barra de menú en la parte superior, con las entradas del menú que ha definido. Puede crear menús desplegables mediante la elaboración de widgets adicionales de menú y vincularlos a su principal menú de barra de widgets Menú. Eso se ve así:

```
menubar = Menu(self)
filemenu = Menu(menubar)
filemenu.add_command(label='Convertir', command=self.convert)
filemenu.add_command(label='Limpiar', command=self.clear)
menubar.add_cascade(label='Fichero', menu=filemenu)
menubar.add_command(label='Salir', command=root.quit)
root.config(menu=menubar)
```

Después de crear el menú desplegable, se utiliza el método **add_cascade()** para agregarlo al nivel superior de la barra de menú y asignarle una etiqueta.

Ahora que hemos aprendido los widgets más populares de Tkinter, podemos aplicarlo en nuestra Raspberry Pi2 para diseñar nuestros propios programas.

9.5 CONTROL DE UN LED CON TKINTER

Se trata de realizar el control de encendido y apagado de un LED utilizando una interface gráfica que contenga dos botones dentro de una ventana principal. Mediante una pulsación en los botones, accionaremos el encendido o apagado simple del LED conectado al pin GPIO 26 de la Raspberry. Primero inicializamos Tkinter mediante el siguiente trozo de código:

```
from tkinter import *
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(26, GPIO.OUT, pull_up_down=GPIO.PUD_UP, initial=1)
#Variables de Python

debug=True
tkrun=0
tkenable=0
out=0

#Define la ventana de TKinter

root = Tk()
root.wm_title("GPIO Test") # Título de la ventana
root.config(background = "#FFFFF0")
```

Establecemos el pin GPIO 26 como salida, ya que en ahí tenemos conectado el LED. Lo demás hace referencia al tamaño y color de fondo de la ventana.

La siguiente parte del código define las funciones de los botones de encendido y apagado del LED, así como el evento que lo hace conmutar:

```
def onbtnPress():
    if ( debug ): print ("On Botón Presionado")
    GPIO.output(26, 0)

def offbtnPress():
    if ( debug ): print ("Off Botón Presionado")
    GPIO.output(26, 1)

#Definición del frame de los botones

primaryFrame = Frame(root)
primaryFrame.grid(row=0,column=0,sticky=W+N,padx=10,pady=10)
primaryFrame.grid_columnconfigure(0)

#Definiciones de los botones

onBtn=Button(primaryFrame,text="On",command=onbtnPress,
width=10)

onBtn.grid(row=0, column=0, padx=10, pady=2)

offBtn=Button(primaryFrame,text="Off",command=offbtnPress,
width=10)
offBtn.grid(row=1, column=0, padx=10, pady=2)

root.mainloop()
```

La ejecución del programa debe hacerse con los privilegios de root para tener acceso a la Pi2. El resultado se observa en la figura 9.15:

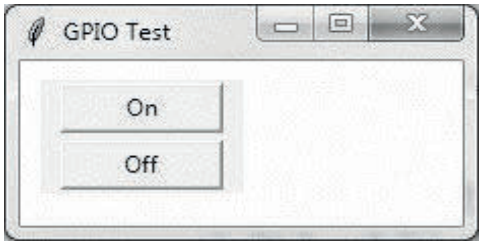


Figura 9.15

9.6 FUNCIÓN MONOESTABLE EN UN LED CON TKINTER

En este caso vamos a encender un LED conectado al mismo pin GPIO 26, pero solo durante un cierto tiempo, trascurrido el cual se volverá a apagar. Añadimos un nuevo botón al código del apartado anterior y utilizamos la función del estado monoestable, tal y como se indica a continuación:

```
def monobtnPress():
    if ( debug ): print("Monoestable botón presionado")
    GPIO.output(26, 0)
    root.after(100, tkmono)

def tkmono():
    GPIO.output(26, 1)

monoBtn=Button(primaryFrame,text="Monoestable",command=monobt
nPress,width=10)
monoBtn.grid(row=0, column=1, padx=10, pady=2)
```

El tiempo de encendido es de 100 ms aunque se puede cambiar desde el código. El resultado se observa en la figura 9.16:

Figura 9.16