

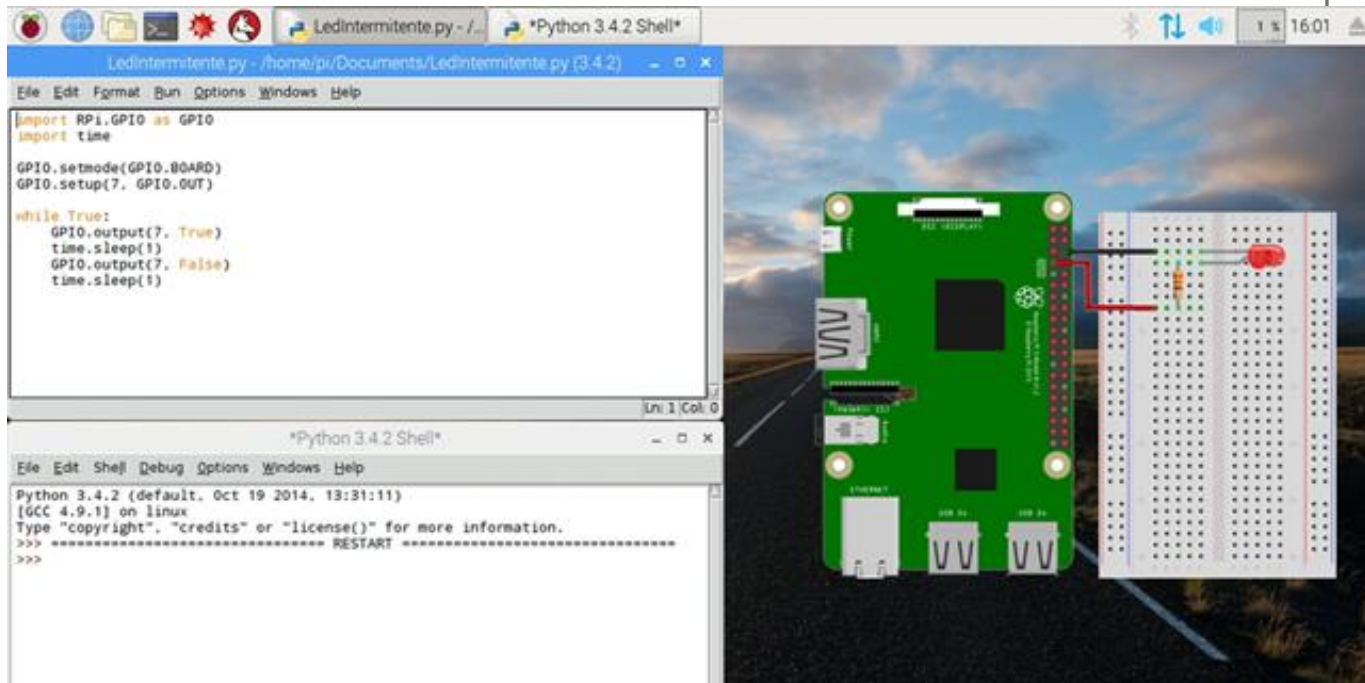
Uso de pines GPIO en Python

Contenido

1. Introduccion	2
2. ¿Qué es GPIO?.....	3
3.ADVERTENCIAS.....	6
4. Primeros programas	7
4.1 Programación del led intermitente en Python.....	7
4.2 Programación del pulsador en Python.....	9
5.Interrupcciones:	11
6 Salida analógica. PWM.:.....	14
6.1 ¿Qué es PWM?	14
6.2 Programación del led intermitente en Python.....	15
7.Programa Integrador:.....	19

1. Introduccion

En esta clase aprenderás qué son y cómo utilizar los pines **GPIO de tu Raspberry Pi** para activar entradas y salidas mediante el lenguaje de programación **Python**. También aprenderás nociones básicas de **electrónica** para realizar numerosas prácticas.



Pines GPIO en Python

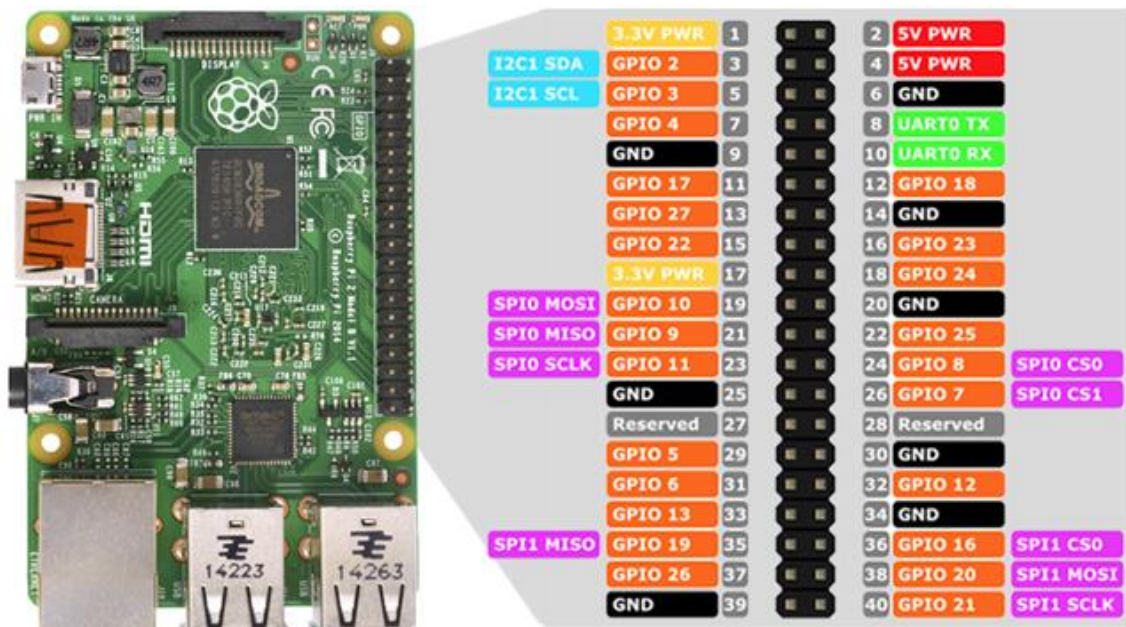
Para instalar el controlador para Python, ejecutaremos los siguientes comandos desde la consola:

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


Como se puede observar, el número de pines pasó de 26 a 40 para tener más disponibilidad, aunque volvemos a comentar que los 26 primeros pines son comunes para todas las versiones. Los pines GPIO tienen funciones específicas (aunque algunos comparten funciones) y se pueden agrupar de la siguiente manera:

- **Amarillo (2):** Alimentación a 3.3V.
- **Rojo (2):** Alimentación a 5V.
- **Naranja (26):** Entradas / salidas de propósito general. Pueden configurarse como entradas o salidas. Ten presente que el nivel alto es de 3.3V y **no son tolerantes a tensiones de 5V.**
- **Gris (2):** Reservados.
- **Negro (8):** Conexión a GND o masa.
- **Azul (2):** Comunicación mediante el protocolo I2C para comunicarse con periféricos que siguen este protocolo.
- **Verde (2):** Destinados a conexión para UART para puerto serie convencional.
- **Morado (5):** Comunicación mediante el protocolo SPI para comunicarse con periféricos que siguen este protocolo.

Todos los pines son de tipo "unbuffered", es decir, no disponen de buffers de protección y puedes dañar la placa con un mal uso.







Pines GPIO para Raspberry Pi 2 Modelo B4

Existen 2 formas de numerar los pines de la Raspberry Pi, en modo GPIO o en modo BCM.

- En el **modo GPIO**, los pines se numeran de forma física por el lugar que ocupan en la placa (representados por el color gris) viene siendo igual para todas las versiones (comenzamos a contar desde arriba a la izquierda y finalizamos abajo a la derecha).
- En el **modo BCM**, los pines se numeran por la correspondencia en el chip Broadcom (que es la CPU de la Raspberry Pi).

Por este mismo motivo puedes encontrar 2 nomenclaturas a la hora de realizar las prácticas con la Raspberry Pi, cuando nos refiramos al modo GPIO o al modo BCM. A continuación mostramos una tabla de equivalencias.

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

De los pines GPIO disponibles, hay una serie de pines con capacidad de PWM (como veremos más adelante). Sin embargo no se dispone de ningún convertidor de analógico-digital. Esto quiere decir que para medir valores de sensores analógicos necesitaremos utilizar un convertidor externo o un Arduino en la mayoría de los casos.



3.ADVERTENCIAS

Cuando se utilizan los pines de *GPIO* hay que poner mucho cuidado para no dañar la propia Raspberry Pi. Es muy importante comprobar los niveles de tensión y la corriente solicitada. Los pines de *GPIO* pueden generar y consumir tensiones compatibles con los circuitos de 3.3V, como la nueva electrónica. No conectar nunca componentes de 5V, o podes quemar el chip y quedarnos sin Raspberry Pi.

Los pines GPIO ofrecen una tensión de 3.3V y no son tolerantes a tensiones de 5V.

Hay que tener presente que la intensidad de corriente que sale de esos pines proviene de la **fuentes de 3.3V** y esta fue diseñada para una carga de unos **3mA por cada pin GPIO**, suficiente para encender diodos led, pero poco más.

Los pines GPIO ofrecen una corriente de 3mA.

4. Primeros programas

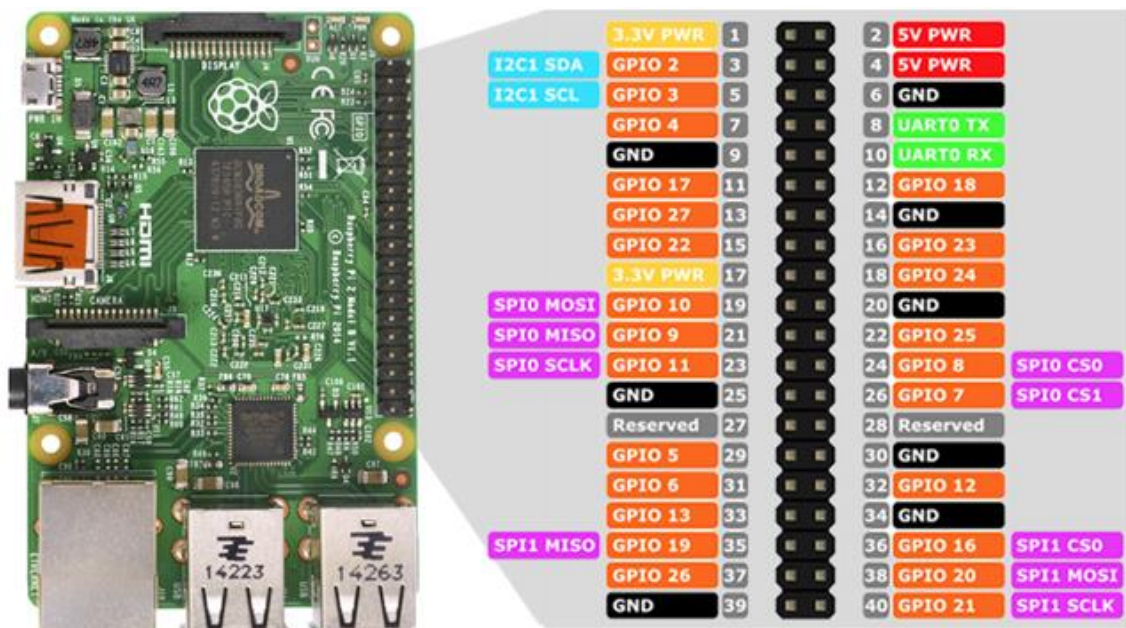
4.1 Programación del led intermitente en Python

En este primer ejemplo se va a programar y construir un led que parpadea continuamente con una frecuencia de 1 segundo. Para la programación, debemos abrir el entorno de programación Python 3 (IDLE) de nuestra Raspberry Pi y a continuación crearemos un nuevo fichero para escribir el código de programación.

La primera línea que vamos a escribir corresponde a la importación de la librería de los pines GPIO, es decir, con esta librería podemos utilizar las funciones implementadas para controlar los pines de nuestra Raspberry Pi.

import RPi.GPIO as GPIO

A continuación indicamos el modo con el cual nos vamos a dirigir al pin a utilizar ya que podemos dirigirnos al pin por el número en la placa (*GPIO.BOARD*), o por el canal al cual está conectado en el Chip Broadcom (*GPIO.BCM*), como puedes ver en la siguiente imagen.



Pines GPIO para Raspberry Pi 2 Modelo B

En nuestro caso, vamos a utilizar la primera opción, es decir, indicaremos a Python que vamos a utilizar el pin situado en su posición de la placa, 7 en este caso. La siguiente instrucción será la inicialización del pin, es decir, los pines pueden ser utilizados como entradas o salidas. Por último, solamente nos quedará activar (True) o desactivar (False) el pin seleccionado. Para ello, se va a situar el código dentro de un bucle infinito (recuerda la indentación del código).

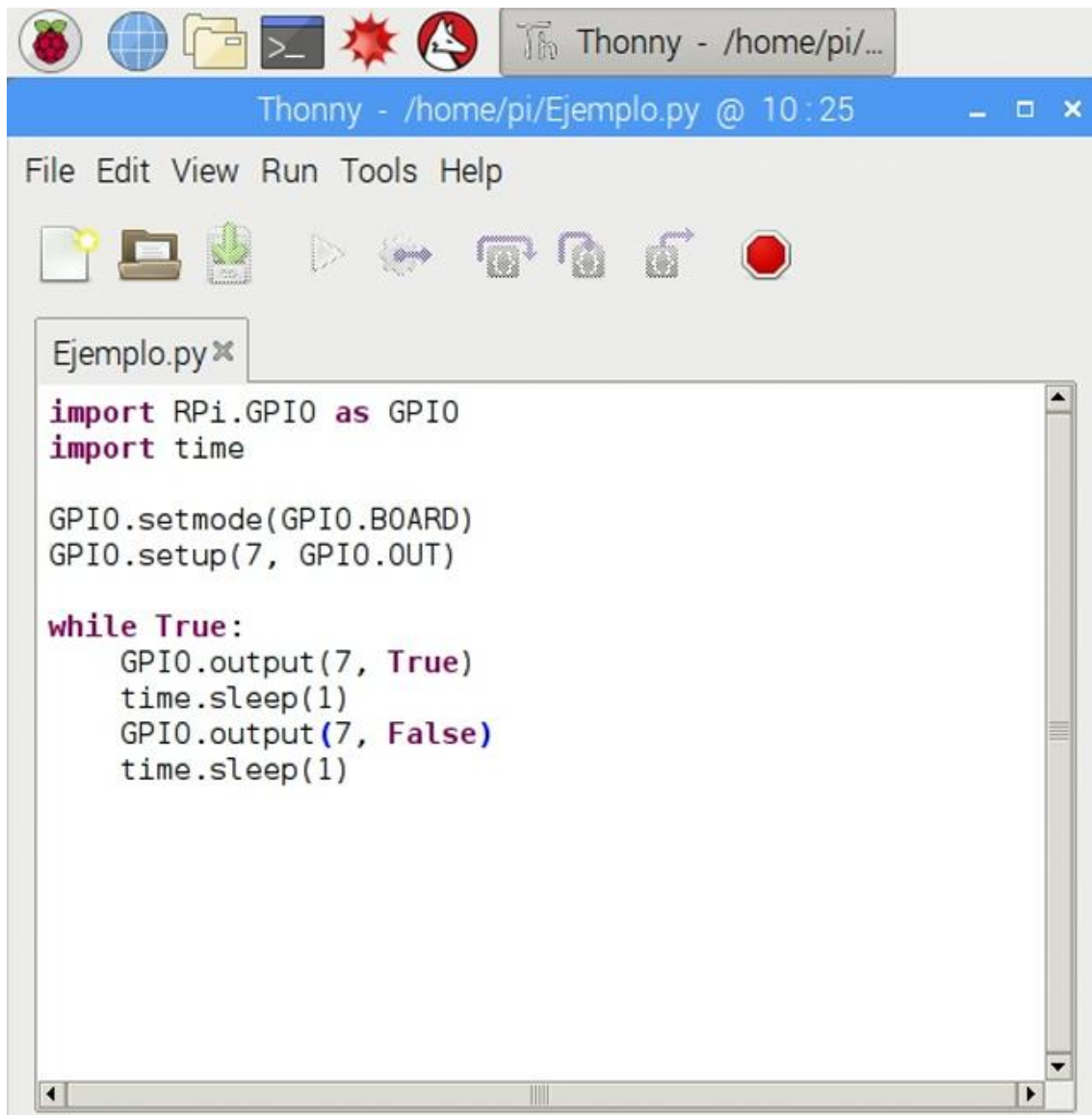
```
import RPi.GPIO as GPIO    #Importamos libreria de pines GPIO
import time                #Importamos la función de Tiempo
```

```
GPIO.setmode(GPIO.BOARD)  #Configuracion de pines en modo GPIO
```

GPIO.setup(7, GPIO.OUT) **#Pin 7 GPIO4 configurado como salida**

while True: **#loop del programa principal**
 GPIO.output(7, True) **#pongo en alto el pin**
 time.sleep(1) **#espero 1 milisegundo**
 GPIO.output(7, False) **#pongo en bajo el pin 7**
 time.sleep(1) **#delay1 milisegundo**

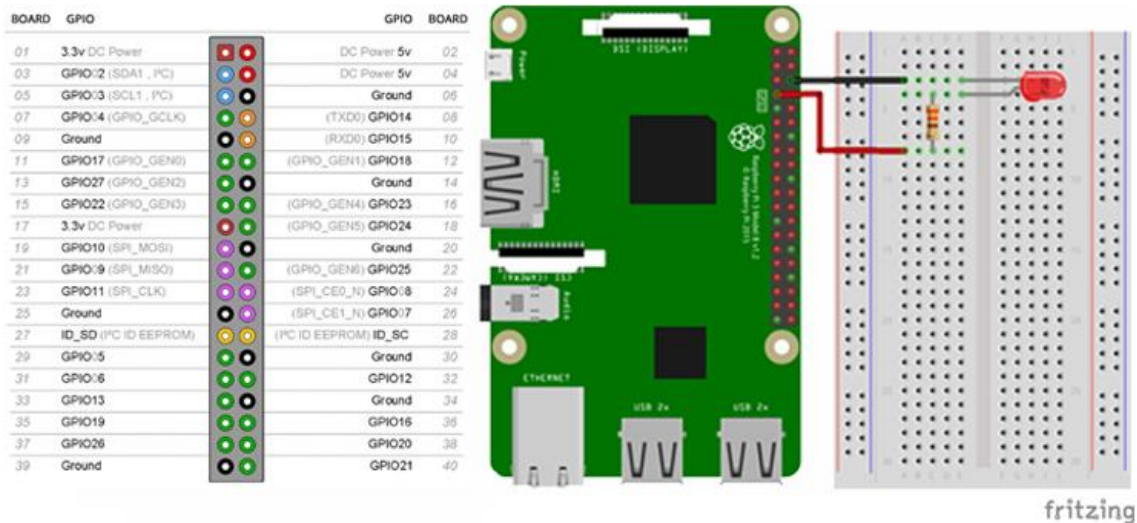
En la siguiente imagen puedes ver el resultado del código en Python para encender y apagar un led conectado al pin número 7 de nuestra Raspberry Pi con una frecuencia de 1 segundo.



Intermitente en Python y GPIO

Esquema eléctrico

Por último, fijándonos en la tabla de los pines GPIO de nuestra Raspberry Pi, conectamos al cátodo de nuestro led el pin de masa (GND) y al ánodo del led el pin que hemos activado en la programación con Scratch, el pin GPIO 4 (pin número 7 de nuestra placa).



Esquema eléctrico de Raspberry Pi 2 B para el intermitente

4.2 Programación del pulsador en Python

En esta ejemplo se va a programar un pulsador para encender o apagar un led en nuestra Raspberry Pi. Para ello realizaremos el ejercicio de forma similar que el primer ejemplo, salvo que le añadiremos un interruptor.

El interruptor lo declaramos de tipo entrada y en la condición se puede observar que al pulsar sobre el botón se encenderá el led situado en la posición 7 de nuestro pin GPIO y en caso contrario se apagará.

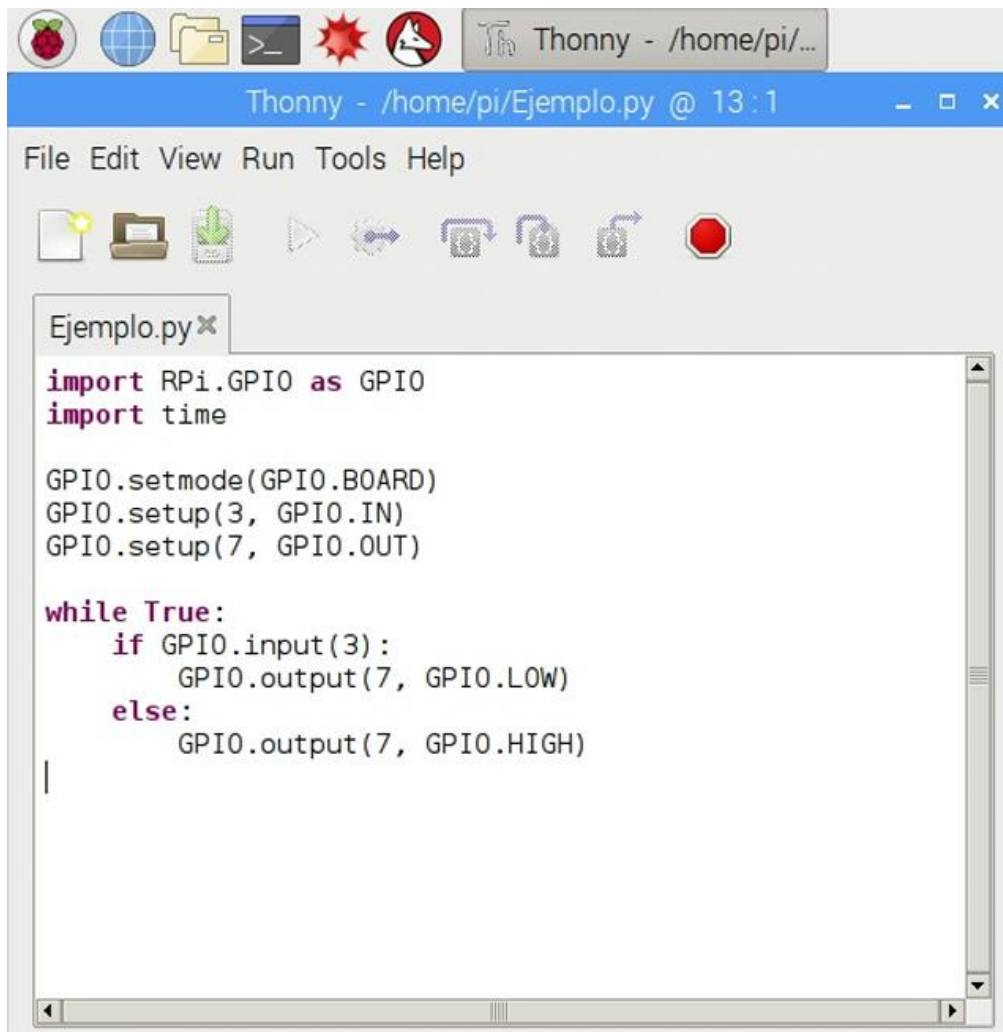
```
import RPi.GPIO as GPIO      #importo la librerías de GPIO
import time                  #importo funciones de tiempo

GPIO.setmode(GPIO.BOARD)    #uso los nombre de pines como su numeracion
GPIO.setup(3, GPIO.IN)       #Inicializo el pin 3 como entrada
GPIO.setup(7, GPIO.OUT)      #inicializo el pin 7 como salida

while True:                  #Bucle principal del programa
    if GPIO.input(3):         #si el if es verdadero gpio3 en alto
        GPIO.output(7, False) #apaga en pin 7
    else:                     #caso contrario se prende el pin 7
        GPIO.output(7, True)
```

- Fíjate que si pusiera una instrucción fuera de ese bucle, nunca se ejecutaría.

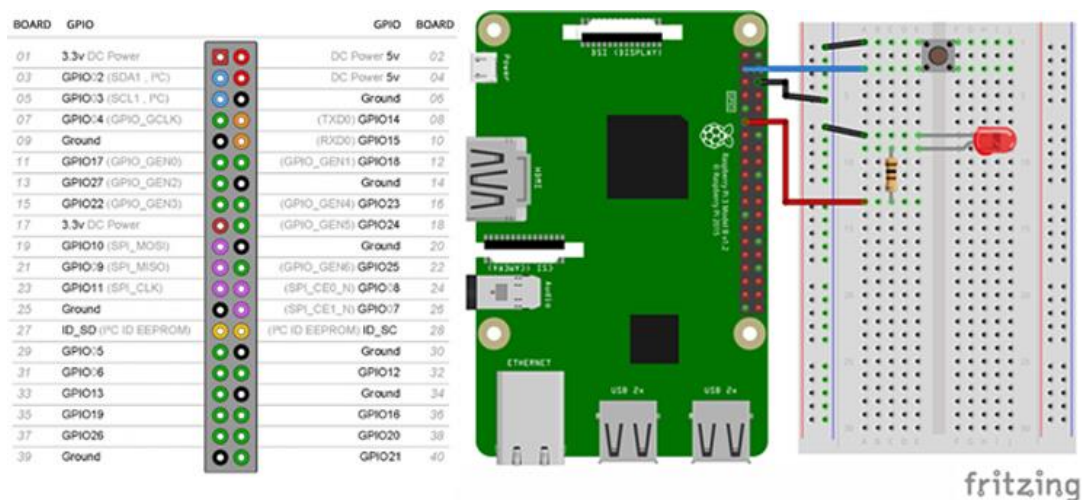
En la siguiente imagen puedes ver cómo se ha programado el código en el editor de Thonny Python para Raspberry Pi



Pulsador en Python y GPIO

Esquema eléctrico

Por último, procedemos a montar el pulsador sobre la protoboard y los conectamos a los pines GPIO de nuestra Raspberry Pi como los hemos programado. En este caso vamos a utilizar el pin físico número 3 que equivale al GPIO 2.



Esquema eléctrico del pulsador en Raspberry Pi

5.Interrupciones:

Si queremos detectar un cambio de estado en esta entrada, el método que hemos usado hasta ahora es el de emplear las entradas digitales para consultar repetidamente el valor de la entrada, con un intervalo de tiempo (delay) entre consultas.

Este mecanismo se denomina “poll”, y tiene 3 claras desventajas.

- Suponer un continuo consumo de procesador y de energía, al tener que preguntar continuamente por el estado de la entrada.
- Si la acción necesita ser atendida inmediatamente, por ejemplo en una alerta de colisión, esperar hasta el punto de programa donde se realiza la consulta puede ser inaceptable.
- Si el pulso es muy corto, o si el procesador está ocupado haciendo otra tarea mientras se produce, es posible que nos saltemos el disparo y nunca lleguemos a verlo.

Para resolver este tipo de problemas, los microprocesadores incorporan el concepto de **interrupción**, que es un mecanismo que permite asociar una función a la ocurrencia de un determinado evento. Esta función de callback asociada se denomina ISR (Interruption Service Rutine).

En programación, una interrupción es una señal recibida por el procesador o MCU, para indicarle que debe «interrumpir» el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.

Una interrupción es una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción, la cual, por lo general, no forma parte del programa. Una vez finalizada dicha subrutina, se reanuda la ejecución del programa. Las interrupciones HW son generadas por los dispositivos periféricos habilitando una señal del CPU (llamada IRQ del inglés “interrupt request”) para solicitar atención del mismo.

Todos los dispositivos que deseen comunicarse con el procesador por medio de interrupciones deben tener asignada una línea única capaz de avisar al CPU cuando le requiere para realizar una operación. Esta línea se denomina IRQ.

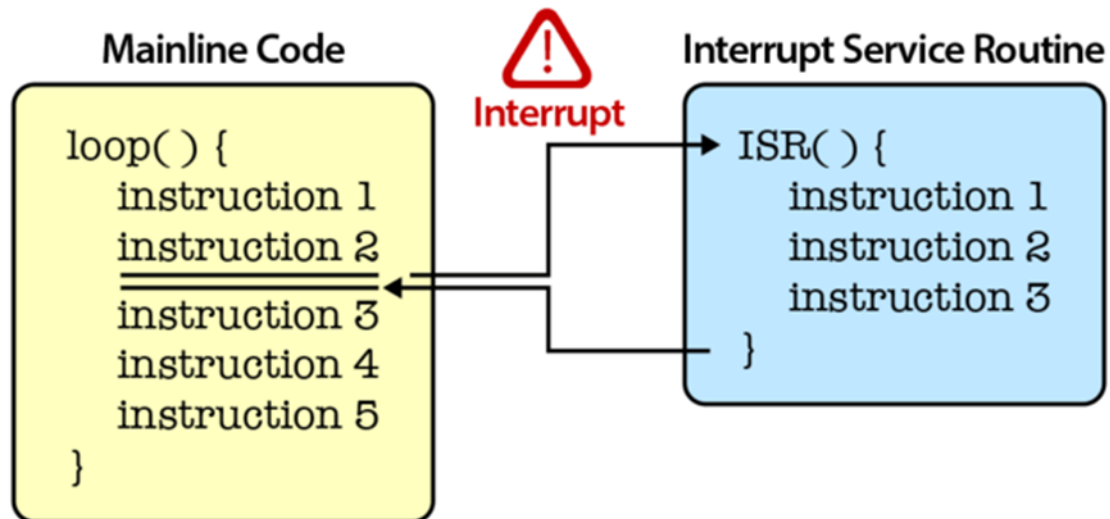
Las IRQ son líneas que llegan al **controlador de interrupciones**, un componente de hardware dedicado a la gestión de las interrupciones, y que está integrado en la MCU.

El **controlador de interrupciones** debe ser capaz de habilitar o inhibir las líneas de interrupción y establecer prioridades entre las mismas. Cuando varias líneas de petición de interrupción se activan a la vez, el controlador de interrupciones utilizará estas prioridades para escoger la interrupción sobre la que informará al procesador principal. También puede darse el caso de que una rutina de tratamiento de interrupción sea interrumpida para realizar otra rutina de tratamiento de una interrupción de mayor prioridad a la que se estaba ejecutando.

Procesamiento de una Interrupción:

1. Terminar la ejecución de la instrucción máquina en curso.
2. Salvar el estado del procesador (valores de registros y flags) y el valor del contador de programa en la pila, de manera que en la CPU, al terminar el proceso de interrupción, pueda seguir ejecutando el programa a partir de la última instrucción.

3. La CPU salta a la dirección donde está almacenada la rutina de servicio de interrupción (Interrupt Service Routine, o abreviado ISR) y ejecuta esa rutina que tiene como objetivo atender al dispositivo que generó la interrupción.
4. Una vez que la rutina de la interrupción termina, el procesador restaura el estado que había guardado en la pila en el paso 2 y retorna al programa que se estaba usando anteriormente.



Tipos de Interrupciones:

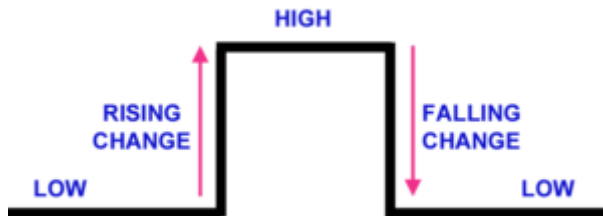
- Interrupciones HW o externas: Estas son asíncronas a la ejecución del procesador, es decir, se pueden producir en cualquier momento independientemente de lo que esté haciendo el CPU en ese momento. Las causas que las producen son externas al procesador y a menudo suelen estar ligadas con los distintos dispositivos de entrada o salida.
- Interrupciones SW: Las interrupciones por software son aquellas generadas por un programa en ejecución. Para generarlas, existen distintas instrucciones en el código máquina que permiten al programador producir una interrupción.
- Un evento programado o Timer. Son las interrupciones asociadas a los timers y gracias a ellas funciona millis().
- Excepciones: Son aquellas que se producen de forma síncrona a la ejecución del procesador típicamente causada por una condición de error en un programa. Normalmente son causadas al realizarse operaciones no permitidas tales como la división entre 0, el desbordamiento, el acceso a una posición de memoria no permitida, etc.

Ponemos añadir interrupciones en nuestra raspberry y olvidarnos de bucles de lectura.

Las interrupciones son de tres tipos:

- **Raising** (Activación del pin: 0 -> 1);

- **Falling** (Desactivación del pin: 1 -> 0);
- **Both** (por ambos cambios anteriores).



La implementación puede hacerse de tres modos:

- **Por espera.** El programa se queda en este punto hasta que suceda el evento:

Ejemplo:

```
canal = GPIO . wait_for_edge ( canal , GPIO_RISING , tiempo de
espera = 5000 )
if canal == 0 :
    imprimir ( 'Se ha agotado el tiempo de espera' )
else :
    imprimir ( 'Borde detectado en el canal' , canal )
```

- **Por Flag.** La ponemos la detección de cambio en las entradas al principio. Cuando hay un cambio, el programa sigue ejecutándose y en algún punto de nuestra aplicación le preguntamos si ha habido un cambio:

Ejemplo:

```
GPIO . add_event_detect ( canal , GPIO . RISING )    # agregar
detección de flanco ascendente en un canal
do_something ()
if GPIO . event_detected ( canal ) :
    print( 'Botón presionado' )
```

- **Por interrupción de programa** :Cuando se produzca un cambio en el pin, se detendrá el programa y se ejecutará la función indicada:

```
def my_callback ( canal ) :
    print ( '¡Esta es una función de devolución de llamada de
evento de borde!' )
    print ( 'Borde detectado en el canal % s ' % canal )
    print ( 'Esto se ejecuta en un hilo diferente al de tu programa
principal' )

GPIO . add_event_detect ( canal , GPIO . RISING , callback =
my_callback )    # agregar detección de flanco ascendente en un
canal
... el resto de su programa ...
```

Si desea más de una función de devolución de llamada:

```
def my_callback_one ( canal ) :
    print ( 'Devolución de llamada uno' )

def my_callback_two ( canal ) :
    print ( 'Devolución de llamada dos' )

GPIO . add_event_detect ( canal , GPIO . RISING )
GPIO . add_event_callback ( canal , my_callback_one )
GPIO . add_event_callback ( canal , my_callback_
```

6 Salida analógica. PWM.:

Hasta ahora hemos realizado prácticas con **salidas y entradas digitales**. Sin embargo, en ocasiones necesitamos señales analógicas. Es decir, en las lecciones anteriores comprobábamos si habíamos pulsado un pulsador (entrada digital) y en ese caso encendíamos un led (salida digital).

En las siguientes prácticas vamos a necesitar utilizar **salidas analógicas**. Puesto que en Raspberry Pi no disponemos de convertidores digital a analógico nos vemos obligados a utilizar otras técnicas para poder simular las señales analógicas en Raspberry Pi. A esta técnica se le conoce como modulación por ancho de pulsos o PWM.

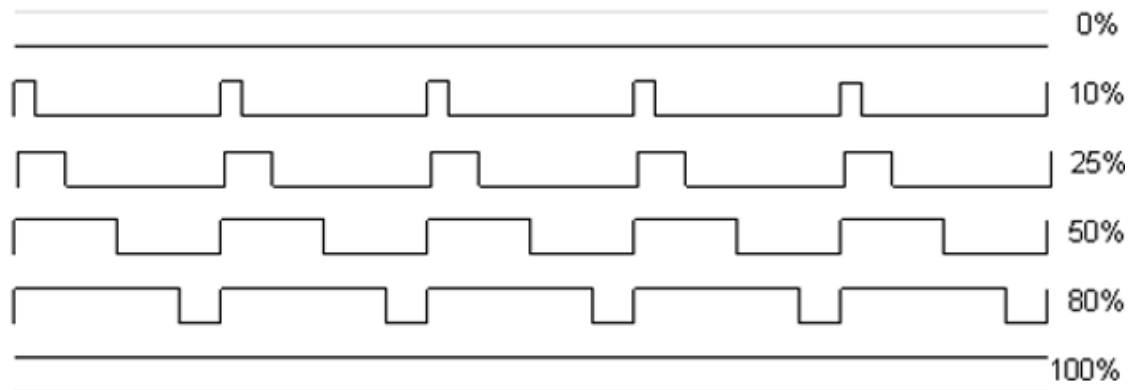
6.1 ¿Qué es PWM?

Pulse Width Modulation (PWM) es una técnica que consiste en la variación del *duty cycle* de una señal digital periódica, fundamentalmente con dos posibles objetivos:

- Por un lado se puede utilizar como mecanismo para transmitir información. Por ejemplo, los servo-motores tienen una entrada digital por la que se transmite el ángulo deseado codificado en PWM.
- Por otro lado se puede utilizar para regular la cantidad de potencia suministrada a la carga. Por ejemplo, las luminarias LED frecuentemente utilizan reguladores PWM para permitir el control de intensidad.

La Raspberry Pi tiene una varias patas de GPIO (GPIO12, GPIO13, GPIO18 y GPIO19) que puede configurarse como salida de alguno de los dos canales PWM. El propio BCM2835 se encarga de gestionar la generación de la señal, liberando completamente al procesador principal.

El periférico PWM de la Raspberry Pi es muy flexible pero solo dispone de dos canales (*PWM0* y *PWM1*). Puede funcionar en modo PWM o en modo serializador. En el modo serializador simplemente saca por la pata correspondiente los bits de las palabras que se escriben en un *buffer*. Veamos primero el modo PWM



PWM en Raspberry Pi

El usuario puede configurar dos valores:

- Un *rango* de valores disponible (hasta 1024).
- Un *valor* que determina el *duty cycle*. El módulo PWM se encarga de mantener el *duty cycle* en la relación valor/rango.

La frecuencia base para PWM en Raspberry Pi es de 19.2Mhz. Esta frecuencia puede ser dividida mediante el uso de un divisor indicado con `pwmSetClock`, hasta un máximo de 4095. A esta frecuencia funciona el algoritmo interno que genera la secuencia de pulsos, pero en el caso del BCM2835 se dispone de dos modos de funcionamiento, un modo equilibrado (*balanced*) en el que es difícil controlar la anchura de los pulsos, pero permite un control PWM de muy alta frecuencia, y un modo *mark and space* que es mucho más intuitivo y más apropiado para controlar servos. El modo *balanced* es apropiado para controlar la potencia suministrada a la carga.

En el modo *mark and space* el módulo PWM incrementará un contador interno hasta llegar a un límite configurable, el rango de PWM, que puede ser de como máximo 1024. Al comienzo del ciclo el pin se pondrá a 1 lógico, y se mantendrá hasta que el contador interno llegue al *valor* puesto por el usuario. En ese momento el pin se pondrá a 0 lógico y se mantendrá hasta el final del ciclo.

6.2 Programación del led intermitente en Python

En este ejemplo se va a programar y variar la intensidad de un led por medio de la técnica PWM. Para ello vamos a proceder a realizar una programación y montaje sobre la protoboard similar a la práctica del led intermitente.

En este caso, vamos a crear una variable declarada como objeto PWM la cual tendrá una frecuencia de 100 (imagina que el ciclo se divide en 100 partes iguales). A continuación,

indicamos el comienzo del ciclo del led en 0, y dentro de un bucle incrementamos el valor de 25 unidades para ver cómo se enciende progresivamente.

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

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7, GPIO.OUT)
```

```
led = GPIO.PWM(7, 100)      #Para crear una instancia de PWM (Canal,frecuencia)
```

```
while True:
```

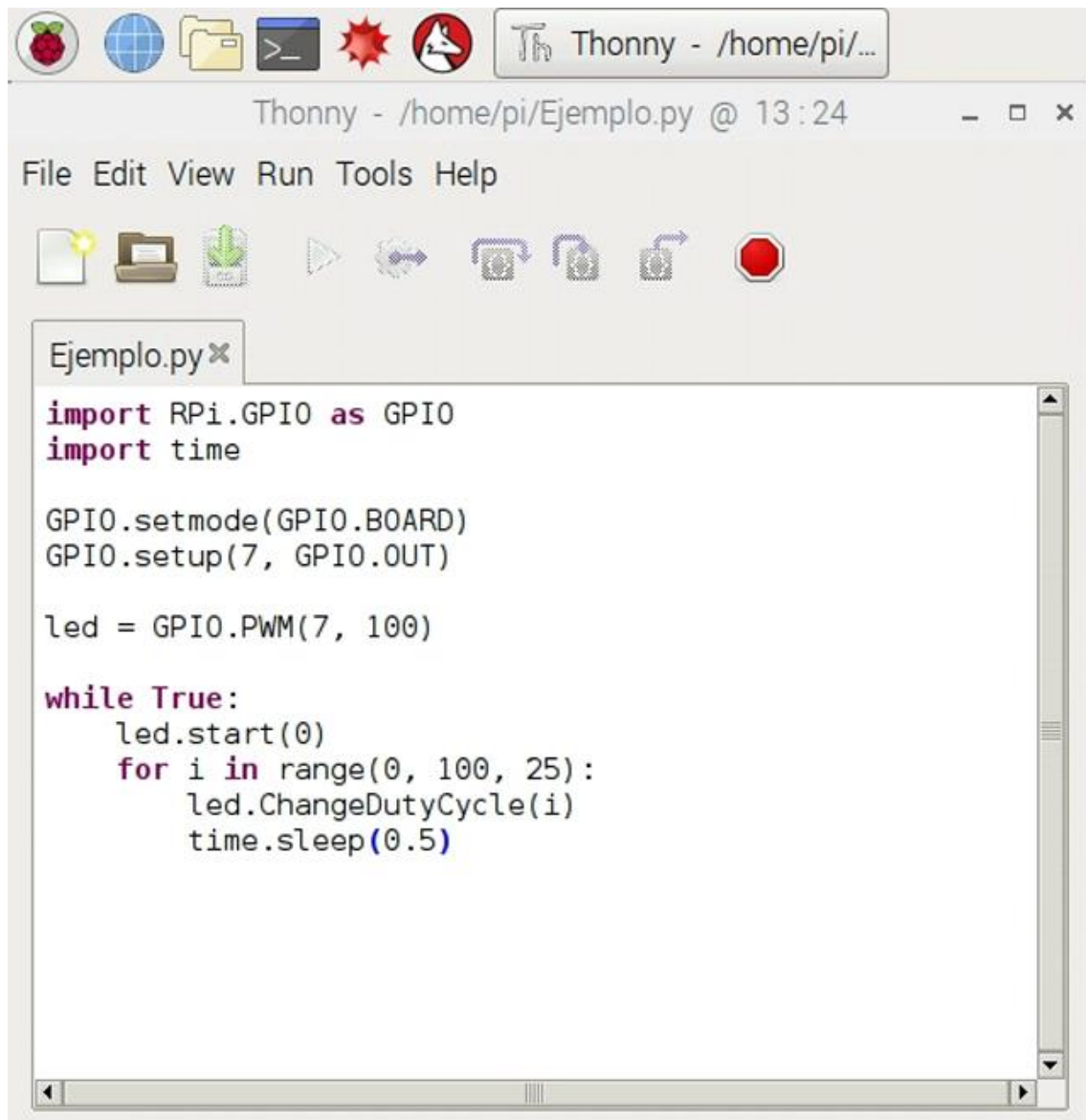
```
    led.start(0)             #inicilizar valor en dc del PWM
```

```
    for i in range(0, 100, 25):
```

```
        led.ChangeDutyCycle(i)    # Para cambiar el ciclo de trabajo de 0 a 100
```

```
        time.sleep(0.5)
```

En la siguiente imagen puedes ver cómo se ha programado el código en el editor de Thonny Python para Raspberry Pi



```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
GPIO.setup(7, GPIO.OUT)

led = GPIO.PWM(7, 100)

while True:
    led.start(0)
    for i in range(0, 100, 25):
        led.ChangeDutyCycle(i)
        time.sleep(0.5)
```

Led PWM en Python y GPIO

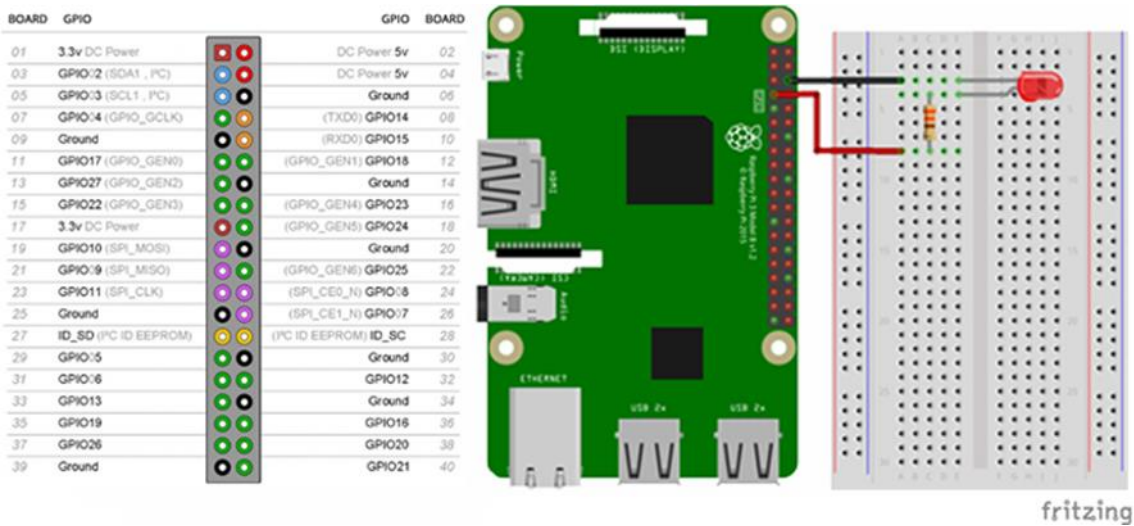
Cálculos

En este caso utilizamos 1 led con el cual el fabricante nos asegura que funcionan a una tensión o voltaje de 2,1V y admiten una corriente máxima de 20mA. De estos datos tenemos que:

- El Voltaje o diferencia de potencial en el led será de $3,3V - 2,1V = 1,2V$.
- La Intensidad en el led será de **20mA** (la misma que en el circuito ya que no varía).
- La Resistencia que hay que aplicarle según la Ley de Ohm será de: $R = V / I = 1,2V / 20mA = 60\Omega$, redondeando a **100Ω**.

Esquema eléctrico

Por último, procedemos a construir el circuito sobre la placa de prototipado y los conectamos a los pines GPIO de nuestra Raspberry Pi como los hemos programado.



Esquema eléctrico del led PWM en Raspberry Pi

7.Programa Integrador:

```
#led rojo escribe led amarillo es el espacio y cambiode letra
from conversion import Palabras #uso esta bliblioteca como funcion
import RPi.GPIO as GPIO

pul=40 #pin 22 es el pulsador
interrup = 0 #variable global para la interrupccion

GPIO.setmode(GPIO.BOARD)

#entrada digital con pull up
GPIO.setup(pul,GPIO.IN,pull_up_down= GPIO.PUD_UP)#entrada con resistecia de pull-up
pulsador en alto
```

#Callback llamdas de funcion interrupccion

```
def Interrupcion(channel1):
    global interrup
    interrup=1
    os.system("clear")
    print("se interrumpio externamente")
```

#interrupciones

```
GPIO.add_event_detect(pul,GPIO.FALLING,callback=Interrupcion,bouncetime=300)
```

#programa principal

```
while(interrup = 0):
    cad=input("Introduce una cadena de texto: ")

    for i in cad:
        a=Palabras(i) #le voy a enviar una frase por teclado y me la repondera en morse

    print(a)

GPIO.cleanup()
exit()
```

Uso de biblioteca propia

#creo bibliotecas del letras en morse

```
import RPi.GPIO as GPIO #libreria de los pines de la raspberry i en minuscula
import time
```

#configurar pines

```
led_enc=35
led_esp=37
GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setup(led_enc,GPIO.OUT) #el que escribe.
GPIO.setup(led_esp,GPIO.OUT)#el que letra letra y espacio()
```

```
class LED:
```

```
    def __init__(self,l1,l2):
```

```
        self.led1=l1
```

```
        self.led2=l2
```

```
    def Led_punto(self):
```

```
        GPIO.output(self.led1,True) #enciendo pata 7 pin 4
```

```
        time.sleep(1) #espero 1 segunbdo = .
```

```
        GPIO.output(self.led1,False) #apago pata 7 GPIO 4
```

```
    def Led_raya(self):
```

```
        GPIO.output(self.led1,True) #enciendo lines 3 segundos
```

```
        time.sleep(3)
```

```
        GPIO.output(self.led1,False)
```

```
    def Led_espacio(self):
```

```
        GPIO.output(self.led2,True) #enciendo pata 7 pin 4
```

```
        time.sleep(1) #espero 1 segunbdo = .
```

```
        GPIO.output(self.led2,False) #apago pata 7 GPIO
```

#para agregar mas perfericos agregamos mas pbjetos

```
led=LED(led_enc,led_esp)
```

```
def Palabras(n):
```

```
    if (n == 'A' or n == 'a'):          #si es A o a prendo el bucle  
                                         #letra A=-
```

```
        led.Led_punto()
```

```
        time.sleep(1)
```

```
        led.Led_raya()
```



```
time.sleep(1)
led.Led_espacio()

print(n)
```

```
elif (n == 'B' or n == 'b'):
```

```
#si es B o b prendo el bucle
#b=...
```

```
led.Led_raya() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
```

```
print(n)
```

```
led.Led_espacio()
```

```
elif (n == 'C' or n == 'c'):
```

```
#si es C o c prendo el bucle
#c=-.-.
```

```
led.Led_raya() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_raya()
time.sleep(1)
led.Led_punto()
```

```
print(n)
```

```
led.Led_espacio()
```

```
elif (n == 'D' or n == 'd'):
```

```
#si es D o d prendo el bucle
#d=-..
```

```
led.Led_raya() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
```

```
print(n)
```

```

        led.Led_espacio()

elif (n == 'E' or n == 'e'):
    #si es B o b prendo el bucle
    #e=.

    led.Led_punto()
    time.sleep(1)

    print(n)

    led.Led_espacio()

elif (n == 'F' or n == 'f'):
    #si es B o b prendo el bucle
    #f=.-.
    led.Led_punto() #-
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_punto()

    print(n)

    led.Led_espacio()

elif (n == 'G' or n == 'g'):
    #si es B o b prendo el bucle
    #g=---.
    led.Led_raya() #-
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_punto()

    print(n)

    led.Led_espacio()

elif (n == 'H' or n == 'h'):
    #si es H o h prendo el bucle
    #b=....
    led.Led_punto() #-
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)

```

```

        led.Led_punto()
        time.sleep(1)
        led.Led_punto()

    print(n)

    led.Led_espacio()

elif (n == 'I' or n == 'i'):
    #si es B o b prendo el bucle
    #i=..
    led.Led_punto()
    time.sleep(1)
    led.Led_punto()

    print(n)

    led.Led_espacio()

elif (n == 'J' or n == 'j'):
    #si es B o b prendo el bucle
    #j=---
    led.Led_punto() #-
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_raya()

    print(n)

    led.Led_espacio()

elif (n == 'K' or n == 'k'):
    #si es B o b prendo el bucle
    #k=-.-
    led.Led_raya() #-
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)
    led.Led_raya()

    print(n)

    led.Led_espacio()

elif (n == 'L' or n == 'l'):

```

```

#si es B o b prendo el bucle
    #l=-..
    led.Led_punto()
    time.sleep(1)
    led.Led_raya() #-
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)

    print(n)

    led.Led_espacio()
elif (n == 'M' or n == 'm'):
#si es B o b prendo el bucle
    #m=--
    led.Led_raya() #-
    time.sleep(1)
    led.Led_raya()

    print(n)

    led.Led_espacio()

elif (n == 'N' or n == 'n'):
#si es B o b prendo el bucle
    #n=-.
    led.Led_raya() #-
    time.sleep(1)
    led.Led_punto()

    print(n)

    led.Led_espacio()

elif (n == 'O' or n == 'o'):
#si es B o b prendo el bucle
    #b=-...
    led.Led_raya() #-
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_raya()

    print(n)

    led.Led_espacio()

```



```

elif (n == 'P' or n == 'p'):
#si es B o b prendo el bucle
    #p=.-.
    led.Led_punto() #-
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_punto()

    print(n)

    led.Led_espacio()

```

```

elif (n == 'Q' or n == 'q'):
#si es B o b prendo el bucle
    #b=--.-
    led.Led_raya() #-
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)
    led.Led_raya()

    print(n)

    led.Led_espacio()

```

```

elif (n == 'R' or n == 'r'):
#si es B o b prendo el bucle
    #r=.-.
    led.Led_punto() #-
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_punto()

    print(n)

    led.Led_espacio()

```

```

elif (n == 'S' or n == 's'):
#si es B o b prendo el bucle
    #s=...
    led.Led_punto()

```

```

time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()

print(n)

led.Led_espacio()

elif (n == 'T' or n == 't'):
#si es B o b prendo el bucle
    #t=-
    led.Led_raya() #-

print(n)

led.Led_espacio()

elif (n == 'U' or n == 'u'):
#si es B o b prendo el bucle
    #u=..-
    led.Led_punto()
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)
    led.Led_raya()

print(n)

led.Led_espacio()

elif (n == 'V' or n == 'v'):
#si es B o b prendo el bucle
    #v=...-
    led.Led_punto() #-
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)
    led.Led_punto()
    time.sleep(1)
    led.Led_raya()

print(n)

led.Led_espacio()

elif (n == 'X' or n == 'x'):
#si es B o b prendo el bucle

```

```

#x=-.-
led.Led_raya() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_raya()

print(n)

led.Led_espacio()

elif (n == 'Y' or n == 'y'):
#si es B o b prendo el bucle
#y=-.--
led.Led_raya() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_raya()
time.sleep(1)
led.Led_raya()

print(n)

led.Led_espacio()

elif (n == 'Z' or n == 'z'):
#si es B o b prendo el bucle
#z=---.
led.Led_raya() #-
time.sleep(1)
led.Led_raya()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()

print(n)

led.Led_espacio()

elif (n == '1'):
#si es B o b prendo el bucle
#1=-.---
led.Led_punto() #-
time.sleep(1)

```

```

led.Led_ Raya()
time.sleep(1)
led.Led_ Raya()
time.sleep(1)
led.Led_ Raya()
time.sleep(1)
led.Led_ Raya()

print(n)

led.Led_espacio()

elif (n == '2'):
#si es B o b prendo el bucle
#2=..---
led.Led_punto() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_ Raya()
time.sleep(1)
led.Led_ Raya()
time.sleep(1)
led.Led_ Raya()

print(n)

led.Led_espacio()

elif (n == '3'):
#si es B o b prendo el bucle
#3=...--
led.Led_punto() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_ Raya()
time.sleep(1)
led.Led_ Raya()

print(n)

led.Led_espacio()

elif (n == '4'):
#si es B o b prendo el bucle

```

```
#4=....-
led.Led_punto() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_raya()
print(n)
```

```
led.Led_espacio()
```

```
elif (n == '5'):
#si es B o b prendo el bucle
```

```
#5=.....
led.Led_punto() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()

print(n)
```

```
led.Led_espacio()
```

```
elif (n == '6'):
#si es B o b prendo el bucle
```

```
#b=-....
led.Led_raya() #-
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()
time.sleep(1)
led.Led_punto()

print(n)
```

```
led.Led_espacio()
```

```
elif (n == '7'):
```

```
#si es B o b prendo el bucle
```

```
#7=--...
```

```
led.Led_raya() #-
```

```
time.sleep(1)
```

```
led.Led_raya()
```

```
time.sleep(1)
```

```
led.Led_punto()
```

```
time.sleep(1)
```

```
led.Led_punto()
```

```
time.sleep(1)
```

```
led.Led_punto()
```

```
print(n)
```

```
led.Led_espacio()
```

```
elif (n == '8'):
```

```
#si es B o b prendo el bucle
```

```
#8=---..
```

```
led.Led_raya() #-
```

```
time.sleep(1)
```

```
led.Led_raya()
```

```
time.sleep(1)
```

```
led.Led_raya()
```

```
time.sleep(1)
```

```
led.Led_punto()
```

```
time.sleep(1)
```

```
led.Led_punto()
```

```
print(n)
```

```
led.Led_espacio()
```

```
elif (n == '9'):
```

```
#si es B o b prendo el bucle
```

```
#9=----.
```

```
led.Led_raya() #-
```

```
time.sleep(1)
```

```
led.Led_raya()
```

```
time.sleep(1)
```

```
led.Led_raya()
```

```
time.sleep(1)
```

```
led.Led_raya()
```

```
time.sleep(1)
```

```
led.Led_punto()
```

```
print(n)
```



```

        led.Led_espacio()

elif (n == '0'):
    #si es B o b prendo el bucle
    #0=-----
    led.Led_raya() #-
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_raya()
    time.sleep(1)
    led.Led_raya()

    print(n)

    led.Led_espacio()

```

```

elif (n == ' '):
    #si es B o b prendo el bucle

    led.Led_espacio()
    print(n)

    led.Led_espacio()

```

```

else:
    print('Error mal ingresado o puntucacio()n')
    GPIO.output(led_enc,True)
    GPIO.output(led_esp,True)
    time.sleep(5)
    GPIO.output(led_enc,False)
    GPIO.output(led_esp,False)
    return 'Error mal ingresado'

```

```

#tareparaterminRCREARUNOBEJTOLLAMADO LED DONDE PUEDA REEMPLAZR EL PUNRO
RAYA
#28/8 termine prim,er parte agregar mas perifericos

return 'Termine bien' #fin del bucle

```

Esquemático

