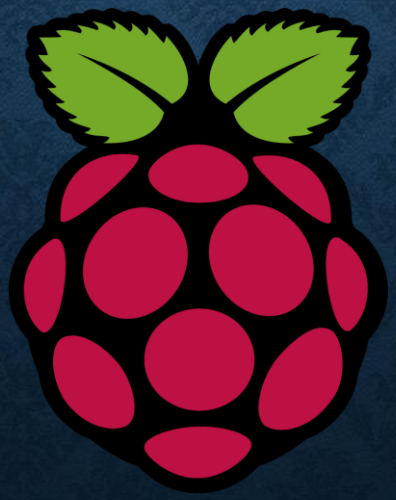
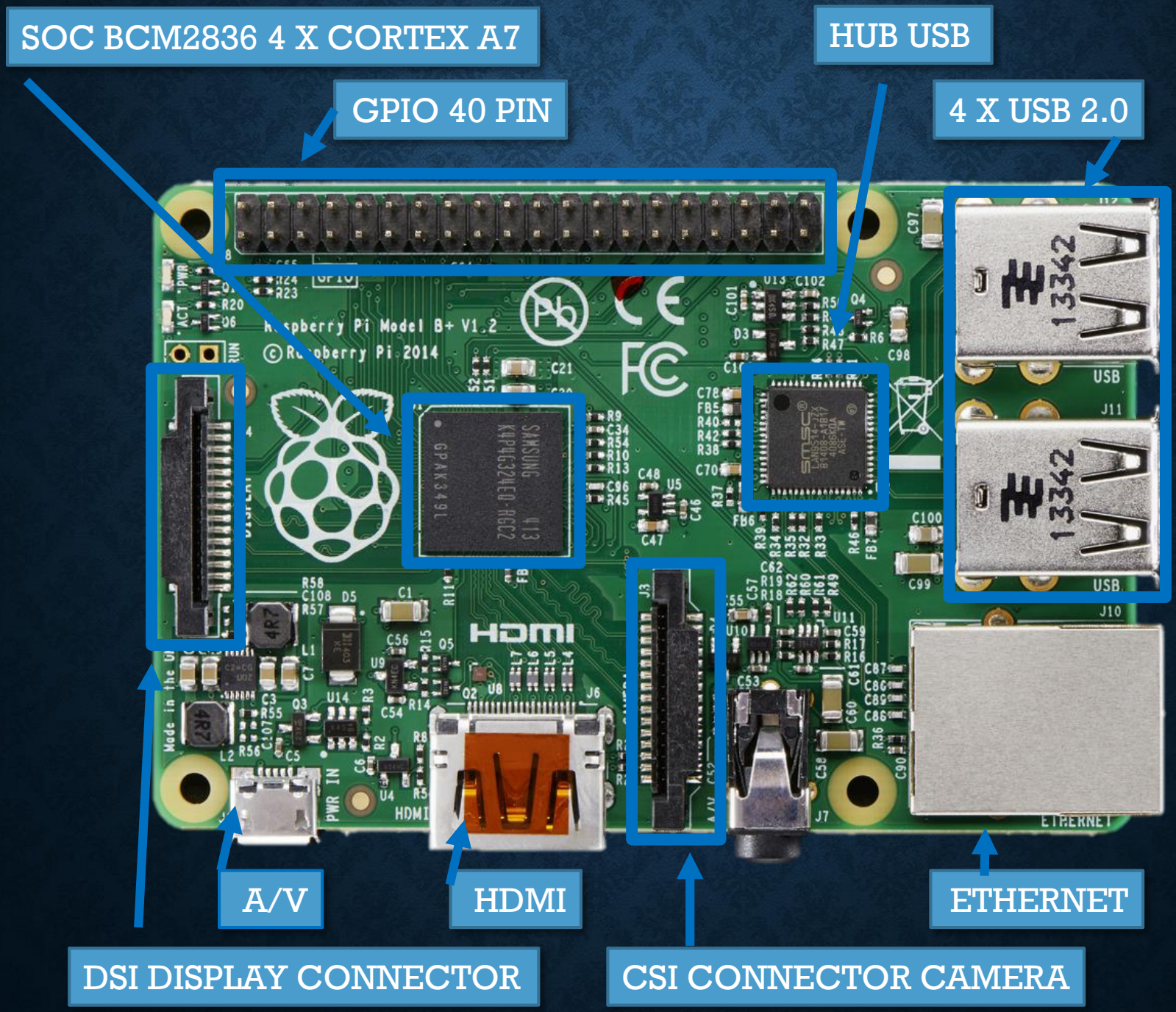
RASPGIPO

Python para Raspberry



Product	Recommended PSU current capacity	Maximum total USB peripheral current draw	Typical bare-board active current consumption
Raspberry Pi Model A	700mA	500mA	200mA
Raspberry Pi Model B	1.2A	500mA	500mA
Raspberry Pi Model A+	700mA	500mA	180mA
Raspberry Pi Model B+	1.8A	600mA/1.2A (switchable)	330mA
Raspberry Pi 2 Model B	1.8A	600mA/1.2A (switchable)	


SOC BCM2836 4 X CORTEX A7
HUB USB
GPIO 40 PIN
4 X USB 2.0
A/V
HDMI
ETHERNET
DSI DISPLAY CONNECTOR
CSI CONNECTOR CAMERA

Raspberry Pi2 GPIO Header

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

RPI.GPIO

Outputs

Inputs

PWM

Comunicación

RPI.GPIO

Existen dos modos para acceder a los pines

```
import RPi.GPIO as GPIO

#Numeracion de pines de la tarjeta
GPIO.setmode(GPIO.BOARD)
```

```
import RPi.GPIO as GPIO

#Numeracion de pines del BCM2836
GPIO.setmode(GPIO.BCM)
```

RPI.GPIO

Configuración básica de un pin como entrada o salida

```
#Configura channel como entrada
GPIO.setup(channel, GPIO.IN)
```

```
#Configura channel como salida
GPIO.setup(channel, GPIO.OUT)
```

RPI.GPIO

Al terminar de usar los pines deben ser liberados

```
#Libera todos los pines utilizados
GPIO.cleanup()
```

```
#Libera una tupla de pines
GPIO.cleanup((channel1, channel2))
```

```
#Libera una lista de pines
GPIO.cleanup([channel1, channel2])
```

OUTPUTS

- La salidas tienen dos valores HIGH y LOW

HIGH pone el pin 12 a 3.3v

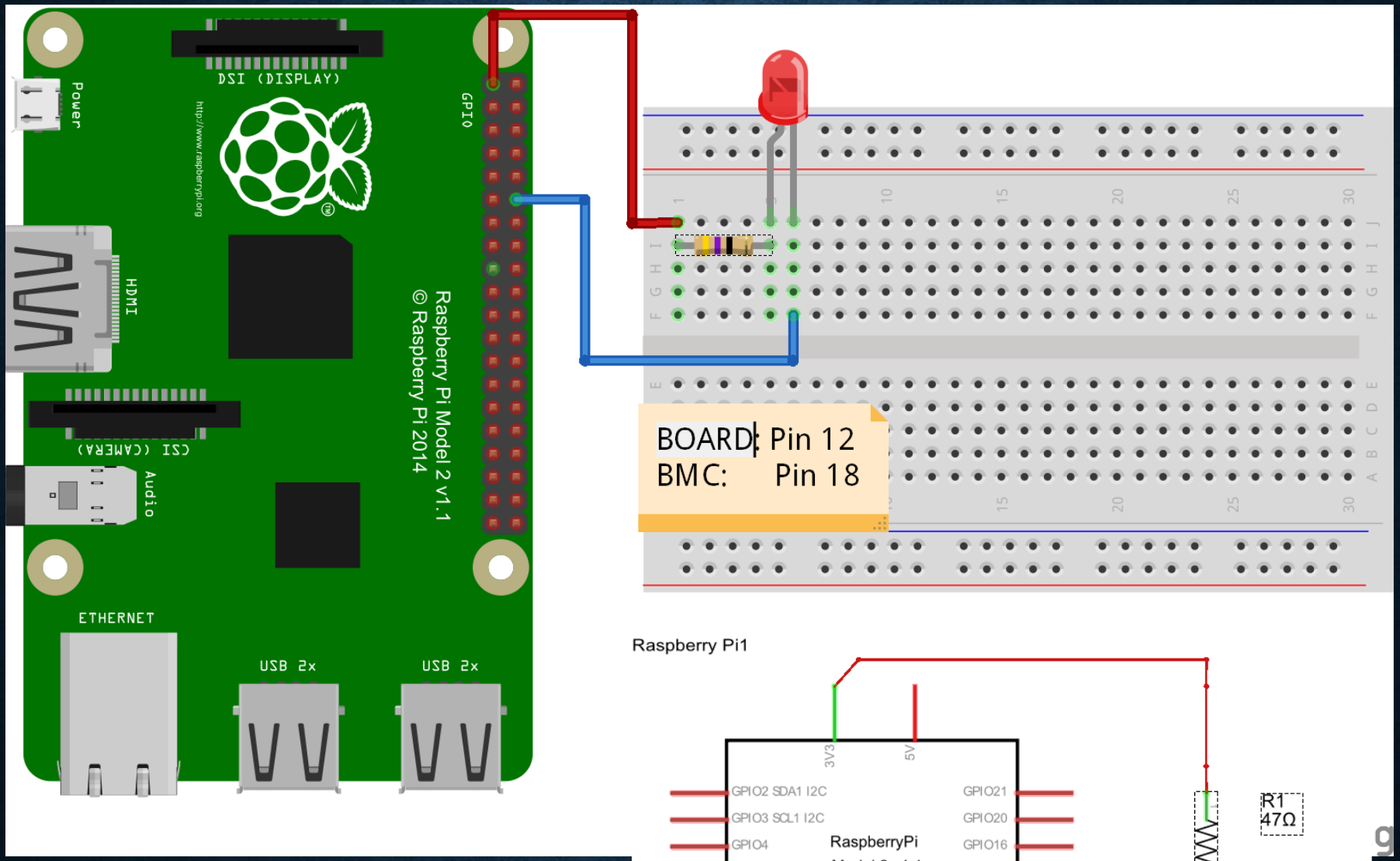
```
#Define una salida en uno logico
GPIO.output(12, GPIO.HIGH)
 # o
GPIO.output(12, 1)
 # o
GPIO.output(12, True)
```

HIGH pone el pin 12 a GND

```
#Define una salida en cero logico
GPIO.output(12, GPIO.LOW)
 # o
GPIO.output(12, 0)
 # o
GPIO.output(12, False)
```

BLINK

```
1  import RPi.GPIO as GPIO
2  import time
3
4
5  #Variables
6  led=12       #asignamos el pin 12 a la variable led
7  speed=1    #tiempo de espera en segundos
8
9  #Numeracion de pines de la tarjeta
10 GPIO.setmode(GPIO.BOARD)
11
12 #Configuramos pin 12 como salida
13 GPIO.setup(led, GPIO.OUT, initial=GPIO.HIGH)
14
15 try:
16     while True:
17         time.sleep(speed)
18         GPIO.output(led, GPIO.LOW)
19         time.sleep(speed)
20         GPIO.output(led, GPIO.HIGH)
21         print ("hola")
22 except KeyboardInterrupt:
23     GPIO.cleanup()
```


DSI (DISPLAY)
Power
GPIO
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
Raspberry Pi Model 2 v1.1
© Raspberry Pi 2014
BOARD: Pin 12
BMC: Pin 18

Raspberry Pi1
3V3
5V
GPIO2 SDA1 I2C
GPIO3 SCL1 I2C
GPIO4
GPIO17
GPIO27
GPIO22
GIPO10 SPI0_MOSI
GPIO9 SPI0_MISO
GPIO11 SPI0_SCLK
ID_SD I2C ID EEPROM
GPIO5
RaspberryPi
Model 2 v1.1
GPIO21
GPIO20
GPIO16
GPIO12
ID_SC I2C ID EEPROM
GPIO7 SPI0_CE1_N
GPIO8 SPI0_CE0_N
GPIO25
GPIO24
GPIO23
GPIO18 PCM_CLK
R1
47Ω
LED1
Red (633nm)

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


def blink(led, speed, initial, state):
    actual=time.time()
    if(actual-initial>speed):
        state= not state
        GPIO.output(led, state)
        initial=time.time()
    return (initial, state)

#Variables
led=12      #asignamos el pin 12 a la variable led
speed=1   #tiempo de espera en segundos

#Numeracion de pines de la tarjeta
GPIO.setmode(GPIO.BOARD)

#Configuramos pin 12 como salida
GPIO.setup(led, GPIO.OUT, initial=GPIO.HIGH)

inicial=time.time()
state=1

try:
    while True:
        inicial, state = blink(led,speed,inicial,state)
        print ("hola")
except KeyboardInterrupt:
    GPIO.cleanup()
```

INPUTS

- Pueden ser configuradas con pull up o pull down (10K resistor)

```
#Define una entrada en PULL UP
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#Define una entrada en PULL DOWN
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

INTERRUPCIONES

wait_for_edge()

event_detected()

threaded
callback

WAIT_FOR_EDGE()

GPIO.wait_for_edge(pin , edge, timeout)

Edge:

- GPIO.RISING
- GPIO.FALLING
- GPIO.BOTH.

```
# Espera por 5 segundos para un franco de subida
channel = GPIO.wait_for_edge(channel, GPIO_RISING, timeout=5000)
if channel is None:
    print('Timeout occurred')
else:
    print('Edge detected on channel', channel)
```

EVENT_DETECTED()

La función event_detected sirve para que el programa escuche en todo momento si existe un edge

```
#Crea un evento para saber si existe un franco
GPIO.add_event_detect(channel, GPIO.RISING)
do_something()
if GPIO.event_detected(channel):
    print('Button pressed')
```

THREADED CALLBACKS

- Funciones que son llamadas en segundo plano.

```
#Llamando un callback
def my_callback(channel):
    print('This is a edge event callback function!')

GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback)
```

THREADED CALLBACKS

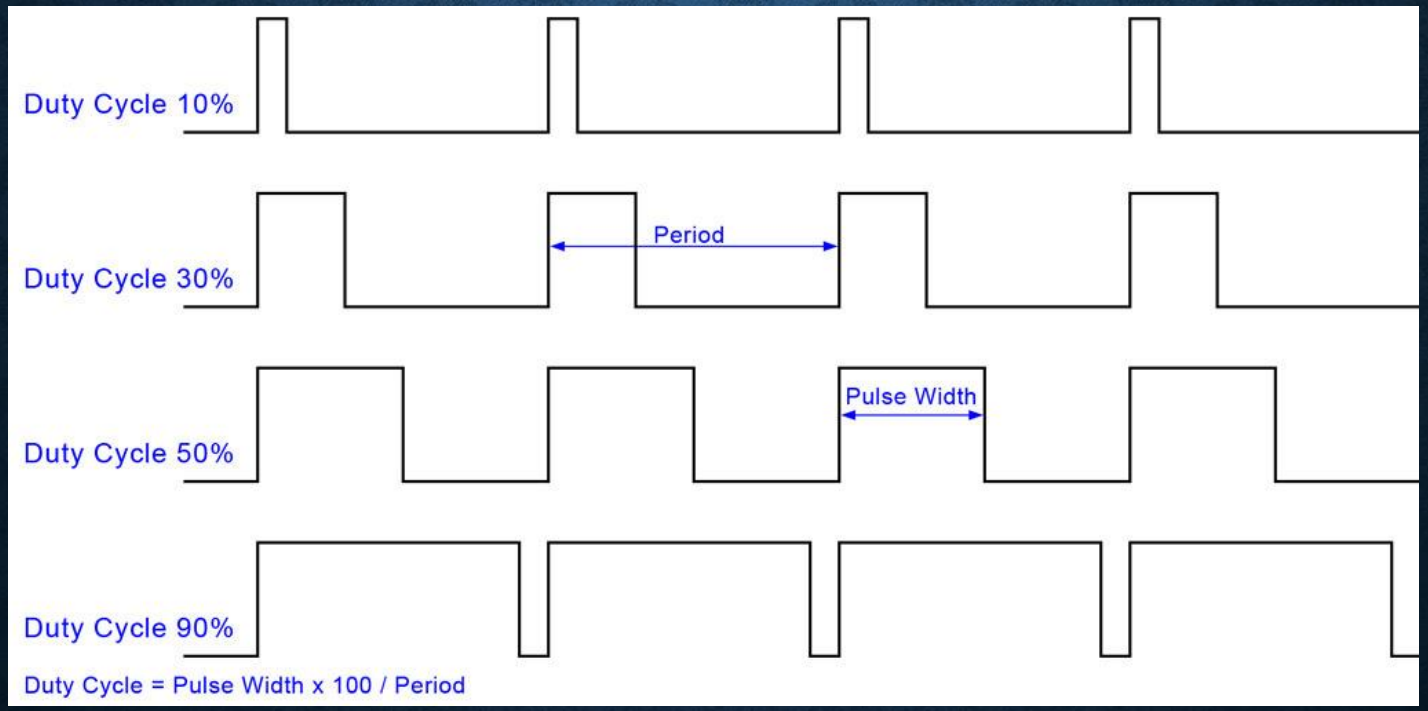
- Es posible tener más de un callback por evento.

```
#Llamando mas de un callback
def my_callback_one(channel):
    print('Callback one')

def my_callback_two(channel):
    print('Callback two')

GPIO.add_event_detect(channel, GPIO.RISING)
GPIO.add_event_callback(channel, my_callback_one)
GPIO.add_event_callback(channel, my_callback_two)
```

PWM


Duty Cycle 10%
Duty Cycle 30%
Period
Duty Cycle 50%
Pulse Width
Duty Cycle 90%
Duty Cycle = Pulse Width x 100 / Period

PWM

- Para crear un PWM necesitamos el puerto de salida y la frecuencia.

```
#Creamos un PWM
p = GPIO.PWM(led, fc)
```

- Para iniciar el PWM es necesario usar el ciclo de trabajo con un valor entre 0 y 100.

```
#Iniciamos el PWM
p.start(dc)
```

PWM

- Es posible cambiar la frecuencia y el ciclo de trabajo de un PWM.

```
#Cambia la frecuencia
p.ChangeFrequency(freq)
```

```
#Cambia el ciclo de trabajo
p.ChangeDutyCycle(dc)
```

- Siempre debe detenerse un PWM después de ser usado.

```
#Detenemos el PWM
p.stop()
```