

# Taller personalizado de Programación y Robótica en CMM BML 24\_25 – CL12

[R] **Neopixel** -: Manejo Básico

[R] **Neopixel** 2da parte: Manejo  
avanzado (bucles for)

**21 Mayo 2025**



Voluntario : J.C. Santamaria

# Clase

## 2425\_12 –

### Índice

(75 minutos)

---

**1. [Robótica] Neopixel Manejo básico**

---

**2. [R y P] Neopixel Manejo avanzado**  
bucles **for**: usos + complejos creando  
funciones

---

**3. [R] Neopixel + Wifi: Leer cheerlight**



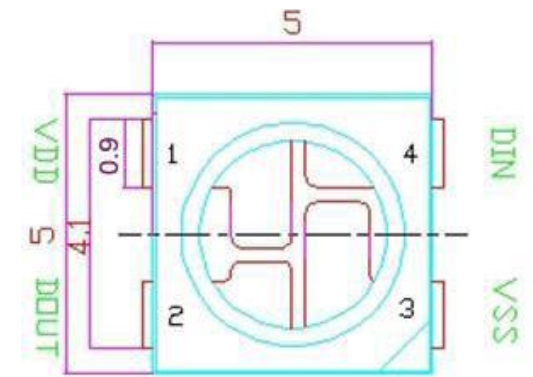
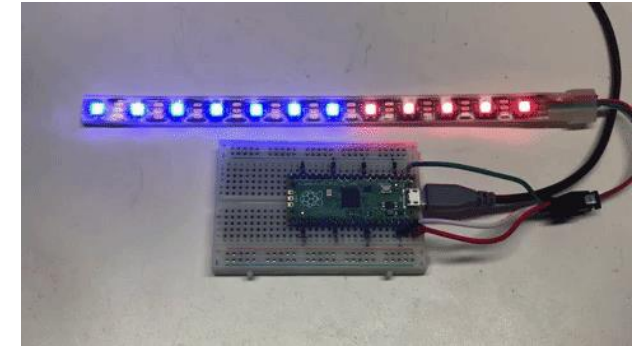
# Clase 2425\_12.1 [R] Neopixel ¿Qué son y para que se usan?

## Seguiremos fuente # 3 - MicroPython for Kids

<https://www.coderdojotc.org/micropython/basics/05-neopixel/>

Los **neopixeles** son

- Dispositivos de Salida,
- Tipo led que combinan los 3 colores básicos Red Green Blue, con intensidades programables en cada color = 8bits x3 => 16 millones de colores
- Se pueden conectar “encadenados” y se puede programar el color de cada pieza de la cadena
- Solo requieren 1 hilo para su control + 2 para alimentación
  - El hilo de control “entra” y “sale” de cada neopixel
- El mas común es el compuesto por unidades de **WS2812B**
- Usaremos la librería integrada en MicroPython: desde 1.18 hay soporte integrado para NeoPixels para el microcontrolador Raspberry Pi Pico y Pico W (RP2040)
- Alimentación de módulos a 5volt con uControlador a 3.3volt -> [Ver Uso Basico](#)
- Cómo mezclar lógicas de 5volt y de 3.3 volt



NO.	Symbol	Function description
1	VDD	Power supply LED
2	DOUT	Control data signal output
3	VSS	Ground
4	DIN	Control data signal input

# Clase 2425\_12.1[R] Neopixel - Displays manejo con 2 “pizarras” Valido para la mayoría de displays

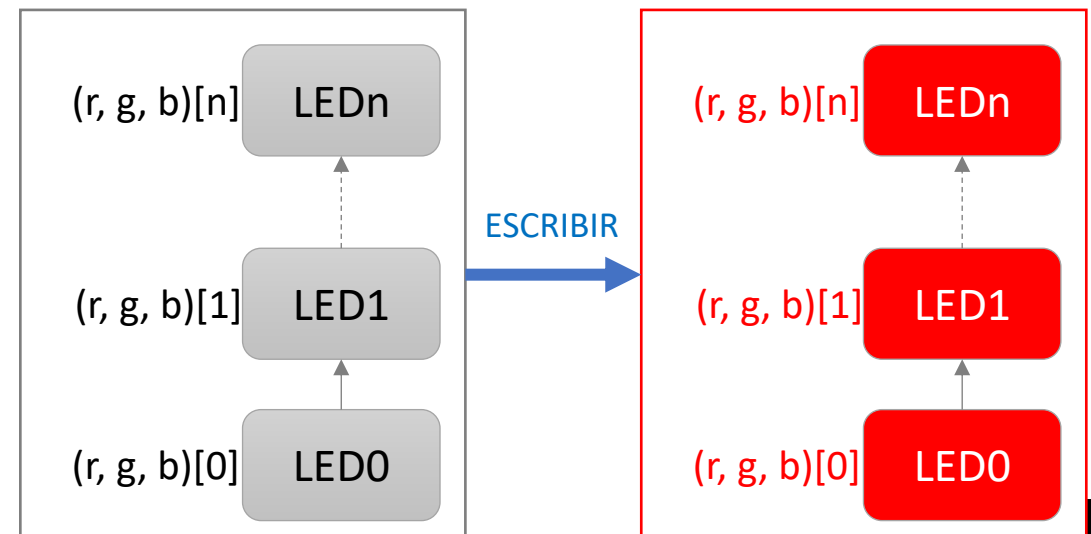
Muchos controladores de display en uPython ( y en lenguaje Arduino), están diseñados como 2 pizarras que representan el display, pixel a pixel.

1. Se manipula la representación del display en la “pizarra” del controlador
2. Se vuelca la pizarra del controlador a la pizarra del dispositivo físico (que guarda en una memoria del dispositivo físico)
3. Se hacen nuevos cambios en la pizarra del controlador ( no se verán en el display)
4. Se vuelca de nuevo la pizarra del controlador a la pizarra del dispositivo físico

..... Ciclo ....

En el caso del los neopixel las pizarras son lineales = una lista de tuplas de 3 números x numero de leds.

Cada tupla tiene 3 valores de 0-255 con el brillo de cada color rojo, verde y azul



# Clase 2425\_12.1 [R] Neopixel - Uso básico Conexiones Ideales

Importar librería : viene con uPyhton **FUNCIONES MUY BASICAS**  
`from neopixel import NeoPixel`

Como se define el color de cada pixel

# Cada pixel se define por 3 valores de color en el orden RGB  
# es decir (255, 0, 0) = rojo ; (0, 255, 0) = verde ; (0, 0, 255) = azul;  
# y todas las combinaciones de valores desde 0 a 255

1. Creamos el objeto con la clase Neopixel

`NUMERO_PIXELS = 1`

`NEOPIXEL_PIN = 15`

`tira = NeoPixel(Pin(NEOPIXEL_PIN), NUMERO_PIXELS )`

2. Configuramos la pizarra del ucontrolador

`tira[0] = (255, 0, 0)`

`tira[1] = (0, 255, 0)`

3. Volcamos a la pizarra del dispositivo físico

`tira.write()`

4. Repetimos 2 -> 3, para ir cambiando la configuración

**Alimentación** : VDD es el +, VSS –

El WS2812B se puede alimentar desde 3.5 a 5.3 volt. A 5 volt lucirá mas

1. Voltios

- -> **5 volt de VBUS en el Pico o**
  - 5 volt externos, con GDN común
- + Condensador 1000uF

2. Amperios

Cada neopx consume un max 60mA, ➔ calcular potencia de la fuente con un valor medio de 20ma por neopx

**Data In (DIN)**

Si se alimentan los neopx a 5volt, necesitamos un cambiador de niveles lógicos para DIN

+ añadir R de 300 a 500 ohm como protección del primer neopx

Se puede alimentar a 3.3 de pata 36 física de la Pico W, VOUT, pero lucirá menos, ¡Tened en cuenta que VOUT puede dar max 300mA -> 15 neopx max!

# Clase 2425\_12.1[R] Neopixel - **Complejo** -El lio de tener niveles lógicos de 5volt y de 3.3 vol - [Ver artículo del blog de Luis Llamas](#)

Los valores lógicos H y L se corresponden a unas bandas de valores de voltaje, en los dispositivos digitales

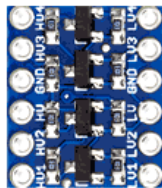
**Para que dos dispositivos digitales se entiendan han de usar las mismas bandas para H y L o que estas bandas sean compatibles** (dependerá de quien es entrada y quien salida)

A veces usaremos módulos diseñados para la tarjeta Arduino a 5 volt (lógica TTL), o que se deben alimentar a 5volt como los Neopixel (lógica CMOS a 5volt). —> **Lo mas sencillo y seguro es usar un conversor de nivel bidireccional** ejemplo de level shifter [en amazon](#) / [aliexpres](#)

**Lado Neopixel a 5volt**

LADO NIVEL ALTO

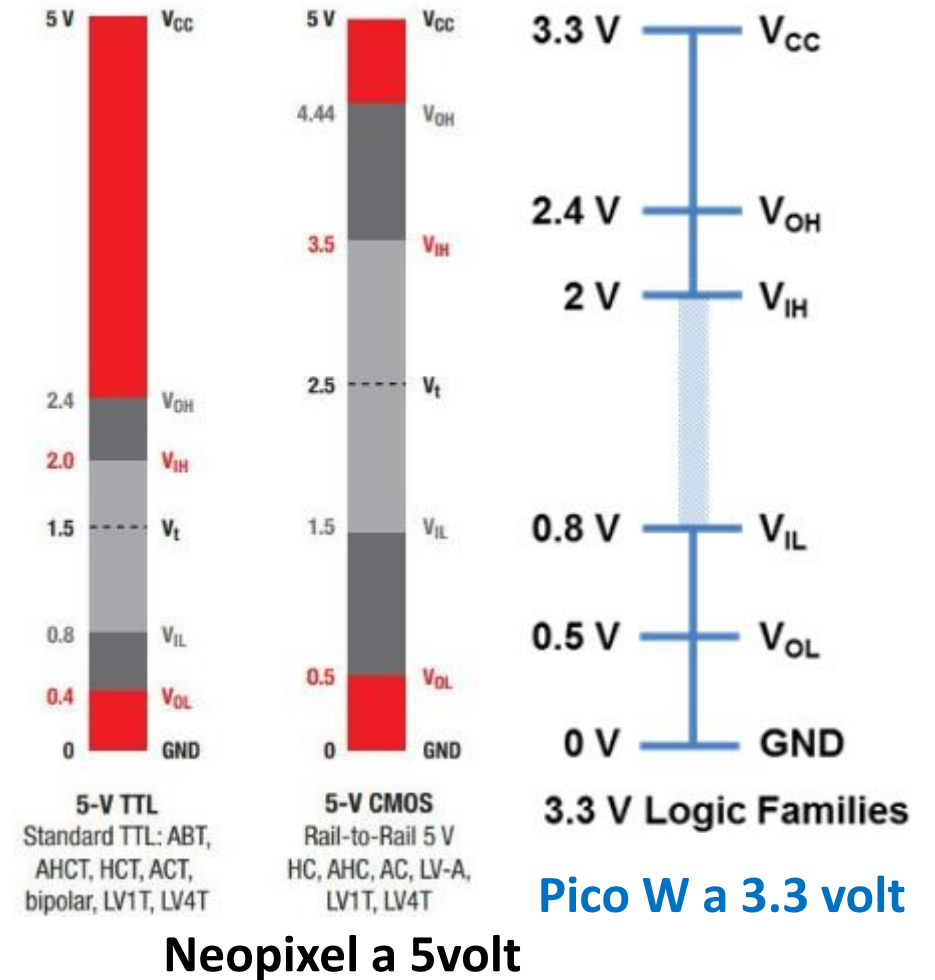
GPIO — HV4  
GPIO — HV3  
GND — GND  
⚠ 5V — HV  
GPIO — HV2  
GPIO — HV1



LADO NIVEL BAJO

LV4 — GPIO  
LV3 — GPIO  
GND — GND  
LV — 3.3V ⚠  
LV2 — GPIO  
LV1 — GPIO

**Lado Pico W a 3.3 volt**



# Clase 2425\_12.1 [R] Neopixel - Ejemplo de Uso básico

```
from machine import Pin
from time import sleep
from neopixel import NeoPixel
```

**BMMR\_CL10\_neopixel2\_basic\_1\_0.py**

```
# El brillo es un valor de 0 a 255, usaremos el mismo para cada color
# Cada pixel se define por 3 valores de color en el orden RGB
# es decir (255, 0, 0) = rojo ; (0, 255, 0) = verde ; (0, 0, 255) = azul;
# y todas las combinaciones de valores desde 0 a 255 en cada color

BRILLO = 127 # brillo medio

# Aqui definimos la espera entre cada cambio a segundos
ESPERA = 2

# 1- Crea el objeto neopixel
NUMERO_PIXELS = 2
NEOPIXEL_PIN = 17
tira = NeoPixel(Pin(NEOPIXEL_PIN),NUMERO_PIXELS )
```

# Clase 2425\_12.1 Neopixel - Ejemplo de Uso básico

```
# 2.1- configuramos pizarra del controlador a rojo y verde
```

```
tira[0] = (255, 0, 0)
```

```
tira[1] = (0, 255, 0)
```

```
# 3.1 Volcamos pizarra de ucontrolador a la del dispositivo y esperamos
```

```
tira.write()
```

```
sleep(ESPERA)
```

```
# 2.2- configuramos pizarra del controlador a azul y rojo
```

```
tira[0] = (0, 0, 255)
```

```
tira[1] = (255, 0, 0)
```

```
# 3.2 Volcamos pizarra de ucontrolador a la del dispositivo y esperamos
```

```
tira.write()
```

```
sleep(ESPERA)
```

```
# 2.3- configuramos pizarra del controlador a rojo+azul y rojo+verde
```

```
tira[0] = (BRILLO, 0, BRILLO)
```

```
tira[1] = (BRILLO, BRILLO, 0)
```

```
# 3.3 Volcamos pizarra de ucontrolador a al del dispositivo y esperamso
```

```
tira.write()
```

```
sleep(ESPERA)
```

```
# 2.4- configuramos pizarra del controlador a todos apagados
```

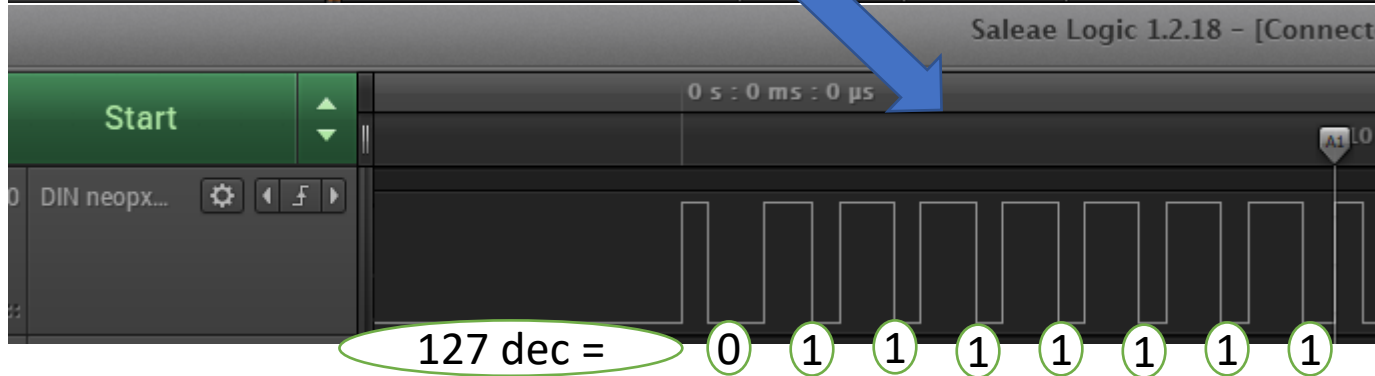
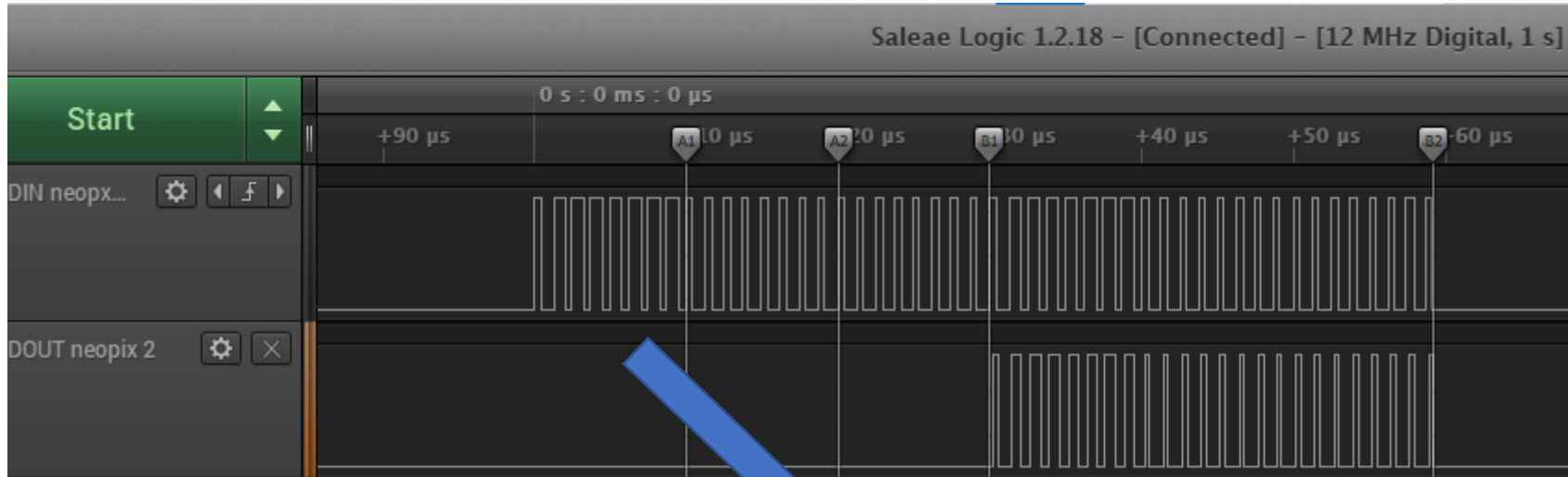
```
tira.fill((0, 0, 0))
```

```
# 3.4 Volcamso pizarra de ucontrolador a la del dispositivo y Fin
```

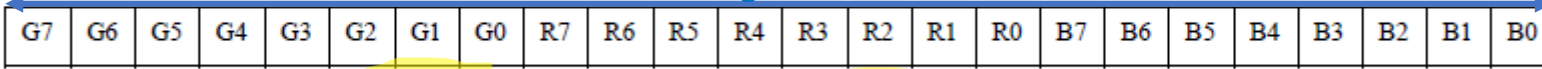
```
tira.write()
```



# Clase 2425\_12.1 Neopixel - **Avanzado** Detalle de protocolo, Ejemplo enviando (0,127,0)(r,g,b) a los 2 neopixel

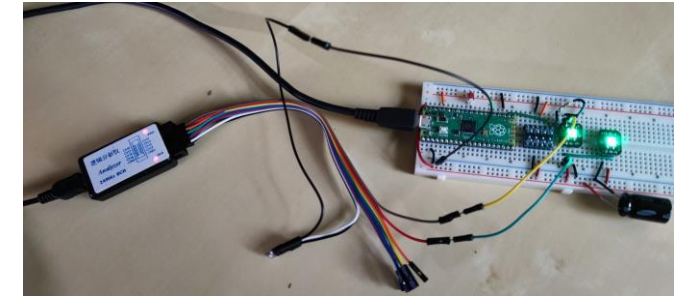
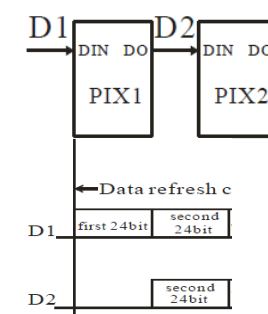


Composition of 24bit data: 30 usegundos



Note: Follow the order of GRB to sent data and the high bit sent at first.

Cascade method:



Data transfer time( TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.4us
T1H	1 code ,high voltage time	0.8us
T0L	0 code , low voltage time	0.85us
T1L	1 code ,low voltage time	0.45us
RES	low voltage time	Above 50μs

800 kilo bits por segundo =  
 $1/800000 = 1,25 \mu$  segundos por bit  
 $1,25 \times 8 = 10 \mu$ sec por 8 bits  
 30 usec para 24 bits

0 code  $\left[ \begin{array}{c} \text{T0H} \\ \text{T0L} \end{array} \right] = 1,25 \mu$ segundos

1 code  $\left[ \begin{array}{c} \text{T1H} \\ \text{T1L} \end{array} \right] = 1,25 \mu$ segundos

RET code  $\left[ \begin{array}{c} \text{Treset} \end{array} \right]$

# Clase 2425\_12.2 [RyP] Neopixel - usos +complejos creando funciones en un anillo de neopixel x16

Ya dijimos a que la librería que viene con uPyhton tiene **FUNCIONES MUY BASICAS:**

- Crear el objeto neopixel
- Escribir en uno de los pixeles un color con 3 enteros de 0-255
- Escribir en todos los neopixel el mismo color
- Leer uno de los pixeles
- Volcar al neopixel la “pizarra” interna a al del objeto neopixel físico

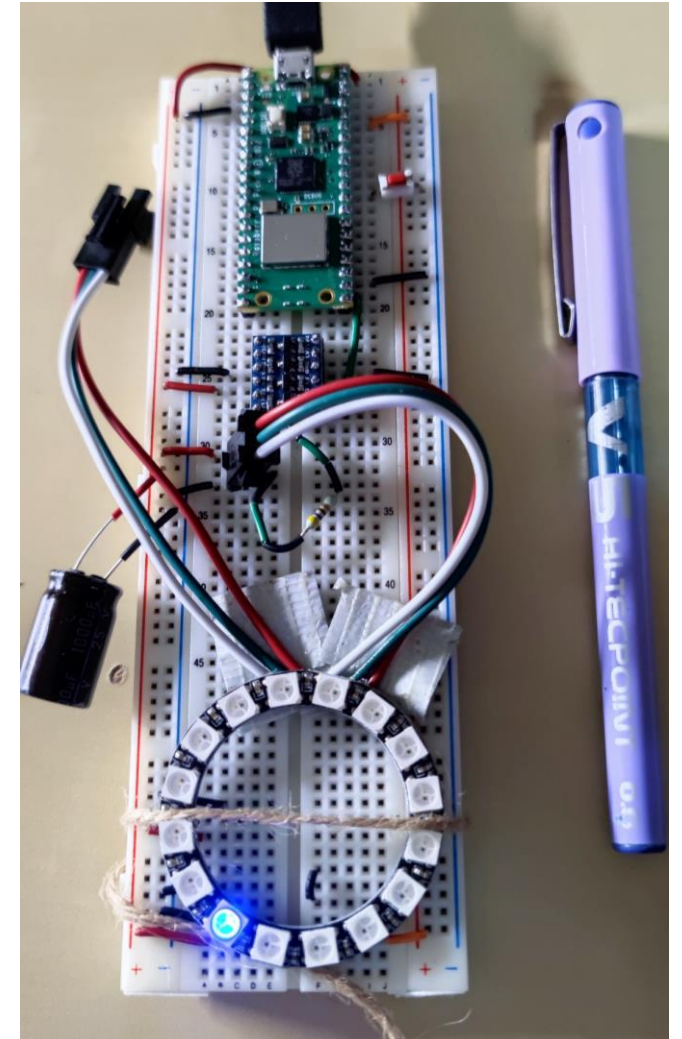
Para hacer cosas más elaboradas necesitamos hacerlo con funciones que usen intensivamente el bucle **for** y **range**

Los bucles **for** se emplean para implementar los **bucles de final definido**, pero se pueden implementar distinto en cada lenguaje

**Avanzado:** En Python los bucles **for** se han implementado como **iteraciones sobre una colección**, es la forma mas versátil aunque si se viene de C resulta extraña.

[Ver link](#) para tutorial en ingles muy completo de realpython

[Otro link](#) con tutorial en castellano



# Clase 2425\_12.2[R] Neopixel - 1 color básico dando vueltas 2 versiones

[BMMR\\_CL11\\_neopixel16\\_rota3\\_1\\_0.py](#)

```
while True:
    for i in range(0, NUMBER_PIXELS):
        anillo[i] = red
        anillo.write()
        sleep(ESPERA)
        anillo[i] = apagado
    for i in range(0, NUMBER_PIXELS):
        anillo[i] = green
        anillo.write()
        sleep(ESPERA)
        anillo[i] = apagado
    for i in range(0, NUMBER_PIXELS):
        anillo[i] = blue
        anillo.write()
        sleep(ESPERA)
        anillo[i] = apagado
```

[BMMR\\_CL11\\_neopixel16\\_rota3\\_1\\_1.py](#)

```
listacol = [red, green, blue]

while True:
    for c in listacol:
        # el color c pasa por cada led del anillo
        for i in range(0, NUMBER_PIXELS):
            anillo[i] = c
            anillo.write()
            sleep(ESPERA)
            anillo[i] = apagado
```

1. Prueba a cambiar el color del resto de los neopixels por “blanco” por ejemplo
2. Para mover un hueco, intercambia “c” y “apagado”

# Clase 2425\_12.2 [R] Neopixel - Arcoiris adaptación de programa en C con 2 funciones y la 2da con 2 bucles **for**

**BMMR CL11 neopixel16 iris 1 0.py**

```
def iris(pos):  
    if pos < 0 or pos > 255:  
        return (0, 0, 0)  
  
    if pos < 85: # 1ra region  
        return (255 - pos * 3, pos * 3, 0)  
  
    if pos < 170: # 2da region  
        pos -= 85  
        return (0, 255 - pos * 3, pos * 3)  
  
    pos -= 170 # 3ra region  
    return (pos * 3, 0, 255 - pos * 3)
```

Mapea el arcoiris de 0 a 255 en una tupla RGB

Parametros:

pos : 0 a 255 mapea el arcoiris en 3 regiones de 84 de ancho

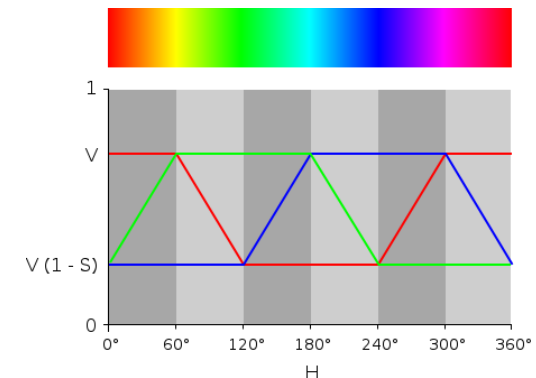
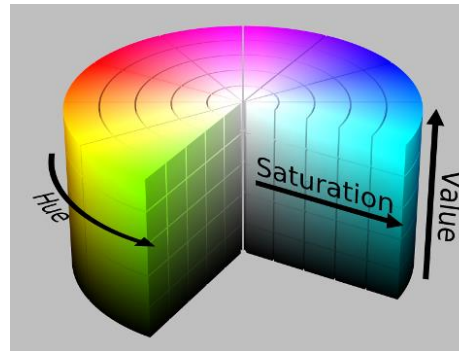
1ra region suma de R y G , pero R disminuye y G aumenta

2da region suma de G y B , pero G disminuye y B aumenta

3ra region suma de B y R , pero B disminuye y R aumenta

Retorno:

una tupla en forma (r, g, b)



# Clase 2425\_12.2 [R] Neopixel - Arcoiris adaptación de programa en C con 2 funciones y la 2da con 2 bucles **for**

```
def cicloArcoiris(espera):  
    global NUMBER_PIXELS, anillo  
    for j in range(255):  
        for i in range(NUMBER_PIXELS):  
            rc_index = (i * 256 // NUMBER_PIXELS) + j  
            anillo[i] = iris(rc_index & 255)  
        anillo.write()  
    sleep(espera)
```

Distribuye el arcoíris en el anillo o tira neopixels usando función iris

Global: usa 2 variables globales

NUMBER\_PIXELS : del anillo o tira neopixel

anillo : objeto neopixel

El **for** interior calcula cada pixel del anillo, espaciando los 256 grados del arcoíris en 256/num\_pixels

El 2do **for** aumenta una posición el arcoíris en cada pixel . Si el num\_pixel es divisor de 256 todo pega Ver tabla

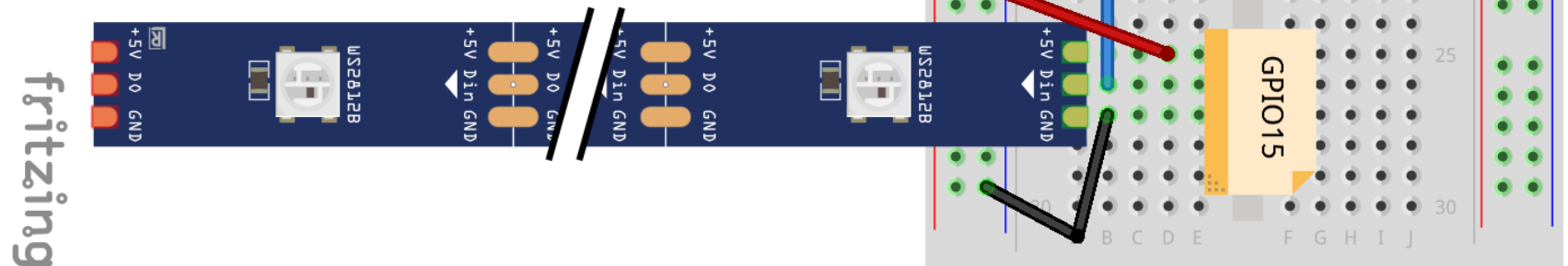
j	i	rc_index	rc_index & 255		Npx_0	Npx_1	Npx_2	Npx_3	Npx_4	Npx_5	Npx_6	Npx_7	Npx_8	Npx_9	Npx_10	Npx_11	Npx_12	Npx_13	Npx_14	Npx_15
0	15	240	240		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1	15	241	241		1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241
2	15	242	242		2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
3	15	243	243		3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
					..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
15	15	255	255		15	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255
16	15	256	0		16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	0
17	15	257	1		17	33	49	65	81	97	113	129	145	161	177	193	209	225	241	1
					..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
253	15	493	237		253	13	29	45	61	77	93	109	125	141	157	173	189	205	221	237
254	15	494	238		254	14	30	46	62	78	94	110	126	142	158	174	190	206	222	238
					..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
0	15	240	240		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1	15	241	241		1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241



# Clase 2425\_12.3 – [R] Usar Wifi Leer cheerlight

**OBJETIVO:** usar la conexión a internet para leer una información muy simple: un código de color RGB (3 x 1byte = 24 bits) que usaremos para encender un led tipo neopixel.

**Montaje HW : solo 1 neopixel en GPIO15, alimentamos a 5 volt, aunque la lógica de Din sea de 3.3volt**



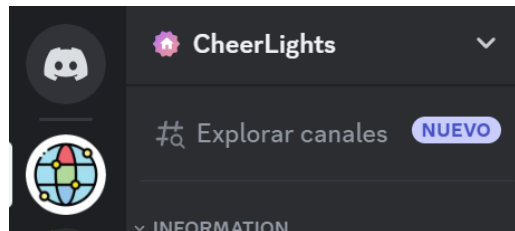
# Clase 2425\_12.3 – [R] Usar Wifi Leer cheerlight

**OBJETIVO:** usar la conexión a internet para leer una información muy simple: un código de color RGB que usaremos para encender un led tipo neopixel. **Montaje HW : solo 1 neopixel**, alimentamos a 3.3volt, en GPIO17

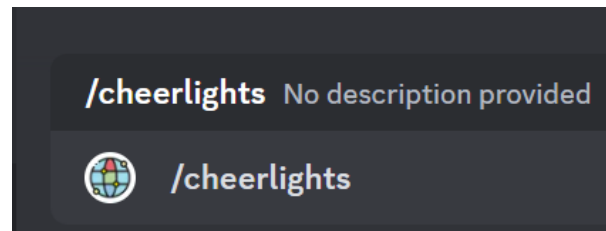
[El tutorial de sunfounder cheerlight](#) es correcto salvo en la parte de Cheerlight => Hay que usar un *server* de [discord](#) para ver el color y cambiarlo. Este código de color se puede cambiar por nosotros o cualquier persona dada de alta en 'cheerlights', ver videotutorial [Controlling CheerLights on Discord](#)

Examinar: [BMMR CL19s wifi out cheer1 1 2.py](#) → Ejecutar en consola **lee\_color** línea a línea para aprendizaje

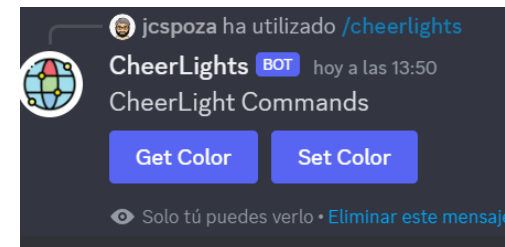
1. Entrar en discord  
>Cheerlight  
(hace falta invitación)



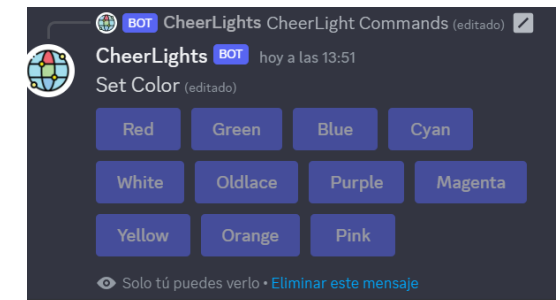
2. En el canal general ,  
intro comando  
**/cheerlight**



3. Click **Set Color**



4. Click en **botón de color**



## Librería requests

Simplifica el manejo de API's. Vamos a usarla intensivamente. En este ejemplo solo leemos de una api muy simple

[Documentación Requests](#)

[Tutorial Requests](#)

## JSON

Javascript Object Notation

[Tutorial simple](#)

## Try –Except

para que no se quede neopixel encendido al salir, por una interrupción de teclado Ctrl+C o STOP