

Universidad Politécnica Salesiana  
Ingeniería en Electrónica y Automatización.

Integrantes Chipugri Anthony  
Oibe Cinthya

Daniel XJ

Práctica: #3

Tema: Clases en Rasperry.

Fecha: 05 / 11 / 2025

Daniel  
05 - 11 - 25

Daniel

Práctica #3

Trabajo en Clase Herencia

Fecha: 10 / 11 / 2025

Daniel

Trabajo Robot herencia + Rasperry Pi

Fecha: 13 / 11 / 2025

Daniel

RECUPERACIÓN

Trabajo API con Telegram (led on - led off)

Fecha: 21 / 11 / 2025

PRACTICA #5

Daniel

RECUPERACIÓN

Trabajo CLASS ROBOT + API + ARCHIVO + TELEGRAM PRACTICA #6

Fecha: 21 / 11 / 2025

Daniel

RECUPERACIÓN

Robot LED + BUTTON + DHT11

Fecha: 21 / 11 / 2025

Daniel

ESTILO

# POO con Herencia y Módulos

## Práctica 3

Anthony Fabricio Chipugsi Pilicita

[achipugsip@est.ups.edu.ec](mailto:achipugsip@est.ups.edu.ec)

Cinthya Aracelly Orbe Muñoz

[corbem@est.ups.edu.ec](mailto:corbem@est.ups.edu.ec)

### I. Introducción

La Raspberry Pi 3, al ser una plataforma versátil y accesible, ofrece la posibilidad de implementar prácticas que integren hardware y software. En este caso, se emplea PuTTY como herramienta de conexión remota para ejecutar programas en Python que permiten controlar periféricos básicos como un LED y un botón.

El objetivo de esta práctica es demostrar cómo la POO, mediante el uso de clases, herencia y módulos, puede aplicarse en la programación de dispositivos electrónicos, logrando un código más organizado y fácil de mantener. A través de la interacción entre el LED y el botón, se ejemplifica la manera en que los conceptos teóricos se trasladan a un entorno físico, reforzando la comprensión de la programación modular y orientada a objetos en un contexto real

### II. Metodología

#### 1. Código modular: Módulos

```
GNU nano 8.4
import RPi.GPIO as GPIO
import time

class butBoard:
    def __init__(self, pin, boton):
        self.pin=pin
        self.boton=boton
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(self.pin, GPIO.OUT)
        GPIO.setup(self.boton, GPIO.IN)
    def encender(self):
        while True:
            if GPIO.input(self.boton):
                GPIO.output(self.pin, False)
            else:
                GPIO.output(self.pin, True)

        GPIO.cleanup()

board=butBoard(12,22)
board.encender()
```

Figura 1. Código modular clase butBoard

-Primero se importa la librería RPi.GPIO, que permite controlar los pines GPIO de la Raspberry.

-Se le asigna el alias GPIO para facilitar su uso en el resto del código. [1]

-Se importa el módulo time, que se usa para generar pausas temporales en la ejecución del programa.

-al escribir `class ledBCM:` se define una clase llamada ledBCM, que encapsula el comportamiento de un LED controlado por GPIO en modo BCM.

- En la línea: `def __init__(self, pin):`, se define el constructor de la clase.

-En la línea: `self.pin = pin`, se guarda el número de pin en el atributo self.pin para que esté disponible en otros métodos de la clase.

```
GNU nano 8.4
import RPi.GPIO as GPIO
import time

class ledBCM:
    def __init__(self, pin):
        self.pin=pin
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.pin, GPIO.OUT)
    def parpadear(self):
        print("El led esta encendido... (Ctrl+C para detener)")

        while True:
            GPIO.output(self.pin, True)
            time.sleep(1)
            GPIO.output(self.pin, False)
            time.sleep(1)

        GPIO.cleanup()

led=ledBCM(18)
led.parpadear()
```

Figura 2. Código modular clase ledBCM

-Se realiza lo mismo que la anterior programación, con la diferencia de definir otra clase llamada butBoard, que representa un sistema con un botón y un LED, luego se define el constructor de la clase y recibe dos parámetros: pin para el LED y boton.

## 2. Código modular: Archivo principal

```
# main.py
from dispositivos import LED, BotonLED
# Ejemplo 1: Parpadeo de LED usando modo BCM
led = LED(pin=18, modo='BCM')
led.parpadear(intervalo=1)
```

Figura 3. Código modular de archivo principal.

## II. Resultados

### Explicación de cómo se aplica la herencia:

-Clase: DispositivoGPIO

-Atributo: pin

-Métodos: encender(), apagar(), limpiar()

-Modos: BCM y BOARD

-Subclase 1 LED: Hereda de DispositivoGPIO y extiende su funcionalidad con el método parpadear(), no necesita redefinir encender() ni apagar(), porque ya los hereda.

-Subclase 2 BotonLED: también hereda de DispositivoGPIO, se añade un nuevo atributo de botón y se agraga el método controlar\_led() que usa el botón como entrada para controlar el LED.

-Finalmente se usa el super().\_\_init\_\_ para invocar el constructor de la clase, evitando repetir la configuración del pin.

## IV. Discusión

Esta práctica demuestra cómo aplicar la Programación Orientada a Objetos en sistemas embebidos con Raspberry Pi 3, utilizando herencia y modularización para controlar un LED y un botón. Se diseñó una clase base (DispositivoGPIO) que encapsula la configuración común de los pines GPIO, y se extendió mediante subclases especializadas (LED y BotonLED) que implementan comportamientos específicos como parpadeo automático o control por botón.

La estructura modular del código facilita la reutilización, mejora la organización y permite escalar el sistema a nuevos dispositivos. Además, se integró manejo de excepciones para garantizar una ejecución segura y una liberación adecuada de recursos. En conjunto, la práctica refuerza conceptos clave de la POO y demuestra su utilidad en aplicaciones físicas reales.

## V. Conclusiones

- La creación de una clase base (DispositivoGPIO) permitió centralizar la configuración de los pines y reutilizarla en subclases especializadas (LED y BotonLED),

reduciendo redundancia y facilitando la extensión hacia nuevos dispositivos.

- Separar la lógica en un módulo (dispositivos.py) y un archivo principal (main.py) mejoró la estructura del proyecto, facilitando la comprensión, el mantenimiento y la posibilidad de escalar el sistema a aplicaciones más complejas.

- El uso de la Raspberry Pi 3 junto con PuTTY permitió trasladar conceptos teóricos de POO a un entorno físico, reforzando la comprensión de la programación modular y demostrando su utilidad en la automatización y control de dispositivos electrónicos.

## VI. Referencias

[1] Raspberry Pi Foundation, “GPIO and RPi.GPIO library,” Raspberry Pi Docs, 2025. [Online]. Available: <https://www.raspberrypi.com/documentation/computers/os.html#gpio>

## VII. Anexos

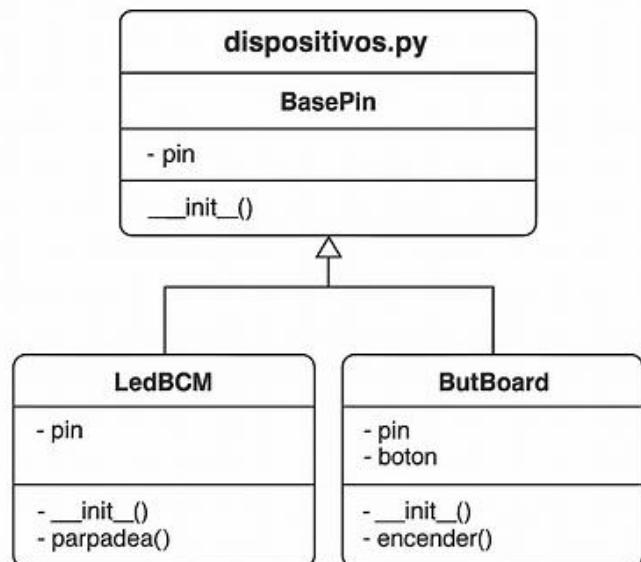


Figura 4. Diagrama UML de las clases.