1. Introducción

- **Objetivo:** Explorar las opciones de entornos de desarrollo para Python en Raspberry Pi.
- **Lenguaje:** Python es ideal debido a su simplicidad, versatilidad y amplia compatibilidad con hardware como GPIO, I2C, SPI y UART.

2. Entornos de Programación

- **Thonny:** Ideal para principiantes, viene preinstalado y es fácil de usar, aunque limitado para proyectos avanzados.
- **PyCharm (Community Edition):** Poderoso y adecuado para proyectos complejos, pero puede ser pesado para versiones de Raspberry Pi con menos recursos.
- **Geany:** Ligero y rápido, ideal para tareas básicas.
- **Mu Editor:** Diseñado para educación y proyectos simples.
- **IDLE:** Básico pero funcional para escritura de código inicial.

3. Recomendación Principal: Visual Studio Code (VS Code)

- **Ventajas:**
    - Ligero y potente con herramientas avanzadas como depuración, autocompletado y Git.
    - Extensibilidad mediante bibliotecas y plugins específicos.
    - Integración de inteligencia artificial con GitHub Copilot.
    - Multiplataforma y gratuito.

4. Tutorial: Instalación de Visual Studio Code

Pasos detallados para instalar VS Code en Raspberry Pi, incluyendo actualización del sistema, instalación de dependencias y configuración del repositorio oficial de Microsoft.

5. Mi Primer Programa: "Hola Mundo"

- Configuración de Python y pip.
- Creación de un archivo de proyecto (hola_mundo.py) en VS Code.
- Ejecución del programa con el comando python3 hola_mundo.py.

6. Extensiones Recomendadas

- **Python Extension:** Funcionalidades como autocompletado, depuración, linting, y soporte para pruebas unitarias.
- **Python Debugger:** Herramientas avanzadas para depuración, crucial para proyectos que interactúan con hardware.

1. Introducción a los Pines GPIO

- **Qué son:** Los GPIO (General Purpose Input/Output) son pines versátiles en la Raspberry Pi que permiten conectar y controlar dispositivos externos como sensores, LEDs y motores.
- **Cantidad:** La Raspberry Pi 4 cuenta con 40 pines organizados en diferentes categorías.

2. Categorías de Pines

1. **Pines de Alimentación:**
    a. Pines de **3.3V** (1 y 17): Alimentan componentes de bajo voltaje.
    b. Pines de **5V** (2 y 4): Conectados directamente a la fuente de alimentación USB-C.
2. **Pines de Tierra (GND):**
    a. Incluyen los pines 6, 9, 14, 20, 25, 30, 34 y 39.
3. **Pines GPIO:**
    a. Configurables como entradas o salidas para interactuar con dispositivos externos.
    b. Operan a **3.3V** y no son tolerantes a **5V**.
4. **Pines de Comunicación:**
    a. **I2C:** Pines 3 (SDA) y 5 (SCL).
    b. **SPI:** Pines 19 (MOSI), 21 (MISO), 23 (SCLK), y 24/26 (CE0/CE1).
    c. **UART:** Pines 8 (TXD) y 10 (RXD).
5. **Pines PWM:**
    a. Pines 12, 32, 33, y 35 para modulación por ancho de pulso, útiles para controlar motores y LEDs.

3. Limitaciones Técnicas

- **Voltaje:** Los GPIO funcionan a 3.3V; exponerlos a 5V puede dañarlos.
- **Corriente por pin GPIO:** Hasta 16 mA, ideal para pequeños dispositivos como LEDs.
- **Corriente total de GPIO:** No exceder 50 mA en todos los pines combinados.
- **Precaución en Pines de 5V:** Sin protección contra sobrecorriente; evitar cortocircuitos.

4. Precauciones y Buenas Prácticas

- Usar resistencias para limitar la corriente y proteger los pines.
- Verificar conexiones antes de energizar el circuito.
- Preferir transistores o relés para manejar corrientes mayores.

Práctica :Entradas Salidas digitales

El objetivo principal es conectar un pulsador y mostrar su estado en tiempo real en la consola de Visual Studio Code.

**Al final del documento se encuentran los esquemas y la referencia del codigo

1. Objetivo de la Práctica

- Conectar un pulsador a un pin GPIO y mostrar su estado (presionado o no) en la consola usando Python.
- Introducir conceptos básicos de control de hardware mediante software en la Raspberry Pi.

2. Materiales Necesarios

- Raspberry Pi con Raspberry Pi OS.
- Pulsador (botón).
- Resistencia de 10 kΩ (pull-down resistor).
- Cables macho-macho.
- Protoboard.
- Visual Studio Code instalado en la Raspberry Pi.

3. Configuración del Circuito

1. **Pulsador**:
    a. Un terminal conectado al pin GPIO17 (pin 11).
    b. Otro terminal conectado a GND (pin 6).
2. **Resistencia**:
    a. Conecta la resistencia de 10 kΩ entre el terminal del pulsador y GND.
    b. Esto asegura que el pin GPIO lea "LOW" cuando el pulsador no está presionado.

4. Código Python

El programa configura el GPIO para leer el estado del pulsador y mostrarlo en la consola.

Características del Código:

- Uso de la biblioteca RPi.GPIO para controlar los pines GPIO.
- Configuración del pin como entrada con resistencia interna pull-down (GPIO.PUD_DOWN).
- Lógica de lectura continua del estado del pulsador en un bucle infinito.
- Gestión adecuada de recursos mediante GPIO.cleanup() al finalizar.

Ejemplo de Mensajes en Consola:

- Pulsador no presionado: "Pulsador no presionado".
- Pulsador presionado: "¡Pulsador presionado!".

5. Ejecución del Programa

- Guarda el archivo como pulsador.py.
- Ejecuta el comando:

```
sudo python3 pulsador.py
```

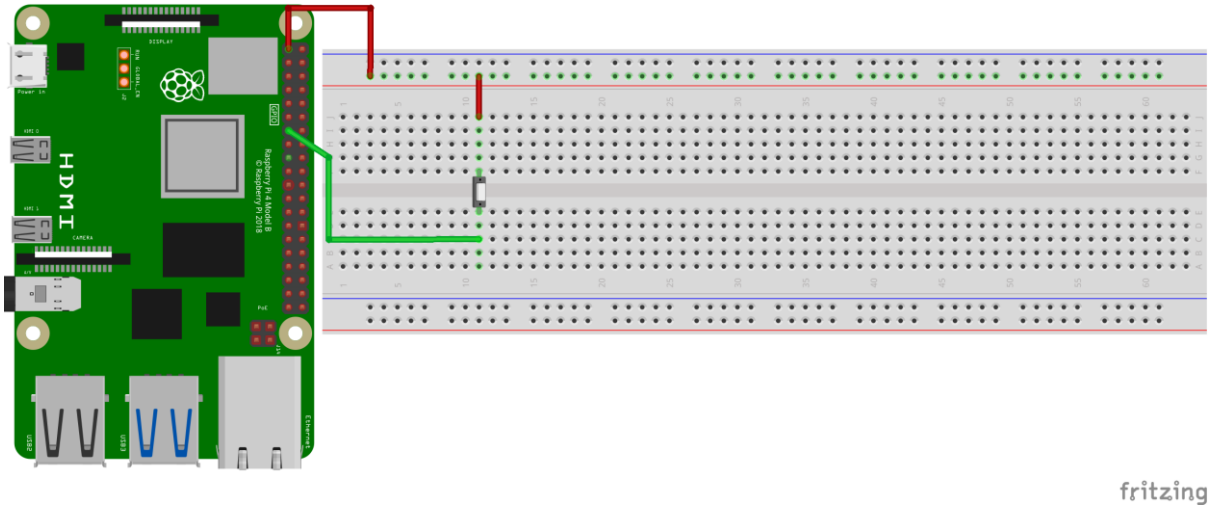
- Observa los mensajes en la consola según el estado del pulsador.
- Interrumpe el programa con Ctrl + C.

6. Manejo de Errores y Configuración Alternativa

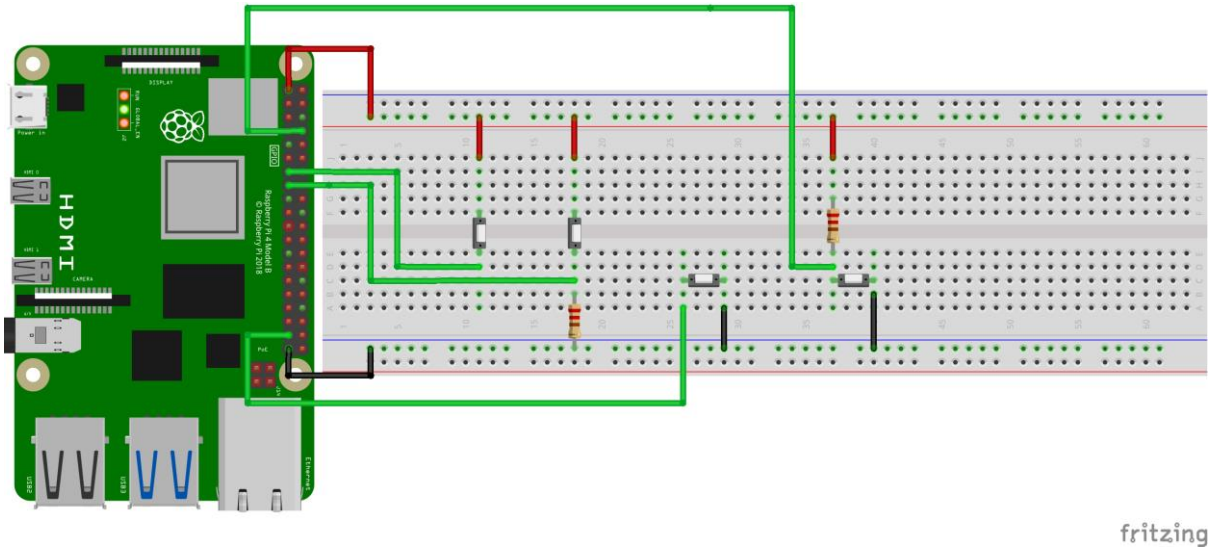
Si se encuentra un error al instalar la biblioteca RPi.GPIO debido a restricciones del sistema:

- Usa el comando recomendado:

```
sudo apt install python3-rpi.gpio
```



Conexión Input-PullUp



Conexiones PullUp y PullDown (internas y externas)

En el apartado “scripts pyhton” código “entradas_digitales.py”

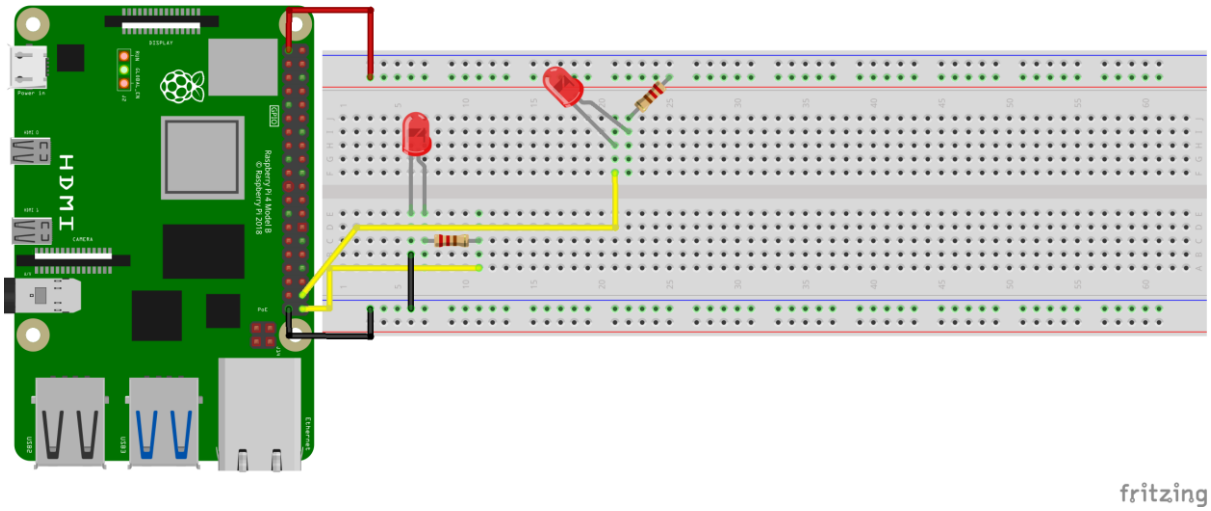
Práctica salidas digitales

En esta sesión, exploramos cómo controlar los pines GPIO para generar **señales de salida**. Esto incluye tareas como encender y apagar LEDs o interactuar con otros dispositivos conectados.
**Al final del documento se incluyen códigos y esquemas.

1. **Salidas digitales**: Configurar los pines GPIO para enviar señales de encendido (HIGH) y apagado (LOW).
2. **Práctica de control**: Aprenderemos cómo manipular LEDs o relés a través de Python.
3. **Comparativa con entradas digitales**: Recordamos conceptos de la sesión previa, donde trabajamos con entradas digitales para leer estados de botones y configuraciones predeterminadas.

Resumen

1. **Objetivo de la Clase**:
    a. Controlar salidas digitales en los pines GPIO para manipular dispositivos como LEDs.
    b. Aprender los conceptos básicos necesarios para combinar entradas y salidas en proyectos futuros.
2. **Herramientas y Prácticas**:
    a. Uso de conectores y protoboards para organizar pines.
    b. Configuración de pines GPIO en modo salida mediante Python.
    c. Ejemplo práctico con LEDs: cómo encender y apagar según instrucciones programadas.



Circuito led PullUp y PullDown

En el apartado “scripts pyhton” código “salidas_digitales.py”

Práctica entradas por terminal

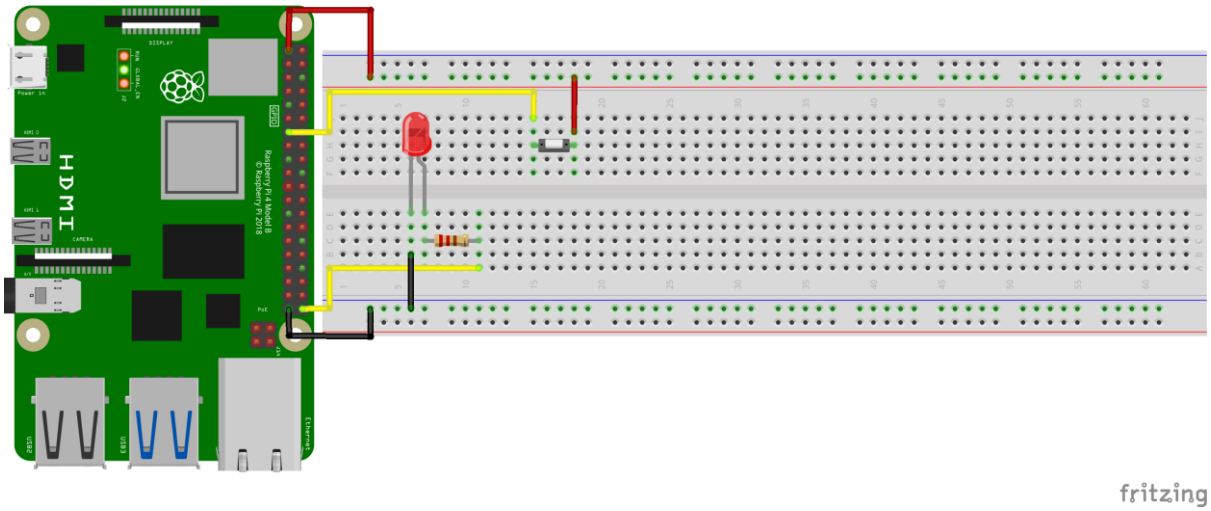
En esta sesión, vamos a aprender a combinar la entrada de información desde la terminal con el control de los pines GPIO. Ahora que ya hemos explorado tareas básicas de control, avanzaremos hacia una integración más compleja, como programar la comunicación con módulos y protocolos como SPI, I2C, etc. Esto ampliará las capacidades de interacción de la Raspberry Pi.
**Al final del documento se incluyen códigos y esquemas.

Temas destacados:

1. **Entrada desde la terminal:** Permite recibir órdenes desde el usuario para controlar dispositivos conectados.
2. **Control de dispositivos:** Basado en las órdenes recibidas, podemos encender o apagar elementos como LEDs o consultar estados, como si un botón está pulsado o el valor de un sensor.
3. **Configuraciones lógicas:** Por ejemplo:
    a. Cuando una salida está activa, un LED se enciende.
    b. Cuando un pulsador no está presionado, el pin mide 0V; al presionarlo, mide 3.3V.

Práctica explicada:

- Se utiliza una resistencia pull-down interna para simplificar la configuración, evitando la necesidad de usar componentes externos adicionales.
- El ejemplo práctico incluye:
    - Configuración de un pulsador como entrada.
    - Un LED como salida.
    - Control desde la terminal para activar o consultar estados.



Control de led con pulsador

En el apartado “scripts pyhton” código “entradas_terminal.py”

Práctica Python y Linux

En esta sesión, aprenderemos a lanzar comandos de Linux desde un programa de Python. Esto amplía considerablemente las funcionalidades de la Raspberry Pi, permitiéndonos, por ejemplo:

- Activar un ventilador en función de la temperatura para refrigerar la Raspberry.
- Reiniciar la Raspberry si se pulsa un botón externo.
- Realizar una amplia gama de automatizaciones y tareas generales según las necesidades.

**Al final del documento se incluyen códigos y esquemas.

Ejemplo práctico:

- Utilizamos un comando común para consultar la temperatura del sistema:

```
vcgencmd measure_temp
```

- **Resultado del comando:** El formato de la respuesta será algo como:

```
temp=50.0'C
```

- Donde:
  - temp= indica el inicio del valor.
  - El valor numérico es la temperatura actual.
  - 'C indica grados Celsius.

En el apartado “scripts pyhton” código “python_linux.py”

Práctica regulación salida

¿Qué es PWM?

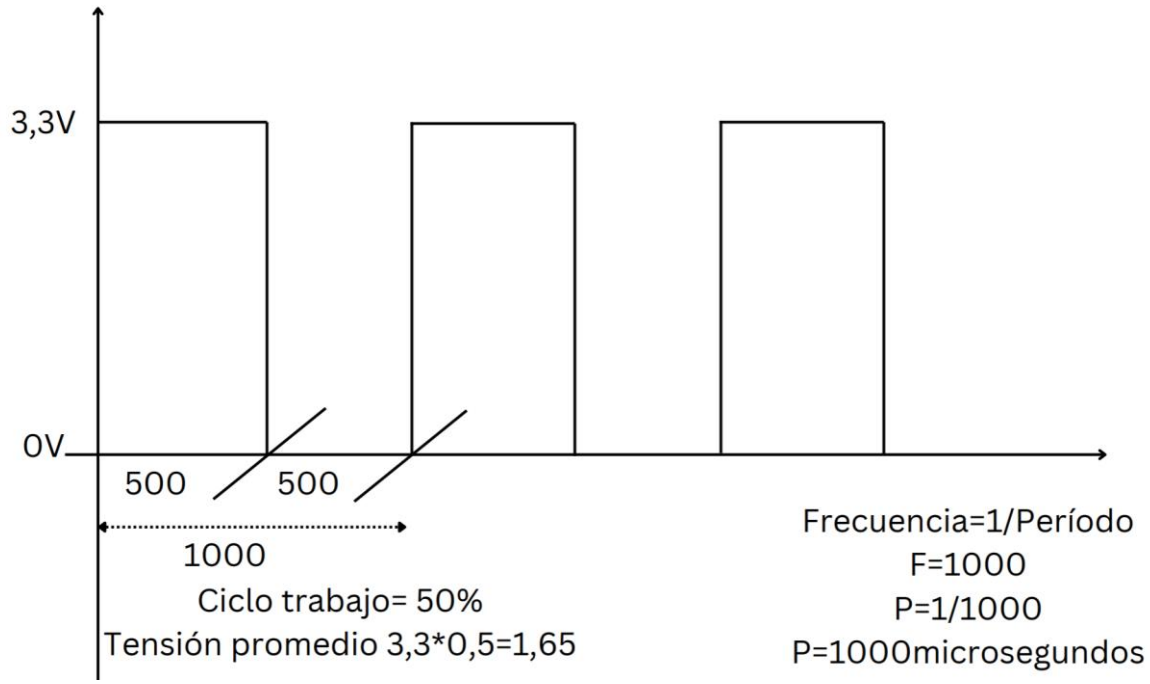
PWM consiste en alternar una señal digital entre alto (HIGH) y bajo (LOW) a una frecuencia específica. La duración de la señal en estado alto dentro de un ciclo se llama ciclo de trabajo (duty cycle), expresado en porcentaje (%).

- 100%: Señal siempre en HIGH.
- 50%: Señal en HIGH la mitad del tiempo y en LOW la otra mitad.
- 0%: Señal siempre en LOW.

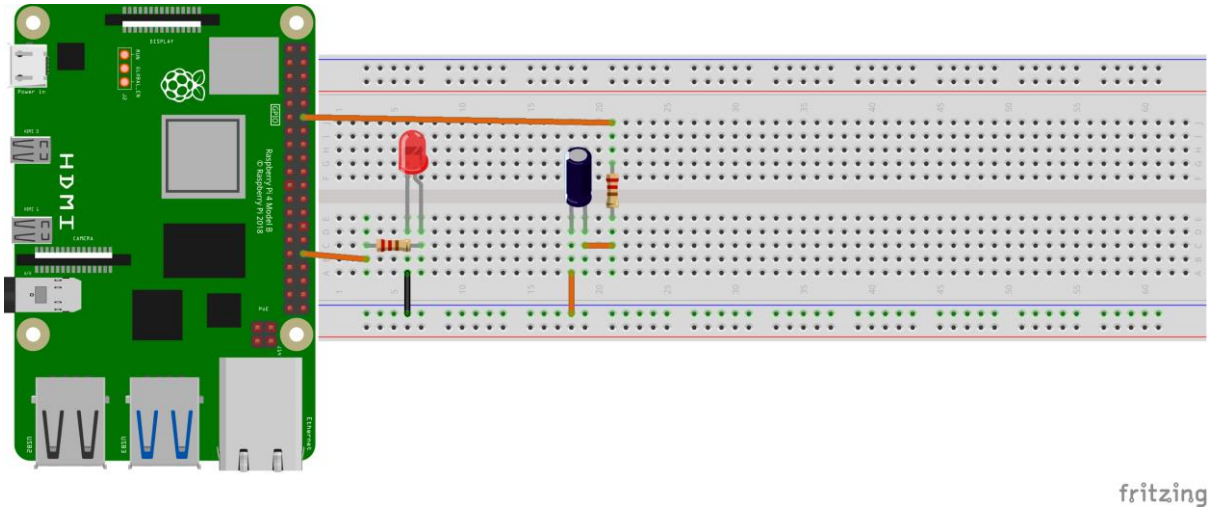
Aplicaciones del PWM:

- Ajustar el brillo de LEDs.
- Controlar la velocidad de motores.
- Generar señales de audio o pulsos en circuitos específicos.

**Al final del documento se incluyen códigos y esquemas.



Señal PWM



Conexión led condensador

En el apartado “scripts pyhton” código “pwm_a_analogico_rc.py” y

Práctica expansión GPIO

¿Qué es el protocolo I2C?

- Fue desarrollado por Philips Semiconductor en 1982.
- Es ampliamente usado en dispositivos electrónicos para comunicación entre componentes en un mismo sistema.
- Comparte relevancia con protocolos como SPI, aunque su uso está más enfocado en distancias cortas, típicamente dentro de un mismo hardware.

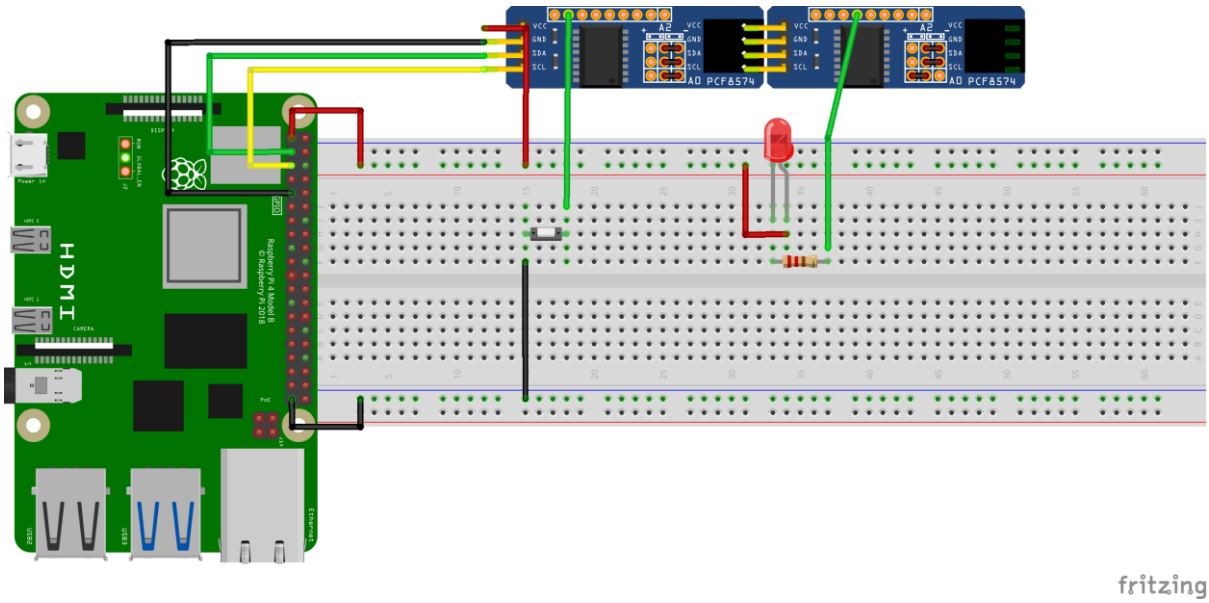
**Al final del documento se incluyen códigos y esquemas.

Características del I2C:

- Protocolo bidireccional que utiliza dos líneas:
  - SDA (Serial Data Line): Para datos.
  - SCL (Serial Clock Line): Para la señal de reloj.
- Adecuado para conectar múltiples dispositivos utilizando direcciones únicas para cada uno.
- Velocidad típica de comunicación de hasta 1 Mbps, ideal para aplicaciones donde la distancia no exceda 1 metro.

Aplicaciones Comunes:

- Comunicación entre sensores, microcontroladores y otros periféricos.
- Uso en dispositivos integrados para compartir datos y comandos.



Conexión de dos PCF8574

En el apartado “scripts pyhton” código “pCF8574.py”

Práctica entradas analógicas

Limitaciones de la Raspberry Pi:

- La Raspberry Pi no incluye salidas analógicas nativas.
- Se pueden aproximar señales analógicas usando PWM (Pulse Width Modulation) con un filtro de paso bajo compuesto por una resistencia y un condensador. Sin embargo, este método no es ideal si se aplica una carga al final del circuito.

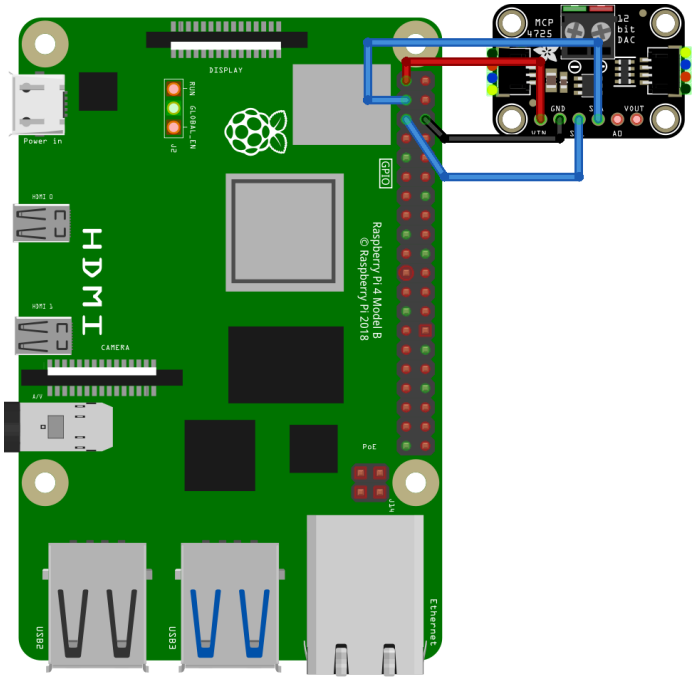
**Al final del documento se incluyen códigos y esquemas.

Uso del Integrado MCP4725:

- El MCP4725 es un DAC (Digital-to-Analog Converter) que permite generar salidas analógicas.
- Funciona con tensiones de 3.3V o 5V y ofrece una resolución de 12 bits.
- Divide el rango de salida (0-3.3V o 0-5V) en 4096 pasos, proporcionando alta precisión.

Ventajas del MCP4725:

- Ideal para generar señales analógicas precisas.
- Puede utilizarse en combinación con la Raspberry Pi para aplicaciones como control de motores, generación de formas de onda con dispositivos que requieren entradas analógicas.



Conexión Raspberry Pi con MCP4725

En el apartado “scripts pyhton” código “MCP4725.py”

Práctica salidas analógicas

Limitaciones de la Raspberry Pi:

- La Raspberry Pi no tiene entradas ni salidas analógicas nativas.
- Para habilitar esta funcionalidad, se utiliza un ADC (Conversor Analógico-Digital), específicamente el ADS1115.

**Al final del documento se incluyen códigos y esquemas.

Configuración del Hardware:

- ADS1115: Un integrado que permite convertir señales analógicas en digitales.
- Alimentación: Funciona con 3.3V y GND.
- Modo de entrada: La señal analógica se conecta a la entrada A0 del ADS1115.
- Potenciómetro: Utilizado para simular una entrada analógica.
- Genera un rango de voltaje entre 0 y 3.3V según la posición de su eje central.
- Comunicación: Utiliza el protocolo I2C para interactuar con la Raspberry Pi.

Características del ADS1115:

- Dirección I2C predeterminada: 0x48.
- Configuración de pines para expandir y manejar múltiples dispositivos en el mismo bus.

Introducción al Experimento:

- El objetivo es verificar la ecuación de la curva de carga de un condensador mediante medición y análisis.
- Se utiliza un condensador de 10 µF y dos resistencias de 1 kΩ y 10 kΩ.

Método del Experimento:

- **Carga del Condensador**:
  - Se activa un pin GPIO de la Raspberry Pi a 3.3V para cargar el condensador.
- **Monitorización**:
  - Los niveles de voltaje en el condensador se registran utilizando el módulo **ADS1115**.
- **Curva de Carga**:
  - El tiempo necesario para alcanzar el 63.2% del voltaje total (aproximadamente **2.09V**) se calcula para validar la constante de tiempo del circuito RC.

- También se mide el tiempo para alcanzar el 99% de la carga total, simulando una carga casi completa.

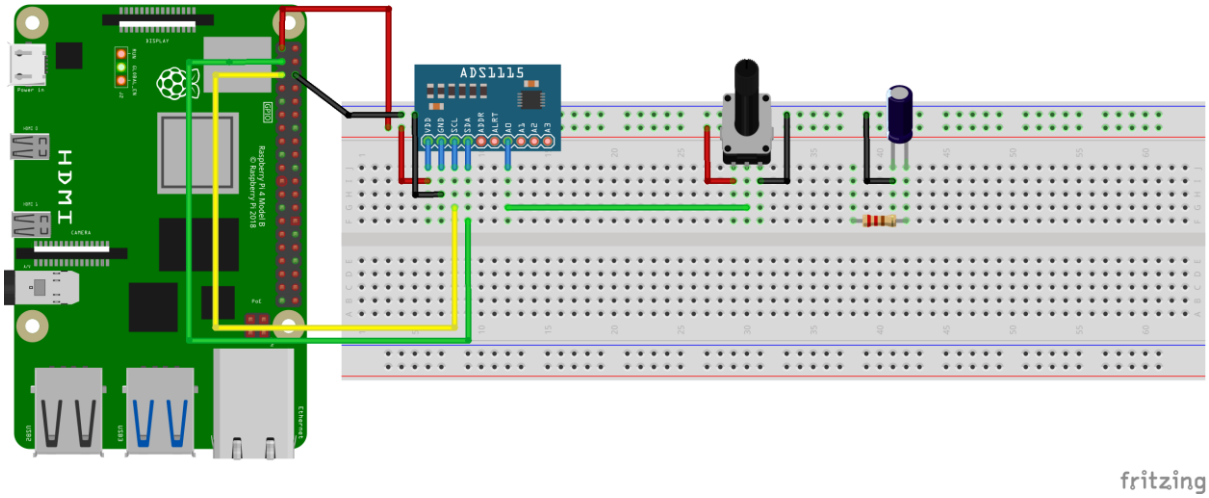
- **Fundamento Teórico**:
- La ecuación básica de carga de un condensador es:

$$V(t) = V_{max} \cdot (1 - e^{-t/RC})$$

Donde:

- Vmax : Voltaje máximo (3.3V en este caso).
- R: Resistencia en ohmios.
- C: Capacitancia en faradios.

**Al final del documento se incluyen códigos y esquemas.



Conexión Raspberry Pi con ADS1115 y potenciómetro

En el apartado “scripts pyhton” código “ads1115.py” y “ads_condensador.py”

Práctica pantalla LCD

Introducción a la Pantalla

- Se utiliza una pantalla de Adafruit compatible con la Raspberry Pi.
- Aunque es de tamaño reducido, permite mostrar información como:
- Dirección IP.
- Temperatura interna.
- Uso de CPU y memoria.

Funcionamiento y Aplicaciones:

- Los datos mostrados pueden rotar o alternarse mediante un menú.
- La interacción puede ampliarse añadiendo un pulsador, que ya se ha aprendido a implementar en sesiones anteriores, para cambiar entre los datos mostrados.

Preparación del Entorno:

- Instalación de librerías necesarias de Adafruit para manejar la pantalla.
- Consideraciones de seguridad, especialmente en sistemas configurados con restricciones (e.g., Raspberry Pi OS Lite).

```
import board import adafruit_ssd1306
#Configurar la pantalla
i2c = board.I2C() oled = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c)
#Mostrar texto en la pantalla
oled.fill(0)
oled.text("Direccion IP:", 0, 0)
oled.text("192.168.1.100", 0, 10)
 oled.show()
```

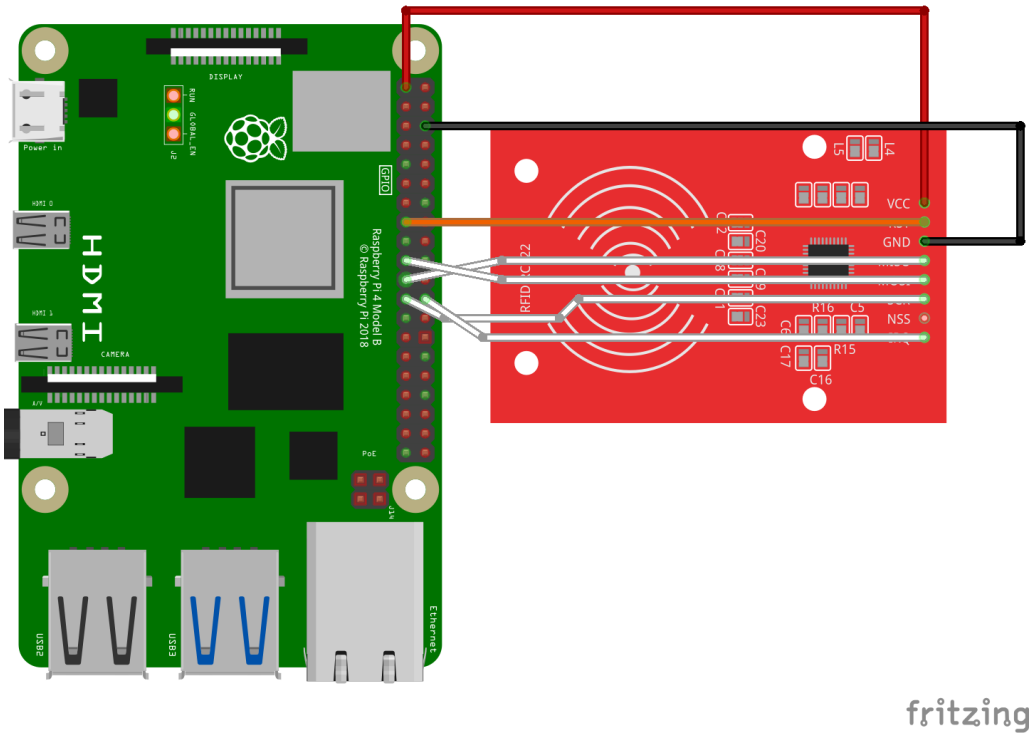
Práctica lector rfid

**Al final del documento se incluyen códigos y esquemas.

El protocolo SPI (Serial Peripheral Interface) permite conectar una serie de componentes, que en este caso actuarán como esclavos, para ampliar las funcionalidades del hardware de una Raspberry Pi.

Este protocolo utiliza la arquitectura maestro-esclavo. En este caso, la Raspberry Pi mantendrá el control de la comunicación. Sin embargo, lo que cambia no es solo el protocolo en sí, sino también la capa física, es decir, los puertos y el cableado utilizados. Este protocolo de comunicación en serie permite una transferencia rápida de datos, por lo que es común encontrarlo en módulos como Ethernet, tarjetas SD y otros dispositivos.

Trabajaremos con un módulo práctico de RFID. Este protocolo síncrono utiliza una señal de reloj compartida entre el maestro y los esclavos para sincronizar la comunicación. Además, exploraremos cómo programar y configurar este protocolo, incluyendo aspectos clave de su implementación física y lógica.



Raspberry Pi conectada a módulo RFID <<verificar en cada caso el pineado del modelo usado, puede no corresponderse con la imagen>>

En el apartado “scripts pyhton” código “rfid.py” y “comandos spi”

Práctica comunicación UART

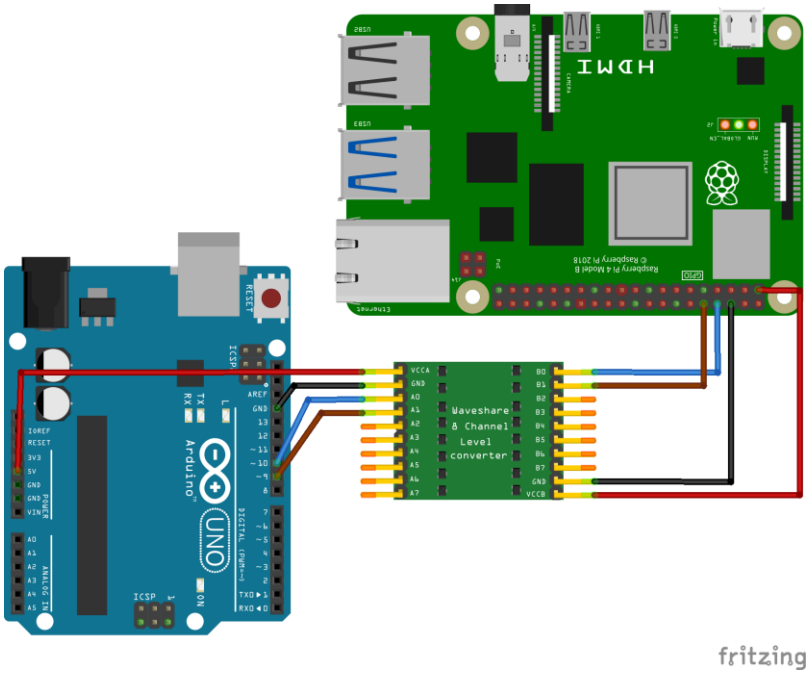
**Al final del documento se incluyen códigos y esquemas.

La comunicación UART (Universal Asynchronous Receiver-Transmitter), que también está presente en los GPIO de las Raspberry Pi. Este tipo de comunicación tiene diferencias significativas con las que hemos visto anteriormente. No utiliza una arquitectura maestro-esclavo ni es una comunicación síncrona.

En este caso, no existe un dispositivo dominante que controle la comunicación. Cada dispositivo puede actuar de manera independiente, transmitiendo datos en cualquier momento. Por ello, el dispositivo receptor debe estar constantemente a la escucha para recibir las tramas de datos cuando se transmitan.

A nivel de programación, esto requiere una mayor atención, ya que debemos gestionar correctamente la recepción y envío de datos. Sin embargo, veremos que estas particularidades pueden manejarse de forma eficiente.

La comunicación UART es susceptible a interferencias, pero sigue siendo ampliamente utilizada en dispositivos como módulos GPS, sensores y otros periféricos. En esta clase aprenderemos cómo implementarla y optimizarla en proyectos con Raspberry Pi, asegurando una comunicación estable y efectiva.



En el apartado “scripts pyhton” código “uart_read.py” y comandos “comandos uart” y “envio_datos_arduino”

Práctica protocolo OneWire

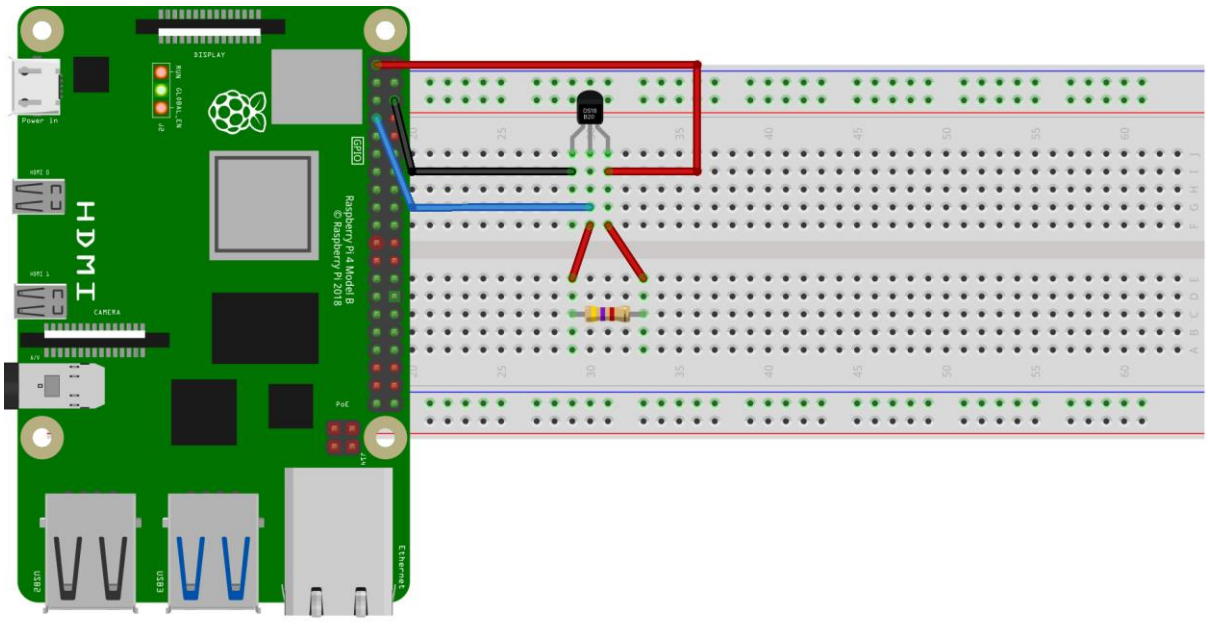
**Al final del documento se incluyen códigos y esquemas.

El Protocolo OneWire fue desarrollado por Dallas Semiconductor y se encuentra presente en diferentes sensores. Uno de los más conocidos es el DS18B20, un sensor de temperatura bastante robusto que incluso está disponible en formato de sonda, lo que permite medir líquidos, además de temperaturas ambientales.

Estos sensores son muy económicos, lo que permite conectar varios de ellos (por ejemplo, dos o tres) para calcular un promedio de las lecturas o detectar si alguna medición está muy desviada. Esto ayuda a asegurar la precisión y permite reemplazar fácilmente cualquier sensor defectuoso debido a su bajo costo.

El protocolo OneWire utiliza un único hilo para la transmisión y recepción de datos, lo que simplifica el cableado. Este diseño minimalista es ideal para proyectos con restricciones físicas o de presupuesto, manteniendo una comunicación eficiente y fiable.

Durante esta clase, profundizaremos en cómo funciona el protocolo OneWire, cómo configurar sensores como el DS18B20 en una Raspberry Pi, y cómo aprovechar al máximo sus características en proyectos prácticos.



En el apartado “scripts pyhton” código “one_wirte.py” y comandos “comandos one wire”