Práctica 1: Introducción. ADC.

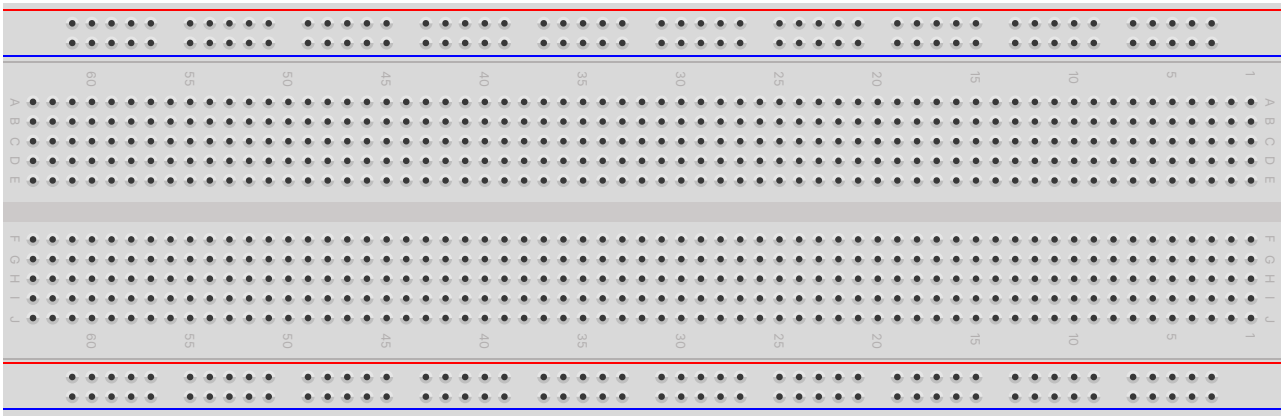
Adquisición y Procesamiento de Señal

Objetivo

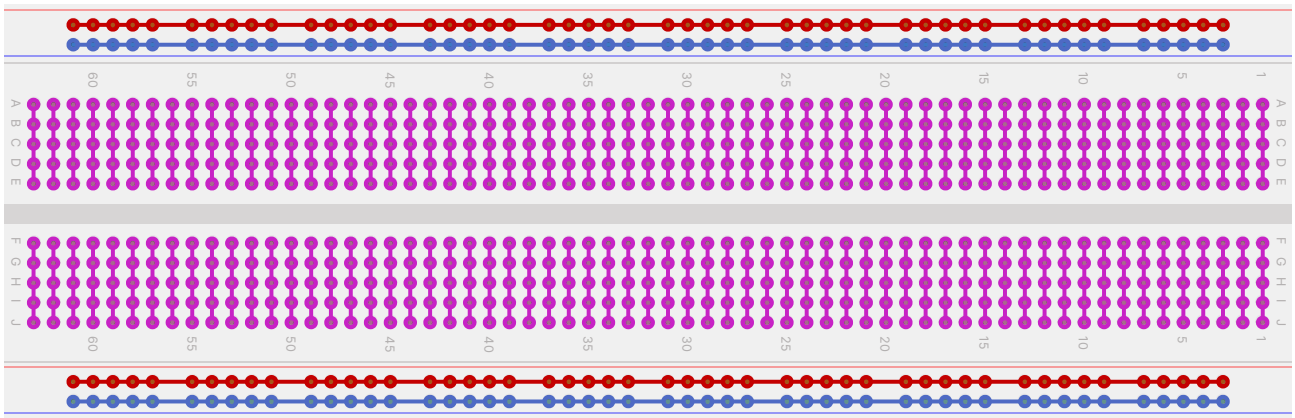
El objetivo de esta práctica es la introducción a la placa de pruebas y al microcontrolador ESP32, así como la comprensión del funcionamiento del conversor analógico-digital (ADC) del ESP32 para la captura de señales analógicas.

Placa de pruebas y ESP32

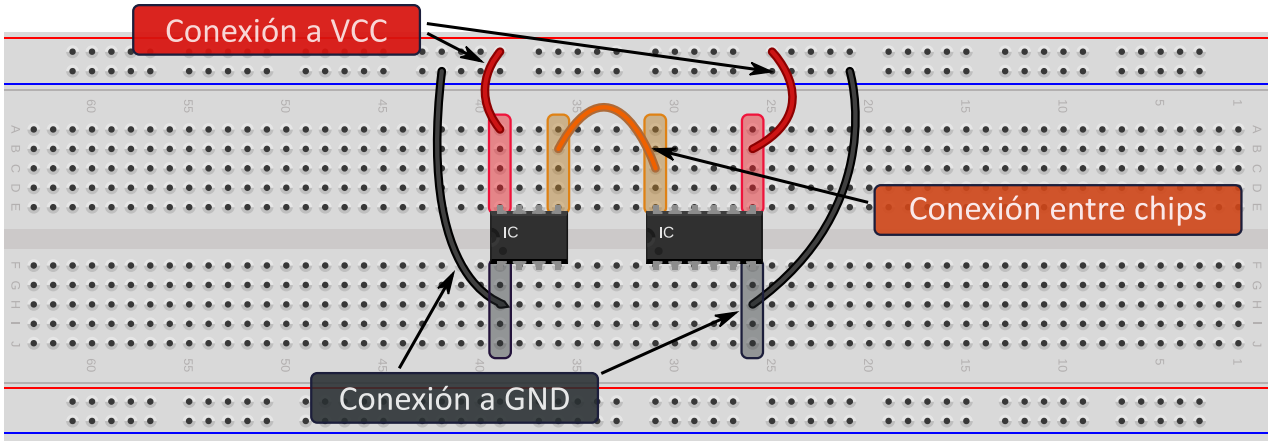
La **placa de pruebas** o **placa de inserción** (en inglés *breadboard* o *protoboard*) en la que está montado el ESP32 es un tablero con una serie de líneas internas que permiten la conexión de los diferentes elementos de un circuito mediante la inserción de cables o ***jumpers*** en los orificios.



Los orificios que recorren la placa por los laterales (entre las líneas rojas y azules) están conectados longitudinalmente, formando 4 líneas de 50 orificios. La roja suele usarse para el voltaje de **alimentación/VCC** (normalmente **5 V** o **3.3 V**) y la azul para el voltaje de **referencia o tierra/GND** (**0 V**). Los centrales están conectados transversalmente formando 2x63 líneas de 5 orificios:



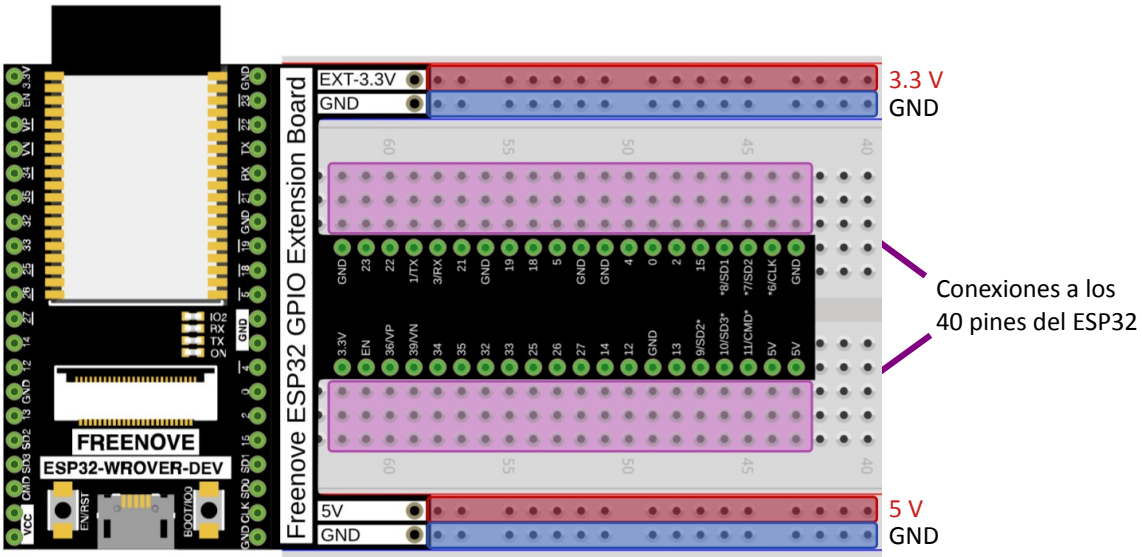
Los componentes se “pinchan” longitudinalmente en la placa de manera que cada pin queda insertado en una línea de 5 orificios diferente. Si el componente tiene dos hileras de pines (por ejemplo, un chip) se coloca en el centro. Esto permite su conexión a la alimentación (VCC y GND) y la conexión entre pines de diferentes chips:



En la placa ya está insertado el **microcontrolador ESP32**, sobre el que realizaremos las prácticas. El ESP32 se podría pinchar directamente en la placa de pruebas, pero debido a su anchura quedarían tapados todos los orificios en uno de sus laterales. En su lugar, el ESP32 está conectado a un adaptador, y éste está conectado a la placa. Además de permitir el montaje del ESP32, el adaptador también proporciona alimentación a las líneas longitudinales (3.3 V y GND a las superiores, 5 V y GND a las inferiores), facilitando la alimentación de otros componentes.

El ESP32 se alimenta a través del puerto Micro-USB. Al conectarse, arranca el firmware MicroPython (ya grabado en su memoria flash interna) y ejecuta el programa Python que tenga almacenado, si existe. Si no existe simplemente se queda a la espera.

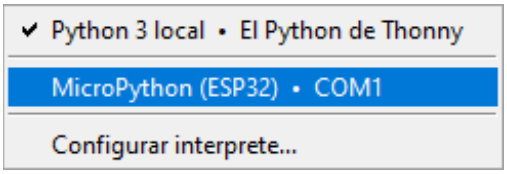
A ambos lados del puerto Micro-USB del ESP32 hay dos botones. El situado a la derecha (con el puerto USB hacia abajo), marcado **BOOT/IO0**, selecciona el *bootloader*, y no lo vamos a usar. El situado a la izquierda, marcado **EN/RST**, reinicia el ESP32, y se puede pulsar en cualquier momento (por ejemplo, cuando el ESP32 no responda).



Entorno de desarrollo

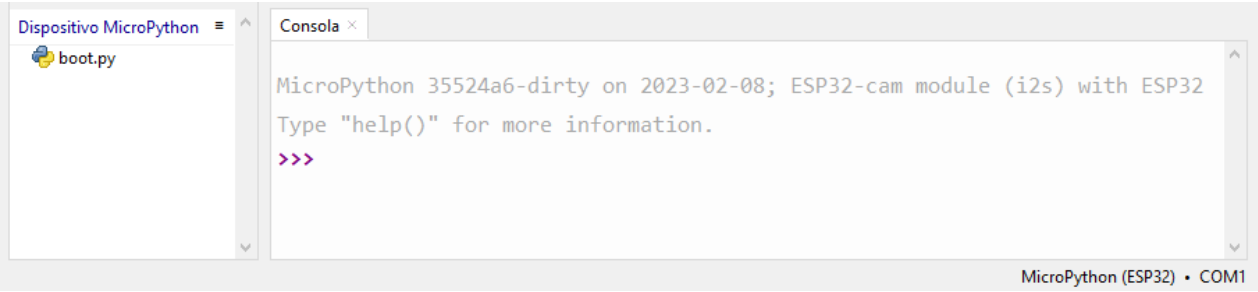
El IDE **Thonny** permite ejecutar programas Python de manera directa en microcontroladores con firmware MicroPython.

Conecta el ESP32 al ordenador a través del cable USB del kit. Si tienes un Windows reciente (10, 11) debería detectar automáticamente el chip. Si es así, en la esquina inferior derecha de la ventana de Thonny deberías poder seleccionar el entorno en el que se ejecuta el programa: en el Python de la máquina local o en el MicroPython del ESP32:



Si no aparece la opción *MicroPython (ESP32) · COM**n***, pulsa *Configurar interprete…* e intenta seleccionarlo en los desplegables de la ventana de diálogo que aparece. Si tampoco funciona, es posible que tengas que instalar el driver para el chip **CH340** (el ESP32 no soporta USB de forma nativa, así que para su comunicación con el ordenador utiliza un CH340 integrado que se encarga de la conversión USB ↔ UART, protocolo que sí es soportado de forma nativa por ESP32 y prácticamente la totalidad de microcontroladores). Puedes descargarlo de: https://www.wch-ic.com/downloads/CH341SER_EXE.html

Al seleccionar el ESP32, la consola (bajo el editor) cambiará al intérprete MicroPython del microcontrolador, indicando en gris la versión de MicroPython, la fecha de compilación y el hardware en que se está ejecutando. También aparecerá una nueva ventana en la parte inferior izquierda que muestra los ficheros presentes en el almacenamiento interno del ESP32:

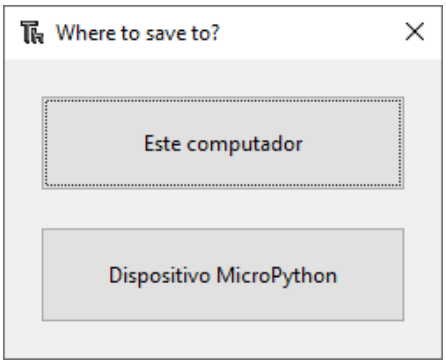


El *firmware* de MicroPython incluye por defecto un solo fichero, `boot.py`, que es ejecutado por el ESP32 cada vez que se reinicia. Por defecto solo tiene comentarios. No lo vamos a modificar; se suele utilizar para configurar opciones de MicroPython que no necesitamos cambiar.

En su lugar crearemos un nuevo fichero, `main.py`, que se ejecuta siempre a continuación de `boot.py`. Es posible escribir el programa directamente en `main.py`, pero es recomendable que este fichero solo contenga una línea importando el módulo que realmente implementa el programa; de esta forma podemos cambiar de un programa almacenado en el ESP32 a otro diferente simplemente editando esa línea, en lugar de renombrar el fichero con el programa deseado a `main.py`. Crea un nuevo fichero (**ctrl+n** o *Nuevo programa* en el menú) y escribe en él la línea:

```
1 import helloworld
```

Guárdalo (**ctrl+s**) como `main.py`. Al guardar el fichero, Thonny te ofrecerá la opción de guardarlo en la máquina local o en el ESP32:



Selecciona *Dispositivo MicroPython*. (**Nota:** el fichero solo se podrá guardar si el ESP32 está inactivo, es decir, mostrando el *prompt* `>>>` de Python en la consola. Si está ejecutando un programa, debes interrumpirlo antes con **ctrl+c**, **ctrl+F2** o con el **botón Detener en la barra de botones**).

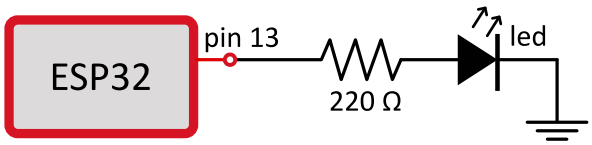
A continuación crea el fichero `helloworld.py` de la misma forma que el anterior. Escribe el siguiente código para un programa que imprime la cadena "hello world" de forma indefinida, una vez por segundo:

```
1  import time
2
3  while True:
4      print('hello world')
5      time.sleep(1)
```

La cadena debería aparecer en la consola bajo el editor cuando se ejecuta el programa (**F5** o **botón Ejecutar en la barra de botones**) o cuando se reinicia el ESP32 (**ctrl+d**, o **botón RST del ESP32**):

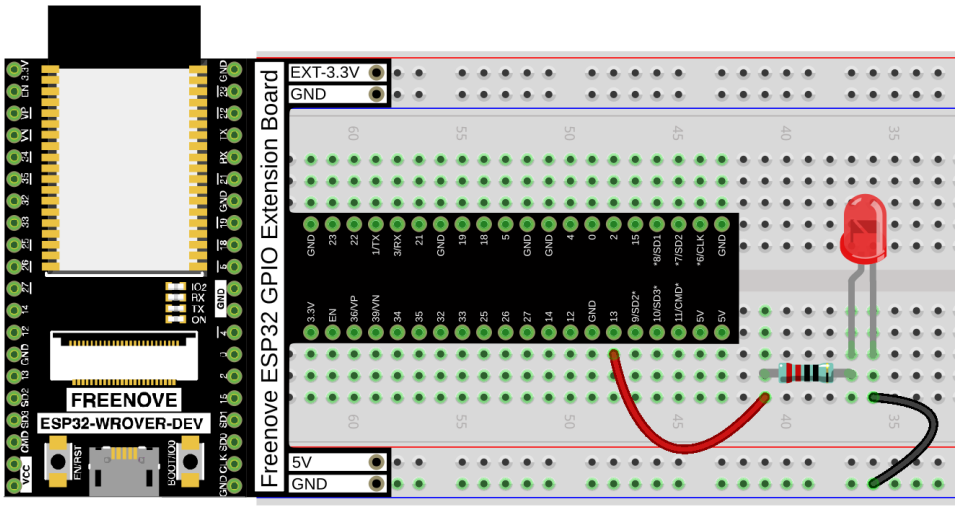
Tarea 1: Salidas digitales

Un microcontrolador no es muy útil si no se conecta a nada. Vamos a conectar el ESP32 a un led para mostrar la capacidad de generar **salidas digitales** arbitrarias en sus pines **GPIO (*General-Purpose Input/Output*)**:



Si el pin tiene un nivel de salida alto (3.3 V, que es el voltaje de alimentación del ESP32 y por lo tanto el voltaje que pueden tener los pines de salida), circulará corriente atravesando la resistencia y el led, que emitirá luz. Si el pin tiene un nivel de salida bajo (0 V) no habrá corriente y el led estará apagado.

Realiza el siguiente montaje, correspondiente al anterior circuito:



Conecta la resistencia de 220 Ω (código de colores rojo-rojo-negro-negro ≡ **2**-**2**-**0**-**0** ⇒ **22**$\mathbf{0} \times 10^{0}$ = 220) y el led como se ve en la imagen: el ánodo del led (**patilla más larga**) a una de las patillas de la resistencia y el cátodo (**patilla más corta**) a GND. Conecta el otro pin de la resistencia al **pin 13** del microcontrolador.

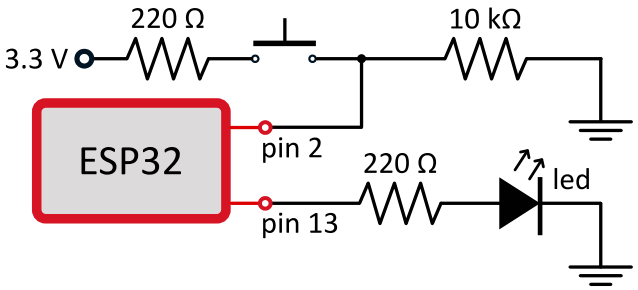
Modifica el código de `helloworld.py` para cambiar el valor del pin en cada iteración del bucle (0 ⇒ 0 V, 1 ⇒ 3.3 V) de manera que el led se encienda y se apague una vez por segundo. Consulta la documentación en https://docs.micropython.org/en/latest/library/machine.Pin.html. Los pasos necesarios son:

- Importar la clase `Pin` del módulo `machine`.
- Crear un objeto `pin_led` de tipo `Pin` asociado al pin 13 y configurado como pin de salida (`Pin.OUT`).
- Modificar el valor del pin en cada iteración (`pin_led.value(v)`, donde **`v`** puede tomar un valor de 0 o 1).

Tarea 2: Entradas digitales

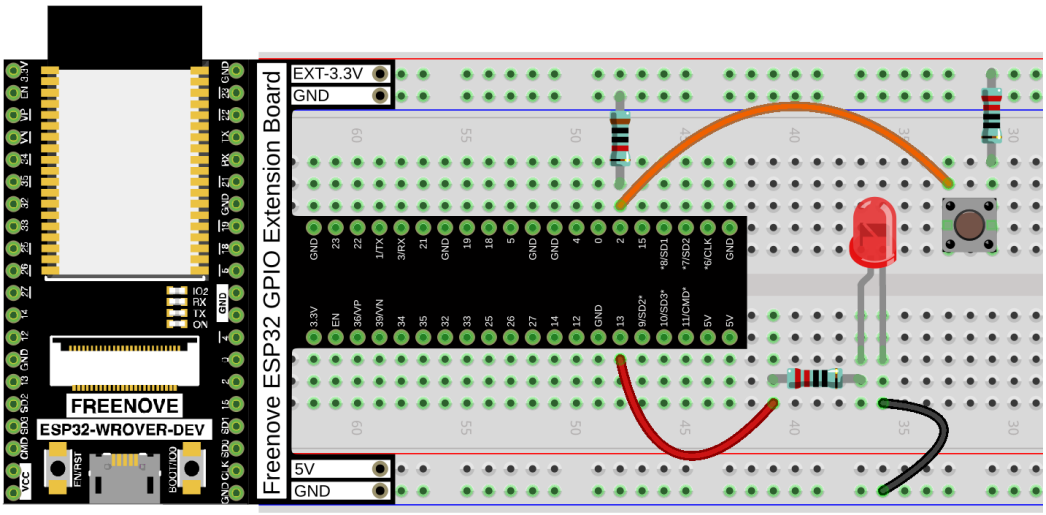
Los pines GPIO, además de ser capaces de producir un voltaje, también son capaces de "leer" un voltaje aplicado sobre ellos. Este voltaje será interpretado como un 1 cuando el voltaje en el pin sea cercano al de alimentación (3.3 V) o como un 0 cuando el voltaje sea cercano al de tierra o GND (0 V).

Vamos a modificar el circuito anterior añadiendo un botón (pulsador momentáneo) que utilizaremos para activar el parpadeo del led:



Cuando el botón no está pulsado, el voltaje en el pin 2 será de 0 V (a través de la resistencia de 10 kΩ): el ESP32 leerá un 0. Cuando se pulsa el botón, el pin se conecta también a 3.3 V a través del botón y de la resistencia de 220 Ω. Como 220 Ω « 10 kΩ, el camino hacia 3.3 V es de mucha menor resistencia que hacia 0 V y el voltaje del pin estará mucho más cerca de 3.3 V que de 0 V: el ESP32 leerá un 1.

El siguiente montaje se corresponde con el anterior circuito:



Conecta un pin del botón a una resistencia de 220 Ω y otro al pin 2 del microcontrolador. Conecta la otra patilla de la resistencia de 220 Ω a 3.3 V. Conecta una resistencia de 10 kΩ (código de colores marrón-negro-negro-rojo ≡ **1**-**0**-**0**-**2** ⇒ **100** × 10^{2} = 10000) entre el pin 2 y GND. (**NOTA**: en las resistencias hay una **quinta banda de color marrón** ligeramente más separada que indica la tolerancia y que puede confundirse con el valor de la resistencia. Si la última banda es roja, la resistencia es la de 220 Ω y está siendo leída al revés.)

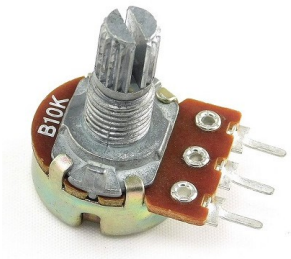
Modifica el programa anterior para que al pulsar el botón el led comience a parpadear una vez por segundo, y cuando se vuelva a pulsar deje de parpadear. Para ello debes:

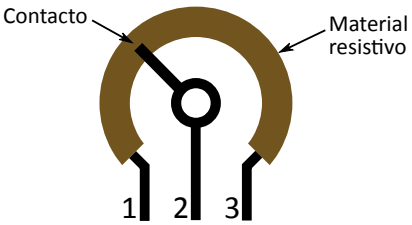
- Crear un objeto `pin_boton` asociado al pin 2 configurado como pin de entrada (`Pin.IN`).
- En cada iteración, leer el valor del pin (`v = pin_boton.value()`) para activar o desactivar el parpadeo del led.

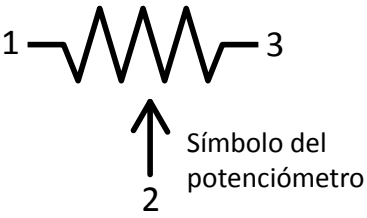
Tarea 3: ADC: potenciómetro

Además de como salidas digitales o entradas digitales, algunos pines GPIO del ESP32 pueden funcionar como **entradas analógicas**, leyendo voltajes arbitrarios gracias al **conversor analógico-digital** (**ADC**) integrado en el ESP32. Esto nos permitirá leer voltajes suministrados por **sensores analógicos,** como por ejemplo fotoresistores o termistores, y capturar así **señales.**

Para demostrar el funcionamiento del ADC utilizaremos en primer lugar un potenciómetro, un dispositivo muy simple que consiste en una resistencia *R* entre dos pines (1 y 3) sobre la que se desliza un brazo conectado a otro pin (2):

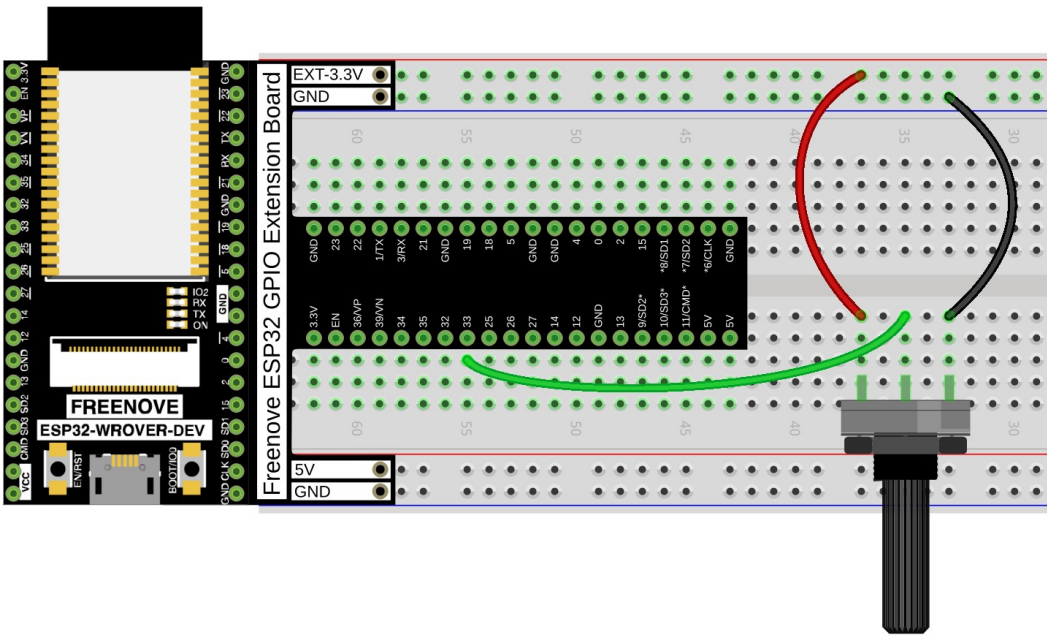




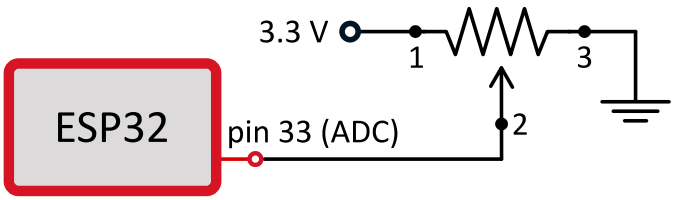


La posición del brazo determina la resistencia entre los pines 1 y 2 (R_{12}) y la resistencia entre los pines 2 y 3 (R_{23}). Girando en sentido horario aumenta R_{12} y disminuye R_{23}, y en sentido antihorario ocurre lo contrario. Se cumple que $R_{12} + R_{23} = R$ (constante). El valor de *R* depende del modelo del potenciómetro. En la foto se puede ver que el valor de *R* es de 10 kΩ, el mismo que el del potenciómetro del kit de prácticas.

Conecta el potenciómetro del kit en la placa del microcontrolador de la siguiente forma (¡con cuidado de no doblar los pines!):



El pin izquierdo (1) del potenciómetro se conecta a la línea de alimentación, de 3.3 V. El pin derecho (3) a tierra (GND), es decir, 0 V. El pin central (2), al pin 33 del microcontrolador, que actuará como entrada analógica:



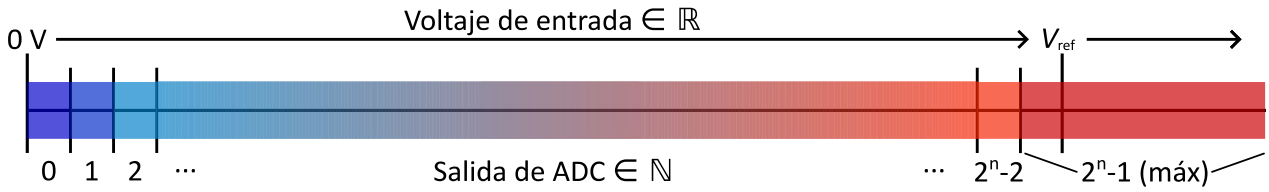
El microcontrolador medirá un voltaje entre 0 V y 3.3 V, dependiendo de la posición del potenciómetro. Si el potenciómetro se gira completamente en sentido antihorario, el pin 2 del potenciómetro (y por lo tanto el pin 33 del microcontrolador) se conectará directamente a 3.3 V ($R_{12} = 0$). Si se gira completamente en sentido horario, se conectará a 0 V ($R_{23} = 0$). En una posición intermedia, el voltaje en el pin 33 será un valor que vendrá dado por la fórmula del **divisor de tensión**:

$$V_{\text{pin}} = V_{cc}\frac{R_{23}}{R_{12} + R_{23}} = 3.3\,\text{V}\frac{R_{23}}{10\,\text{k}\Omega}$$

El ADC no calcula directamente el voltaje, sino que lo convierte a un valor **entero** entre 0 y $2^n - 1$, donde **n** es el número de bits o **resolución** del ADC. Al voltaje 0 le asigna el valor 0 y al **voltaje de referencia** V_{ref} le asigna el valor máximo, $2^n - 1$. A un voltaje intermedio le asignaría:

$$ADC_{val} = \left\lfloor 2^n \frac{V}{V_{\text{ref}}} \right\rfloor \quad (1)$$

Un ADC no puede medir voltajes menores que 0 ni mayores o iguales que el de referencia. A voltajes menores que 0 les asigna el 0 y a mayores o iguales que el de referencia les asigna $2^n - 1$:



El siguiente código lee el valor del pin 2 del potenciómetro cada 0.1 s (**frecuencia de muestreo** de 10 Hz) e imprime por pantalla el valor entero del ADC:

```
import time
from machine import Pin, ADC

adc = ADC(Pin(33))  # configura el ADC para que lea del pin 33
adc.atten(ADC.ATTN_11DB)  # configura el valor de referencia a 3.3V
while True:
    val = adc.read()
    print(val)
    time.sleep(0.1)
```

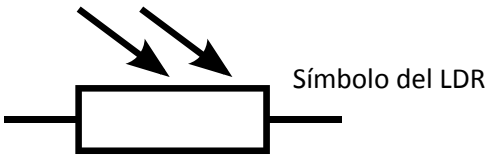
Completa el código para que además muestre el **voltaje** al que corresponde el valor medido según la ecuación (1), teniendo en cuenta que $V_{\text{ref}} = 3.3$ V. Comprueba que al girar el potenciómetro el valor cambia de forma acorde. Imprime ambos valores en la **misma línea**.

Preguntas

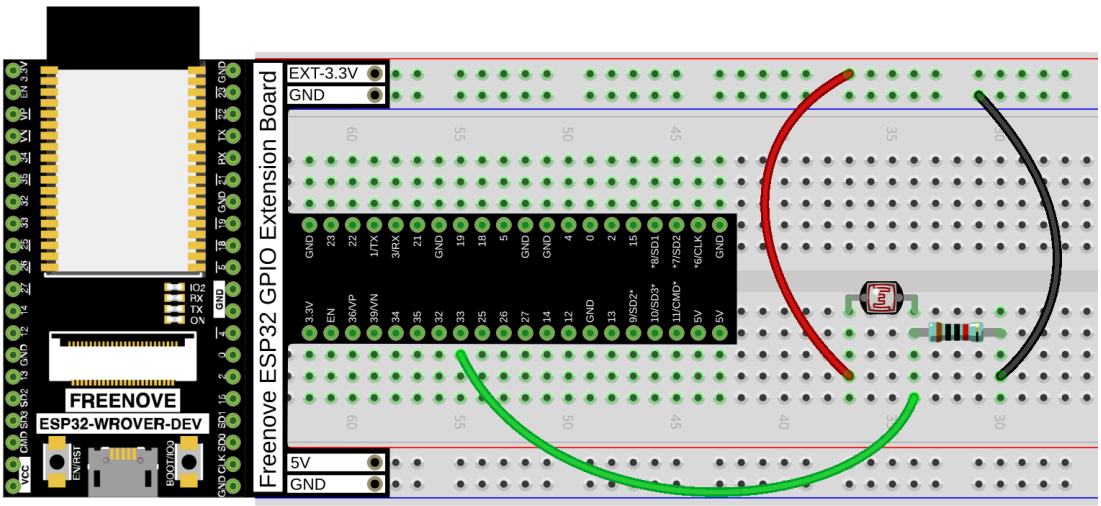
1. Gira el potenciómetro completamente sentido antihorario. ¿Qué valor entero obtiene el ADC? ¿Por qué?
2. A la vista del resultado, ¿qué resolución (en bits) tiene el ADC del ESP32? ¿Por qué? ¿Qué valor entero mostraría como máximo si tuviese una resolución de 8 bits?
3. Cambia la línea 5 para usar una **referencia de 2 V.** Para ello sustituye **ATTN_11DB** por **ATTN_6DB**. **Ajusta el cálculo del voltaje de forma acorde**. ¿En qué posición del potenciómetro llega ahora al máximo? ¿Qué valor de voltaje se obtiene con el potenciómetro girado completamente en sentido antihorario? ¿Qué voltaje tiene el pin en esa posición?

Tarea 4: ADC: fotoresistor

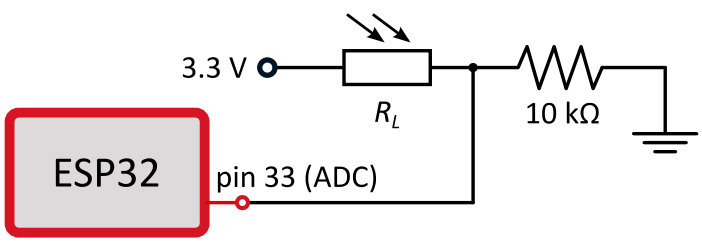
Un fotoresistor o **LDR** (***l**ight-**d**ependent **r**esistor*) es una resistencia cuyo valor depende de la luz que incide sobre él: cuanto **mayor es la luz, menor es la resistencia**. Se puede utilizar como **sensor de luminosidad**.



Conecta el LDR de la siguiente forma, utilizando la resistencia de 10 kΩ del kit (código de colores: marrón-negro-negro-rojo):



De esta forma estamos construyendo un divisor de tensión similar al que teníamos con el potenciómetro del apartado anterior:



La fórmula es por lo tanto la misma que (1), sustituyendo R_{12} por el LDR, con resistencia variable R_L que depende de la luz incidente, y R_{23} por la resistencia fija de 10 kΩ:

$$V_{\text{pin}} = 3.3\,\text{V}\frac{10\,\text{k}\Omega}{R_L + 10\,\text{k}\Omega} \qquad (2)$$

Si **aumenta** la luz que incide sobre el LDR, su **resistencia R_L disminuye**, y el voltaje en el pin 33 sube. Si la luz **disminuye**, la **resistencia R_L aumenta**, y el voltaje en el pin 33 baja.

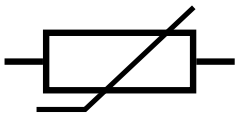
Modifica el código Python del apartado anterior para que muestre, además de la lectura del ADC y el voltaje, **el valor de R_L** calculado usando la ecuación (2) a partir del voltaje, de nuevo en la **misma línea**. Comprueba que al tapar el LDR el valor de resistencia aumenta, y que al enfocarlo hacia una fuente de luz la resistencia disminuye.

Preguntas

1. ¿Cuál es el valor del ADC con la luz ambiente? ¿A qué **rango de valores** de R_L corresponde?
2. ¿Cuál es el valor del ADC al taparlo con la mano? ¿A qué **rango de valores** de R_L corresponde?
3. ¿Cuál es el máximo valor de resistencia que puedes medir con este montaje? ¿Y el mínimo?

Tarea 5: ADC: termistor

Para el último montaje usaremos un termistor, que es una resistencia cuyo valor depende de la temperatura (a **mayor temperatura, menor resistencia**). Se utiliza como **sensor de temperatura**.



Símbolo del termistor

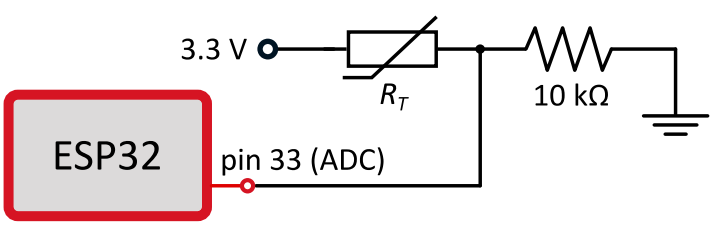
La resistencia de un termistor R_T a una temperatura T se puede aproximar a partir del parámetro β del termistor (unidad: Kelvin) y de un valor de resistencia R_0 conocido a una temperatura T_0 que sea cercana a la temperatura de medida, según la fórmula:

$$R_T = R_0 e^{\beta\left(\frac{1}{T}-\frac{1}{T_0}\right)}$$

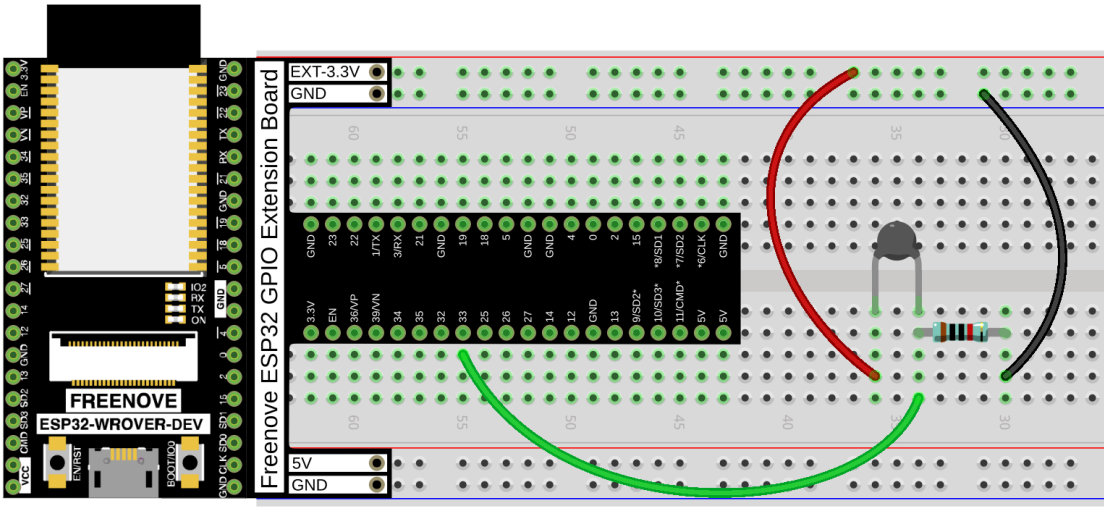
El termistor del kit tiene un parámetro β de 3950 K y una resistencia R_0 = 10 kΩ a una T_0 = 25 °C = 298.15 K, así que la relación entre la temperatura T (en Kelvin) y la resistencia R_T es:

$$R_T = 10\,\text{k}\Omega \cdot e^{3950\,\text{K}\left(\frac{1}{T}-\frac{1}{298.15\,\text{K}}\right)} \Rightarrow \frac{1}{T} = \frac{1}{298.15\,\text{K}} + \frac{\log(R_T/10\,\text{k}\Omega)}{3950\,\text{K}} \qquad (3)$$

Para realizar el montaje simplemente quita el fotoresistor y pon el termistor en su lugar:



Es decir:



La fórmula del voltaje en el pin es la misma que en caso anterior, con R_T en lugar de R_L:

$$V_{\text{pin}} = 3.3\,\text{V}\frac{10\,\text{k}\Omega}{R_T + 10\,\text{k}\Omega}$$

Completa el código anterior para que muestre (en la misma línea que los otros valores) **la temperatura en grados centígrados**, estimada a partir de R_T utilizando la ecuación (3). Comprueba que la temperatura medida se aproxima a la ambiente y que al calentar el termistor (por ejemplo echando aliento sobre él) la temperatura sube rápidamente.

Preguntas

1. Mide la temperatura con una frecuencia de muestreo de 20 Hz. ¿Permanece estable el valor medido? ¿A qué crees que se debe?

2. Calcula la **media móvil de las últimas 50 muestras** de temperatura. Ten en cuenta que, en una media móvil, cada nueva muestra implica recalcular la media (pues se debe incluir la nueva muestra y eliminar la más antigua). Muestra por pantalla la media móvil en la misma linea que los otros valores.

   Captura 200 muestras de la señal de temperatura "cruda" y de la media móvil y haz una **gráfica** usando Matplotlib. Para ello puedes copiar y pegar la salida de la consola de Thonny a un fichero de texto `datos.txt` en tu ordenador (dejando únicamente los dos valores a representar, es decir, crear un fichero con 200 filas y 2 columnas) y, en el Python de la **máquina local**, leer el fichero a un array con `numpy.loadtxt` y representar el array con `matplotlib.pyplot.plot`. ¿Qué efectos provoca realizar la media móvil?