

Programando Raspberry Pi Pico + HuskyLens v1.99.1

1. Programando la Pico

- Instalación de Thonny
- Primeros programas con Python en Thonny
- Instalación del firmware micropython en la Pico

2. Electrónica con la Pico

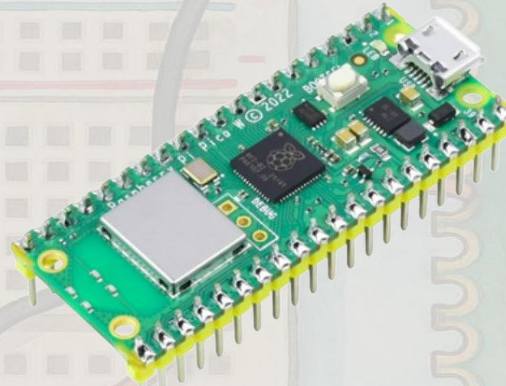
- Encendiendo LEDs
- Pulsadores
- Controlando el brillo
- ADC - Resistencia variable

3. HuskyLens

- Usando la HuskyLens
- Conectando HuskyLens + Pico
- Controlando la HuskyLens desde la Pico

Apéndice: micro:bit


- Conectando la HuskyLens a la micro:bit



by [@javacasm](https://twitter.com/javacasm)

¿Qué debe llevar un buen kit de electrónica?

1. Raspberry Pi Pico W
2. Cables Dupont de colores M-M y H-H
3. Mini Breadboard, mejor si es la [PicoBB](#)
4. Resistencias de distintos valores

- a. 220 Ω para LEDs 
- b. Otros valores

5. Actuadores - Dispositivos de salida:

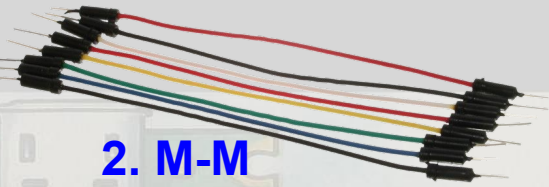
- a. LEDs de diferentes colores
- b. LED RGB
- c. Neopixels
- d. Relé

6. Sensores - Dispositivos de entrada:

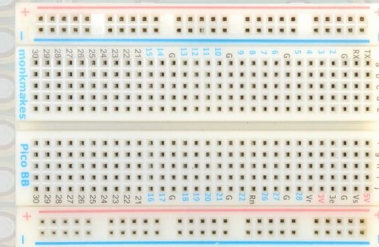
- a. Pulsadores
- b. Potenciómetro o resistencia variable
- c. Fotorresistencias LDR
- d. Sensor de temperatura



1.



2. M-M



3.

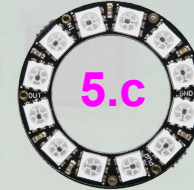


2. H-H

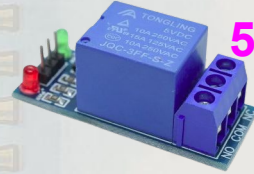


5.a

5.b



5.c



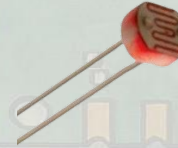
5.d



6.a



6.b



6.c



6.d



Raspberry Pi Pico W

La Raspberry Pi Pico es un microcontrolador diseñado para proyectos de computación física, tareas específicas y aplicaciones de bajo nivel, como controlar sensores, motores, LEDs u otros dispositivos electrónicos

Es ideal para principiantes y expertos que quieran aprender programación y electrónica

Aplicaciones:

La Raspberry Pi Pico es ideal para:

- Proyectos de IoT (especialmente con Pico W).
- Control de dispositivos como sensores, motores, pantallas o LEDs.
- Aprendizaje de programación y electrónica.
- Prototipado de productos electrónicos.
- Proyectos de robótica, automatización del hogar y sistemas embebidos.



Características de la Raspberry Pi Pico W

- **Microcontrolador RP2040:** Dual-core ARM Cortex-M0+ hasta **133 MHz**.
- Memoria: **264 KB de SRAM** y **2 MB de memoria flash** para almacenamiento de código y datos.
- **26 pines GPIO multifunción**, incluyendo
 - 3 entradas analógicas (ADC de 12 bits, 500 ksps).
 - Protocolos y buses: UART, I2C y SPI
 - 16 × canales PWM controlables.
- **8 máquinas de estado de E/S programable (PIO)**, para crear interfaces personalizadas en software y hardware.
- **USB 1.1**
- **Wi-Fi 802.11n** de 2.4 GHz.
- **Bluetooth 5.2**
- [También hay una Pico 2 W](#)

[Diferencias entre Raspberry Pi y Raspberry Pico](#)

[Más detalles técnicos](#)

La Raspberry Pi Pico W

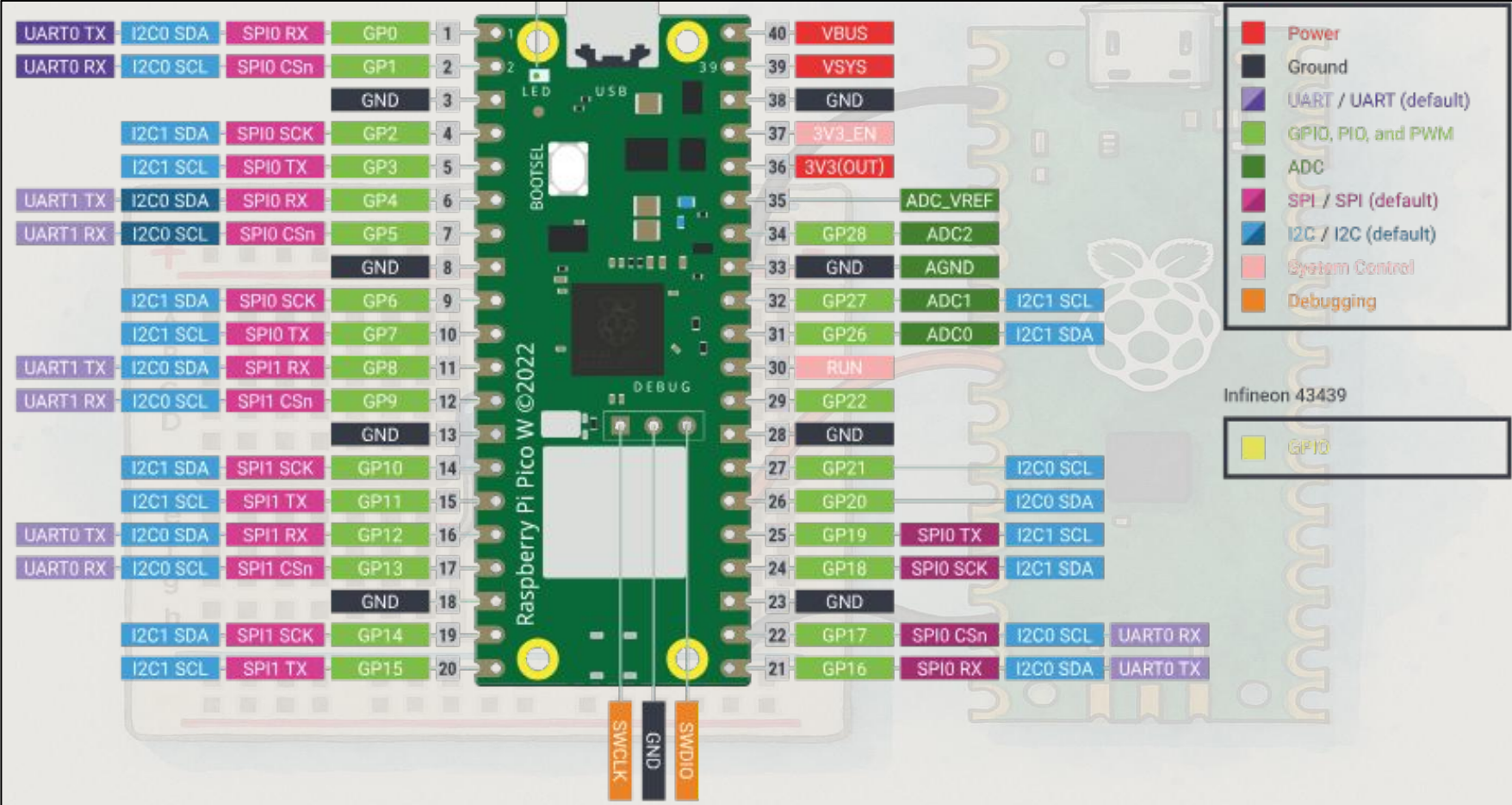
La placa Raspberry Pi Pico W:

- ¡¡Etiquetas en la parte posterior!!.. Fíjate bien cuando hagas los montajes
- **GND:** Tierra
- **GPIO:** Patillas de propósito general
 - Entradas/Salidas digitales: LEDs y botones. Usaremos GPIO14 y GPIO15
 - Entradas analógicas: Sensores. Usaremos GPIO26.
 - Para comunicaciones I2C usaremos GPIO06 y GPIO07



La Raspberry Pi Pico W: Expanded edition

[Imagen ampliada](#)





Instalación de Thonny

Thonny:

Para dar órdenes se necesita un entorno para programar. Thonny es un software sencillo para programar con **lenguaje Python**.

1. Descarga la versión de Thonny (<https://thonny.org/>) para tu sistema operativo.
2. Instálalo (no necesita permisos de administrador)
3. Selecciona el idioma que quieras y la configuración "Standard"

Thonny

Download version **4.1.7** for
Windows • Mac • Linux

Official downloads for Windows

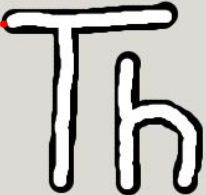
1. **Installer with 64-bit Python 3.10**, requires 64-bit Windows 8.1 / 10 / 11
[thonny-4.1.7.exe \(21 MB\)](#)

Installer with 32-bit Python 3.8, suitable for all Windows versions since 7
[thonny-py38-4.1.7.exe \(20 MB\)](#)

Portable variant with 64-bit Python 3.10
[thonny-4.1.7-windows-portable.zip \(31 MB\)](#)

Portable variant with 32-bit Python 3.8
[thonny-py38-4.1.7-windows-portable.zip \(29 MB\)](#)

Re-using an existing Python installation (for advanced users)
`pip install thonny`

3. 

Language:

Initial settings:



Hola Mundo

1. Consola: permite ejecutar comandos interactivamente, el comando se ejecuta tras pulsar la tecla ENTER.

>>> Es el prompt, que nos indica que python espera nuestras órdenes.

2. Editor: escribimos el programa que se ejecutará completo.

saludo.py

```
1 # Programa Saludo
2 ciudad = input('¿Dónde tienes tu clase hoy? ')
3 print('Hola', ciudad)
```

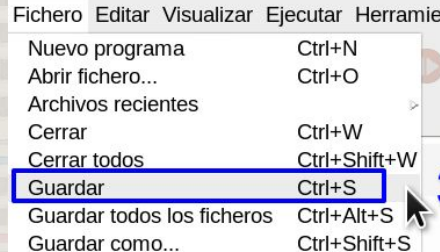
3. Guarda tu fichero



4. Ejecútalo



6. Haz que te pida tu nombre



Solución





Instalación de Micropython en la Pico

Thonny:

Vamos a instalar el firmware de Micropython en la Pico

1. Conecta la Pico W al ordenador mediante un cable USB.
2. Aparecerá una unidad (como un pendrive) RPI-RP2
3. Abre Thonny, ve a menú "Ejecutar", pestaña "Intérprete":

3.1 Selecciona "MicroPython (Raspberry Pi Pico)".

3.2 Pulsa en "Instala o actualiza Micropython".

3.3 Selecciona las características de tu Pico

3.4 Pulsa Instalar

Ya puedes seleccionar tu puerto para usar tu Pico.

Verás la versión y el prompt >>>

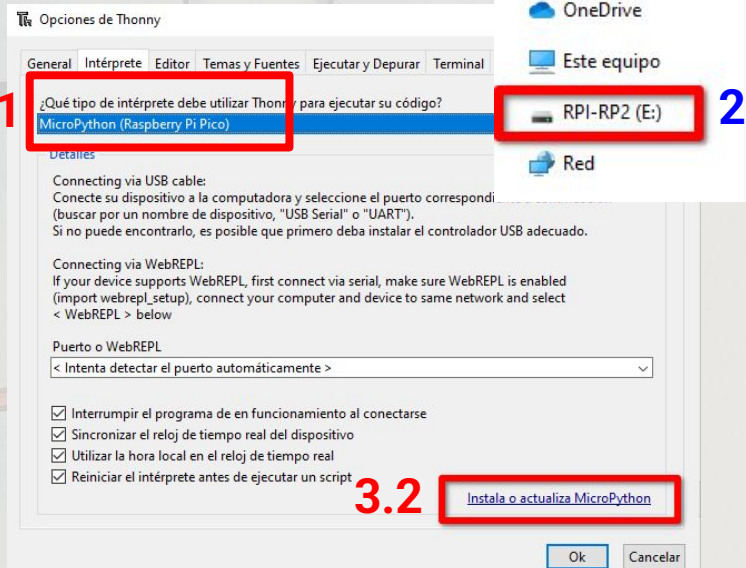
MPY: soft reboot

MicroPython v1.25.0 on 2025-04-15; Raspberry Pi Pico W with RP2040

Type "help()" for more information.

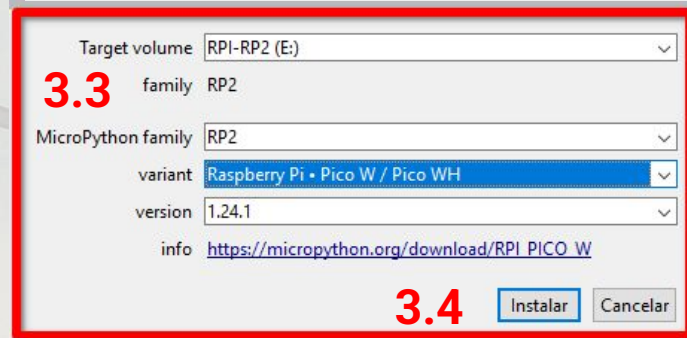
>>>

3.1



2

3.2



3.4



Thonny: entorno de programación

Thonny:

Usaremos la aplicación Thonny para programar nuestra Pico con **micropython**.

1. Conecta la Pico W al ordenador mediante un cable USB.
2. Abre Thonny, ve a "Ejecutar", pestaña "Intérprete":

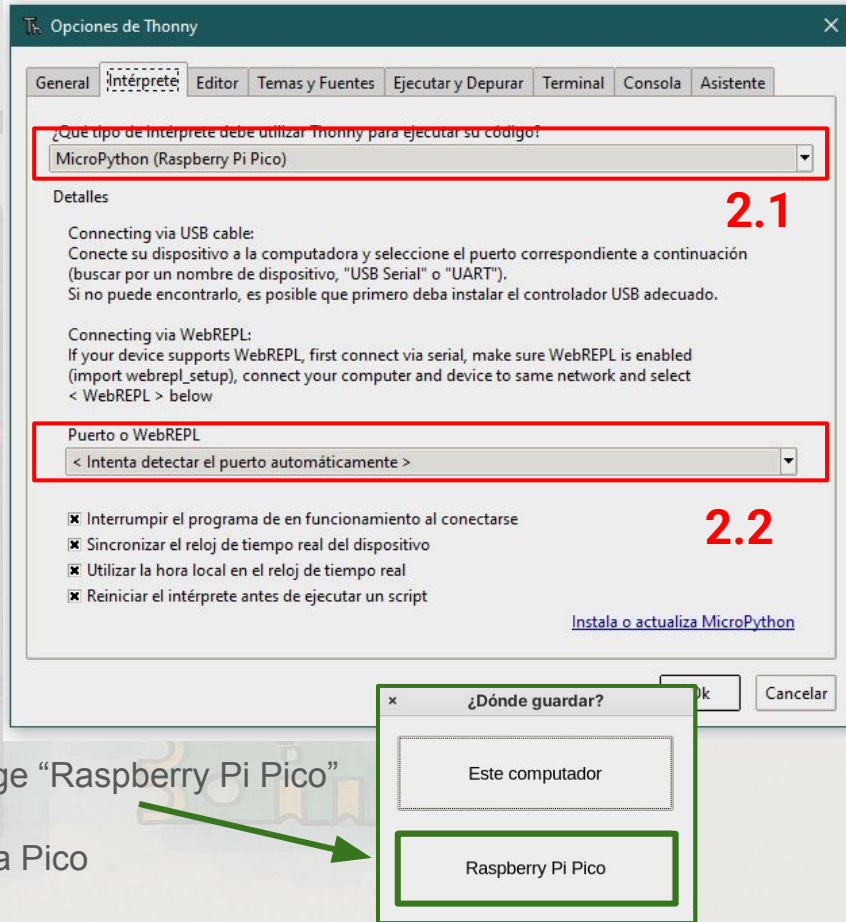
2.1 Selecciona "MicroPython (Raspberry Pi Pico)".

2.2 En "Puerto o WebREPL" -> selecciona el puerto COM....

3. Siempre puedes pulsar abajo a la derecha, cuando Thonny detecte tu placa.

MicroPython (Raspberry Pi Pico) • Board CDC @ COM13

4. Ejecuta el código de [saludo.py](#) en tu Pico
5. Guarda el código en la Pico, usa la opción "Guardar Como" y elige "Raspberry Pi Pico"
6. Para ver los archivos de la Pico, activa "Visualizar" -> "Archivos"
7. Con la pestaña "Archivos" puedes enviar ficheros entre el PC y la Pico

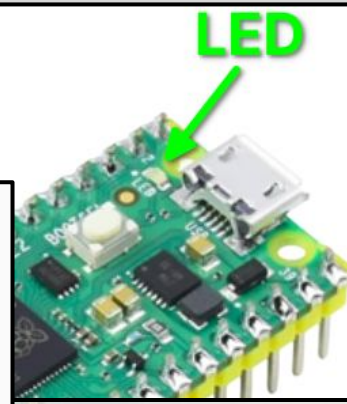


Encendemos el LED integrado en la Pico

1. Copia esta programación a Thonny:

```
# Encendemos el LED de la placa durante 3 segundos
from machine import Pin # Importamos Pin del módulo machine
import time            # Importamos el módulo time completo
led = Pin('LED', Pin.OUT) # Configuramos como salida
led.on()               # Encendemos
time.sleep(3)          # Esperamos 3 segundos
led.off()              # Apagamos
```

[Código](#)



2. Ejecuta el programa:



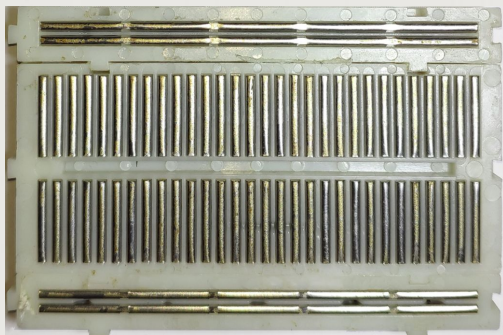
3. Vamos a hacer que el LED parpadee...

- Añadiendo un bucle **While**.
- Añadiendo un tabulador a **las líneas a repetir**.
- Añadimos una **espera** mientras está apagado.
- Para detener el programa pulsamos **Ctrl + C**

```
# Hacemos que el LED de la placa parpadee cada 1 segundo
from machine import Pin # Importamos Pin del módulo machine
import time            # Importamos el módulo time completo
led = Pin('LED', Pin.OUT) # Configuramos como salida
while True:
    led.on()            # Encendemos
    time.sleep(1)       # Esperamos 1 segundo
    led.off()           # Apagamos
    time.sleep(1)       # Esperamos 1 segundo
```

[Código](#)

Montando nuestra electrónica con Protoboard/Breadboard



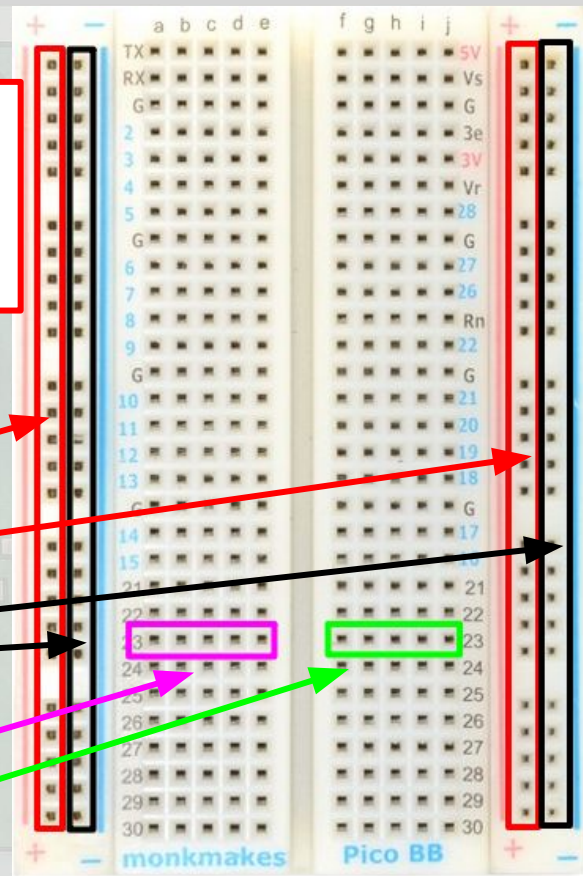
Interior de una protoboard

¡¡REVISA BIEN TUS MONTAJES!!

Recuerda: Cuando huele a humo...
¡¡Ya es tarde!!

La breadboard/protoboard (placa de prototipado):

- Las líneas laterales (railes) conectan todos los agujeros de la **columna**
 - Columnas Positivas +
 - Columnas Negativas -
 - No están conectadas las de un lado y las del otro
 - No aportan alimentación por sí mismas
- Las **filas** conectan los 5 agujeros entre ellos:
 - No están conectadas las de un lado y las del otro.

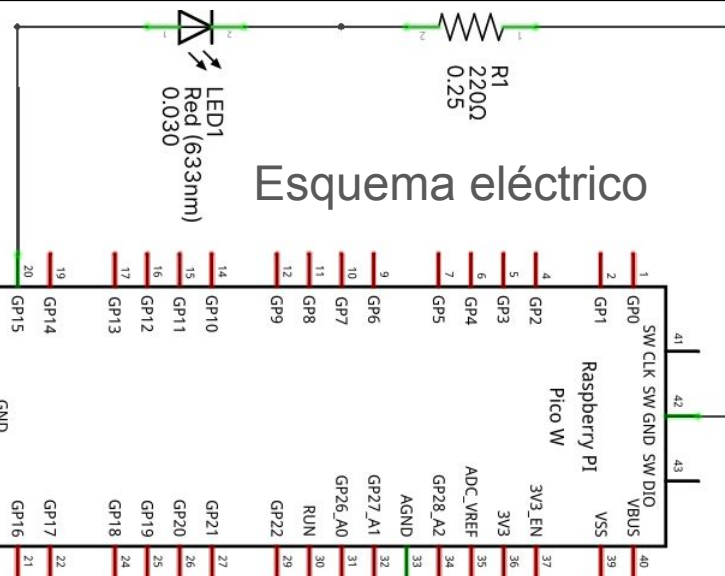
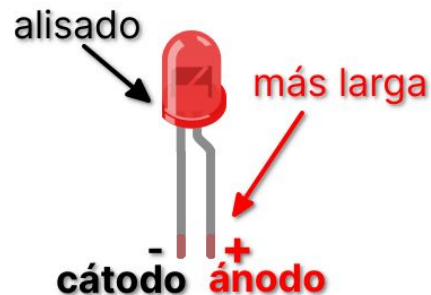


Placa Pico BB

Conectando LEDs externos

LEDs:

- Tienen **polaridad**:
 - La patilla positiva es más larga.
 - El lado negativo está "alisado"
 - Funcionan a menos de 3V por lo que **necesitan una resistencia** de al menos 220Ω en serie.
 - Las resistencias no tienen polaridad.
- Existen en **multitud de colores**
 - Distintos colores consumen diferentes potencias.



Conectamos...

- El **ánodo (+)** del LED a GPIO15
- El cátodo (-) a una resistencia de 220Ω para limitar la corriente.
- El otro extremo de la resistencia a una patilla GND (Todas las patillas GND son equivalentes).

Conectamos LED externo

Conexión:

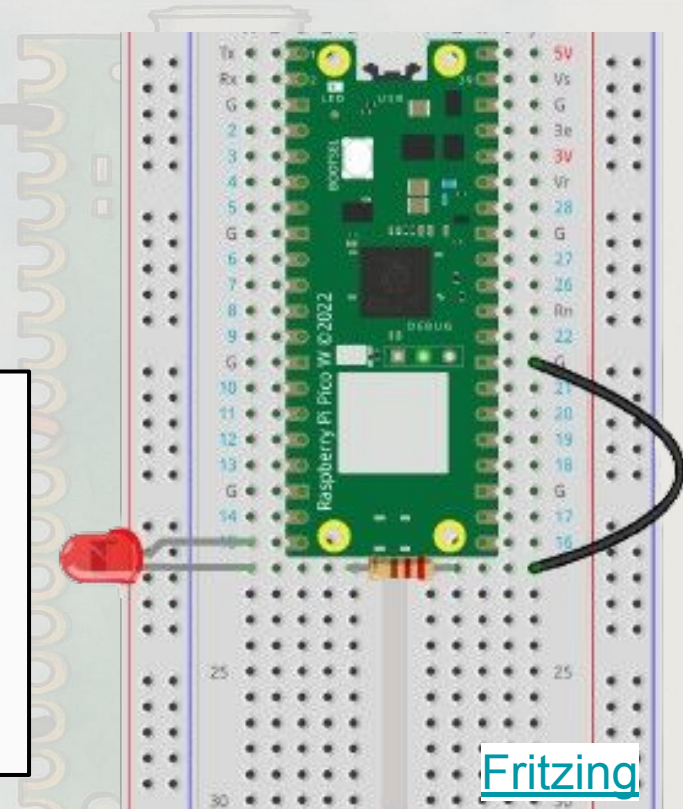
- **Patilla larga (ánodo +)** del LED a GPIO15
- Resistencia de 220Ω conectada al **cátodo (-)** del LED
- Cable (**negro** si es posible) entre la resistencia y un **GND**

Programación:

- Configuramos el GPIO15
- En el bucle: Encendemos y apagamos el LED.

```
# Hacemos que un led conectado a GPIO15 parpadee cada 1 segundo
from machine import Pin # Importamos Pin del módulo machine
import time            # Importamos el módulo time completo
led = Pin(15, Pin.OUT) # Configuramos GPIO15 como salida
while True:
    led.on()            # Encendemos
    time.sleep(1)       # Esperamos 1 segundo
    led.off()           # Apagamos
    time.sleep(1)       # Esperamos 1 segundo
```

[Código](#)



[Fritzing](#)

Experimento: conecta LEDs de varios colores en paralelo ¿qué ocurre?

Ejercicio: haz que el LED de la Pico y el externo parpadeen alternativamente. [Solucion](#)



Encendiendo y apagando el LED manualmente



Usaremos un pulsador en GPIO14 para encender/apagar el LED en GPIO15

```
# Encendemos y apagamos el LED con un pulsador
from machine import Pin
# Configuramos GPIO 15 como salida
led = Pin(15, Pin.OUT)
# Configuramos GPIO14 como entrada
# y en estado bajo (Low/0) por defecto (PULL_DOWN)
boton = Pin(14, Pin.IN, Pin.PULL_DOWN)
while True:
    # Bucle infinito
    if boton.value(): # Si Botón pulsado
        led.on()      # Encendemos el LED
    else:             # Si no
        led.off()     # Apagamos el LED
```

Código



Conectamos el pulsador a GPIO14

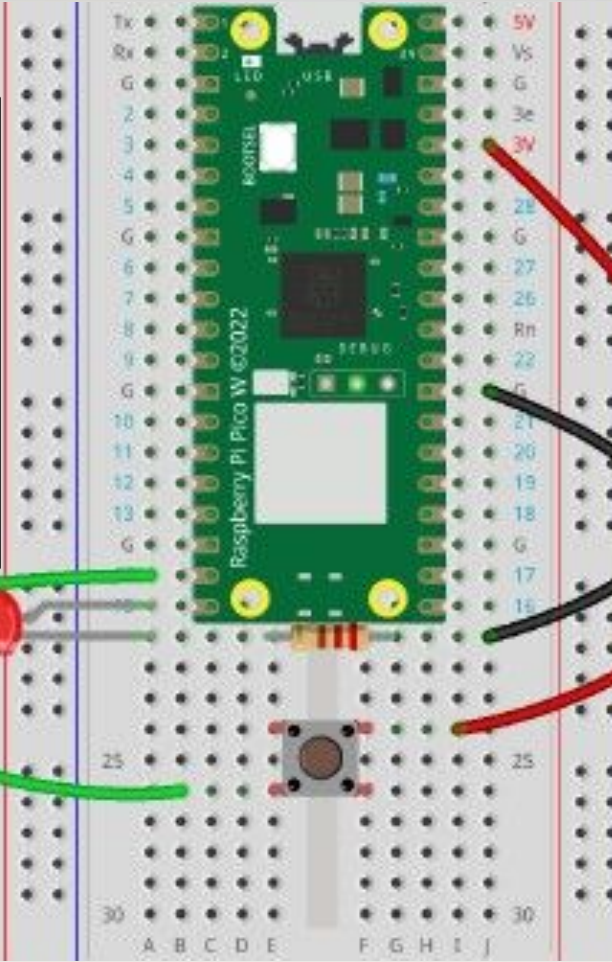
Conectamos el pulsador 3V

Componentes

- Breadboard
- Pico W
- LED
- 3 cables
- Resistencia 220Ω
- Pulsador

Ejercicio: Podríamos hacer lo mismo eléctricamente, sin programación ¿Cómo?

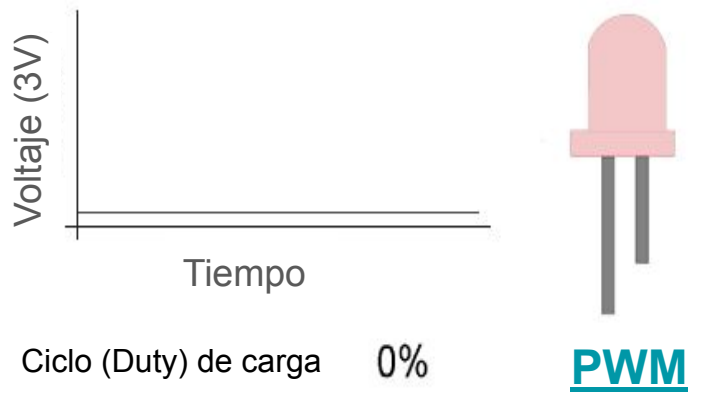
[Solución](#)



Controlando el brillo del LED (PWM)

Vamos a regular el brillo de un LED por medio de PWM.

- **PWM = Modulación de anchura de pulso.**
- Generamos una señal de alta frecuencia (500Hz) que el ojo no ve.
- Encendemos y apagamos el led muy rápidamente y lo dejamos encendido un % del tiempo generando así un % de brillo



```
from machine import Pin, PWM
led = PWM(Pin(15)) # Controlamos GPIO15 con PWM
led.freq(500)      # Usaremos un PWM de frecuencia 500Hz
# Primero subimos el brillo (Fade In)
for brillo in range(0, 65536): # desde 0 a 65535
    led.duty_u16(brillo)
# Bajamos el brillo (Fade Out)
for brillo in range(65535, -1, -1): # desde 65535 hasta 0
    led.duty_u16(brillo)
```

[Código](#)



Ejercicio: Añade un pequeño retardo, de una milésima, con `time.sleep(0.001)` ¿Cuánto tarda ahora el programa?
¿Por qué?

Experimento: ¿qué ocurre si bajamos la frecuencia del PWM por debajo de 30Hz? ¿Cual es el límite?



Ajustando manualmente el brillo con un potenciómetro



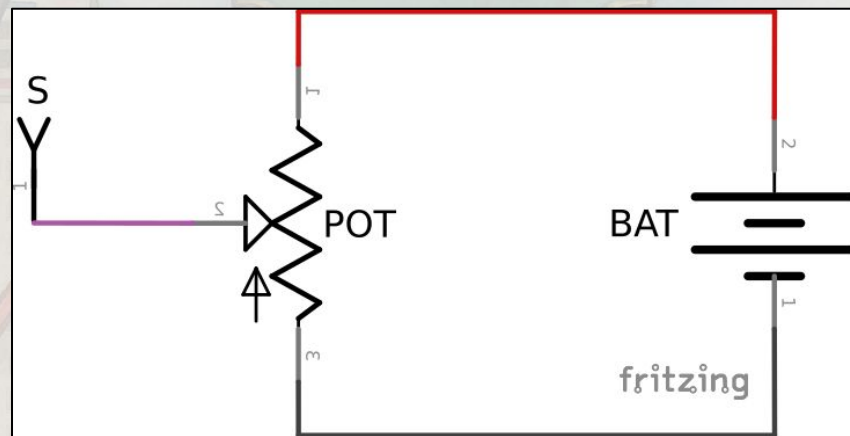
Digital: 0V o 3V

Cualquier valor entre 0V y 3V



- Un potenciómetro es capaz de generar voltajes intermedios entre un mínimo y un máximo, que en nuestro caso serán 0 y 3V
- Mediremos la señal y generaremos un brillo proporcional en el LED.
- Para medir usamos [ADC](#): Conversión de Analógico a Digital
- Obtenemos un entero entre 0 y 65535 equivalente a 0V y a 3V

Ejercicio: Podríamos hacerlo electrónicamente usando una parte del potenciómetro como resistencia ¿Cómo? [Solución](#)



1. Patilla inferior a GND
2. Patilla de en medio genera voltaje intermedio
3. Patilla superior a 3V

Ajustando manualmente el brillo con un potenciómetro

```
# Controlamos el brillo de un LED con un potenciómetro
from machine import Pin, ADC, PWM
import time

# Configuramos GPIO26 como entrada analógica
pot = ADC(26) # Medimos el voltaje del GPIO26 0 -65535
led = PWM(Pin(15)) # Controlamos el brillo del GPIO15
led.freq(500)
while True:
    pot_valor = pot.read_u16() # Medimos el valor entre 0 y 65535
    led.duty_u16(pot_valor) # Damos el mismo valor de brillo
    print(3*pot_valor/65535, 'V') # Imprimos el voltaje medido
    time.sleep(0.01)
```

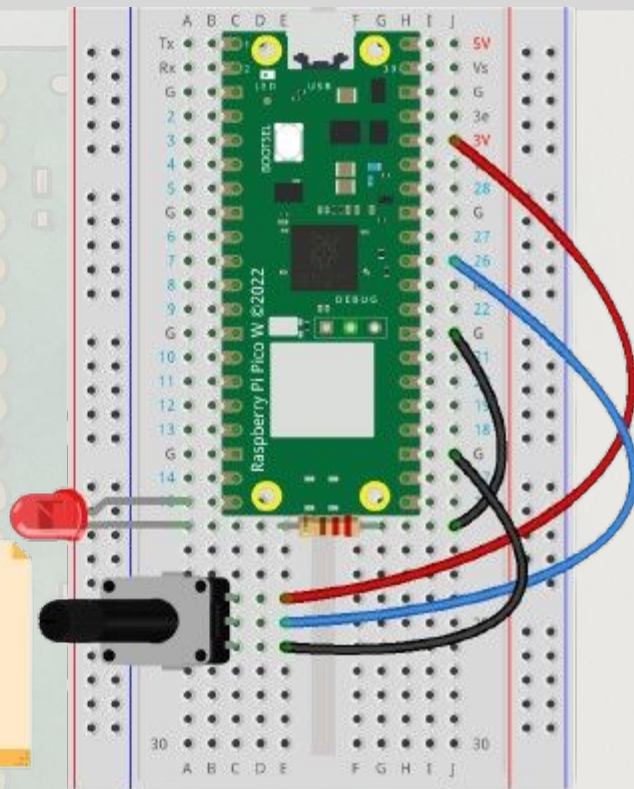
Código

Necesitamos:

- Breadboard.
- Microcontrolador.
- LED.
- 4 cables.
- Resistencia 220Ω
- Potenciómetro.



Un extremo - 3V
El pin central - GPIO26
El otro extremo - GND



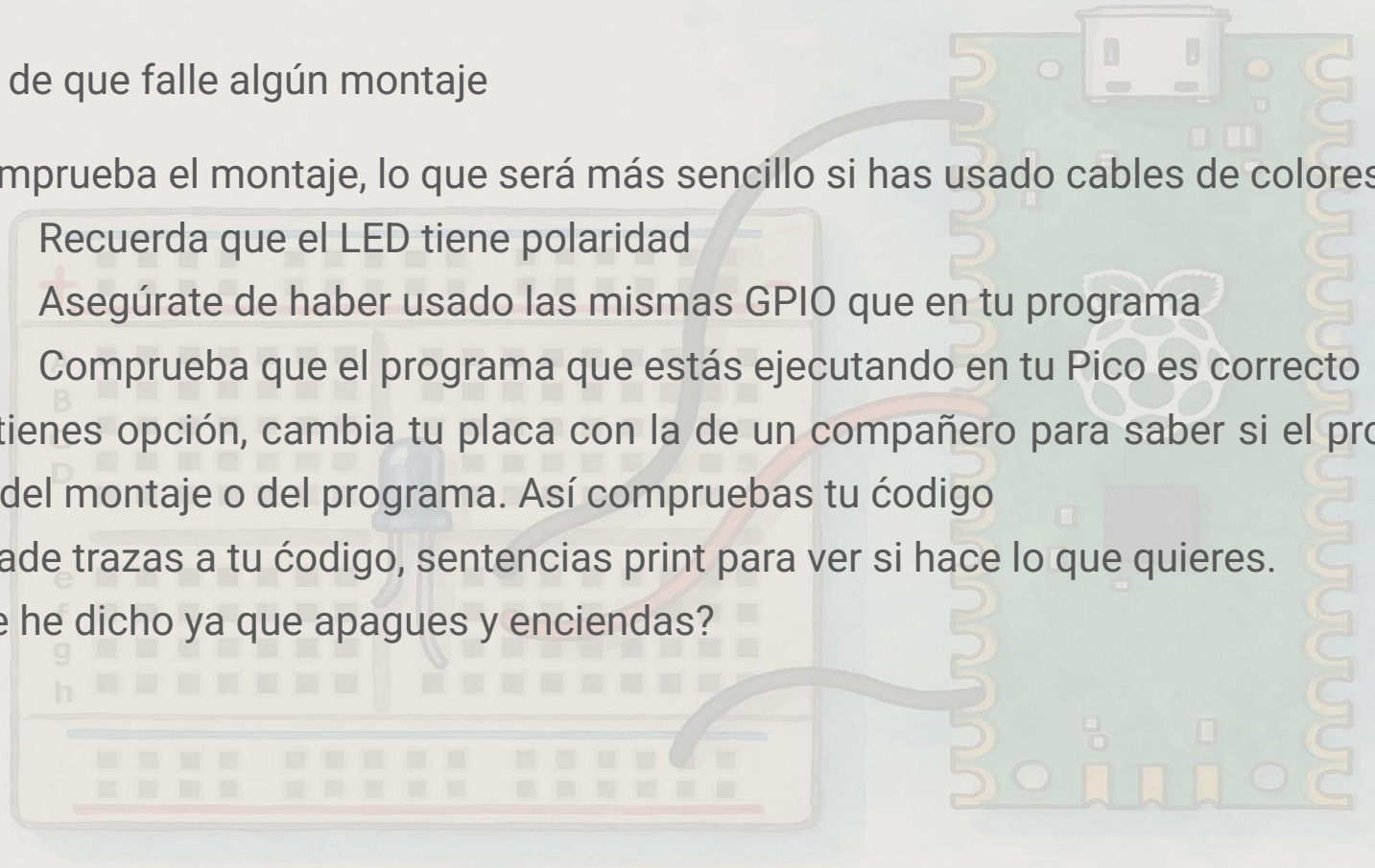
Ejercicio: Podríamos hacerlo electrónicamente usando una parte del potenciómetro como resistencia ¿Cómo? [Solución](#)



Comprobaciones básicas a realizar si falla un montaje:

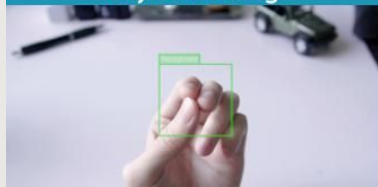
En caso de que falle algún montaje

1. Comprueba el montaje, lo que será más sencillo si has usado cables de colores.
 - a. Recuerda que el LED tiene polaridad
 - b. Asegúrate de haber usado las mismas GPIO que en tu programa
 - c. Comprueba que el programa que estás ejecutando en tu Pico es correcto
2. Si tienes opción, cambia tu placa con la de un compañero para saber si el problema es del montaje o del programa. Así compruebas tu código
3. Añade trazas a tu código, sentencias print para ver si hace lo que quieres.
4. ¿Te he dicho ya que apagues y enciendas?

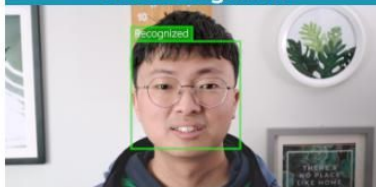


¿Para qué sirve HuskyLens?

Object Tracking



Face Recognition



Object Recognition



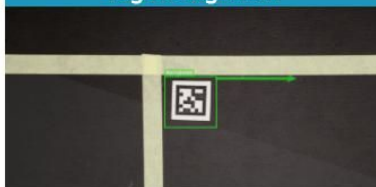
Line Tracking



Color Recognition



Tag Recognition



Cable para conectar con Pico, micro:bit o Arduino

HuskyLens: Cámara con inteligencia artificial diseñada para proyectos de robótica, IoT y automatización:

1. Reconocimiento facial.
2. Seguimiento de objetos.
3. Reconocimiento de objetos.
4. Seguimiento de línea.
5. Detección de color.
6. Detección de etiquetas.
7. Clasificación de objetos.

¿Es más potente ¿HuskyLens o la Pico?

HuskyLens **NO** funciona de forma independiente; **NO** necesita una placa controladora, como Raspberry Pi Pico, para interpretar los datos y ejecutar acciones.

Estructura de la HuskyLens

Botón de aprendizaje LED RGB Botón de función

Procesador
IA

Tarjeta SD

Conector externo
(4 cables)

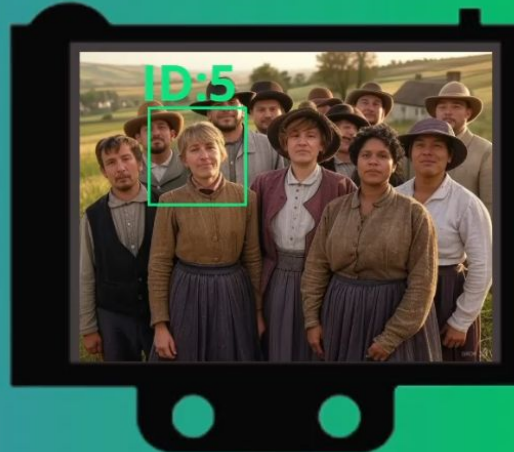
microUSB

Pantalla de 2 pulgadas



Es bastante frágil, mejor imprimir alguna [carcasa imprimible](#)

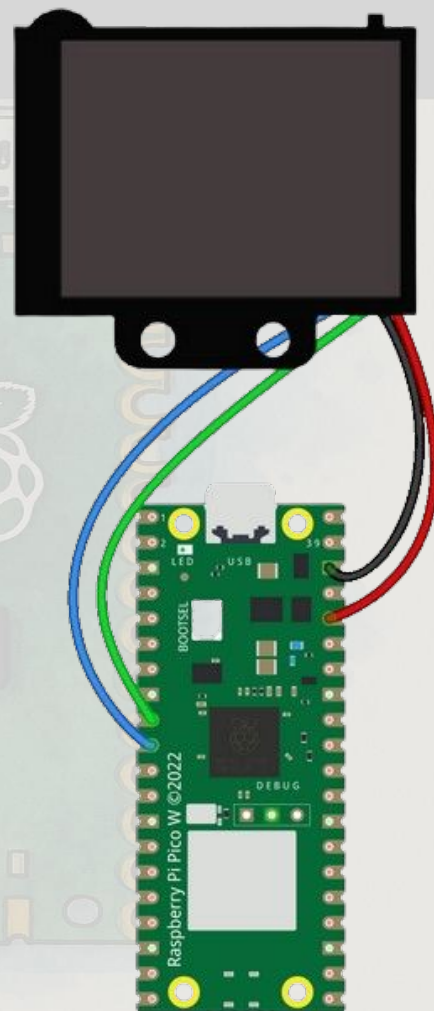
Memorizando Caras



Conecta la HuskyLens con Pico

Conecta el cables de conexión que trae la HuskyLens a los pines de la Raspberry Pi Pico de esta forma:

- VCC → 3,3V (Pin 36. Se encarga de la alimentación)
- GND → GND (Pin 38, o cualquier GND. Es la toma de tierra)
- SDA → GPIO 6 (SDA) (Pin 6. Permite paso de datos)
- SCL → GPIO 7 (SCL) (Pin 7. Sincroniza)



Configura tu HuskyLens para comunicación I2C

Nuestro programa de detección de caras

[Tutorial más detallado](#)

Es imprescindible usar [una librería](#) para que ambos dispositivos se entiendan.

Sigue estos pasos:

1. Conecta la Raspberry Pi Pico con el circuito a tu computadora con un cable USB.
2. Abre Thonny y conecta el USB de la Pico.
3. Abre un programa nuevo y copia este [programa](#).
4. Descarga y copia la [librería](#) con el nombre exacto “pyhuskylens.py”
5. Guarda DENTRO de la Pico.
6. EJECUTA el programa



Raspberry Pi Pico

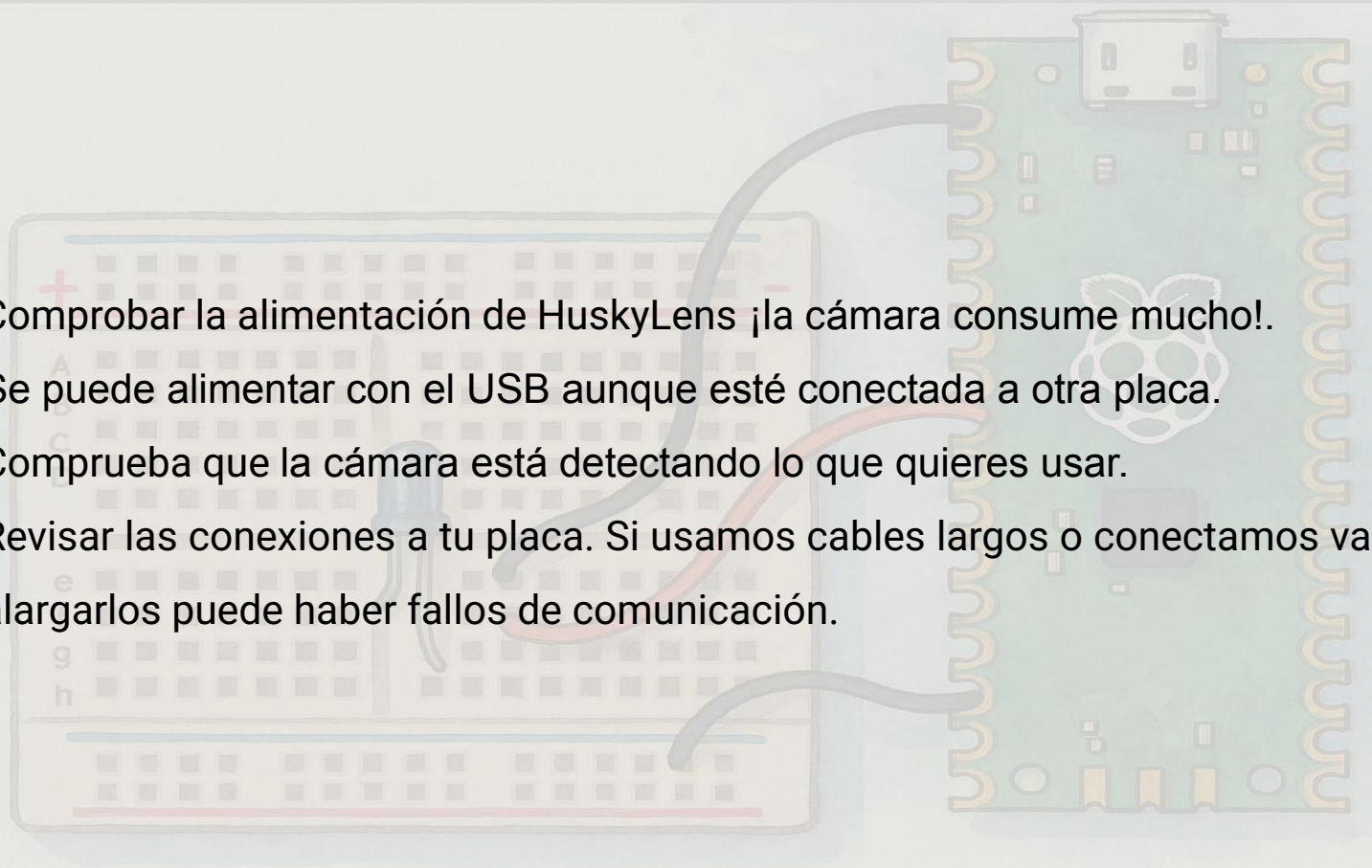
Raspberry Pi Pico

pyhuskylens.py

test_1_pyhuskyLens.py

Si nuestro proyecto con la HuskyLens falla....

1. Comprobar la alimentación de HuskyLens ¡la cámara consume mucho!.
2. Se puede alimentar con el USB aunque esté conectada a otra placa.
3. Comprueba que la cámara está detectando lo que quieres usar.
4. Revisar las conexiones a tu placa. Si usamos cables largos o conectamos varios para alargarlos puede haber fallos de comunicación.



Apéndice: Usando Pico + HuskyLens con microblocks

<http://microblocks.fun/>

- Entorno visual de bloques para programar varias placas con multitud de librerías disponibles.
- Podemos programar la Pico y tenemos librería para la HuskyLens.
- También podemos programar la micro:bit, la ESP32, la ESP8266 y muchas más.

MicroBlocks

Connect

1 Conectamos la Pico

2 Actualizamos el firmware

3 Podemos usar bloques...

4 Importar librerías

5 AI → HuskyLens

when started

forever

set user LED

wait 500 milliseconds

set user LED

wait 500 milliseconds

File Open

AI

HuskyLens

Integrates DIYBOT
HuskyLens camera into
MicroBlocks.
Supports I2C and Serial 9600
Full Arduino Lib support
Changes:
- New _H_ init lib auto init
- New HL set Commas
- Renamed Lcam blocks
- SERIAL 9600 support
- SERIAL read size correction
- Parsing correction

v2.1 by Murat Akgul, Turgut G

Cancel Open

¡¡Gracias Javier!!

Apéndice II: Usando micro:bit + HuskyLens con Makecode



Proyecto Portero automático con Makecode



Microsoft | micro:bit Bloques JavaScript

Buscar...

- Básico
- Entrada
- Música
- LED
- HuskyLens**
- Maqueen v4
- Maqueen v5
- Bucles

Extensiones

Código "telefonillo"

2 Inicialización

Usaremos MakeCode y las extensiones de maqueen y HuskyLens

3

al presionarse el botón A El botón A es el del Porterillo

HuskyLens request data once and save into the result Gracias Ignacio!!!

si HuskyLens check if ID 1 frame is on screen from the result

LED izquierdo encender

LED derecho apagar

mostrar ícono

Los leds de maqueen actúan como el cierre de la puerta

si no

LED izquierdo apagar

LED derecho encender

mostrar ícono

al iniciar

Usamos comunicaciones I2C

HuskyLens initialize I2C until success

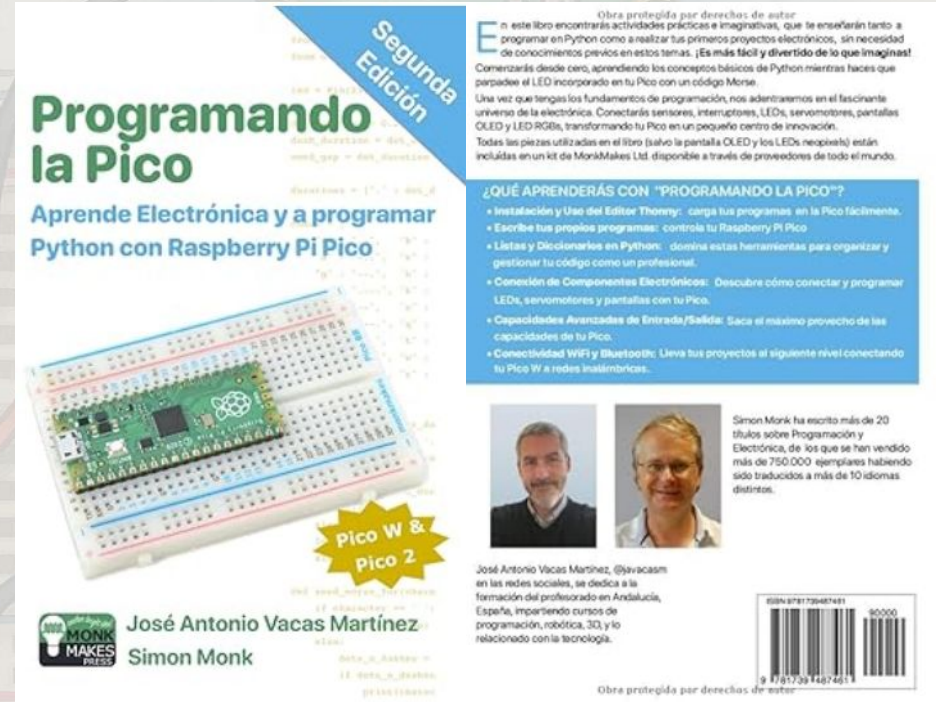
HuskyLens switch algorithm to Face Recognition

[Vídeo](#)

¡Muchas gracias!

Si queréis aprender más de micropython, algo más de electrónica con la Pico...

[Programando la Pico...](#)



by [@javacasm](#)