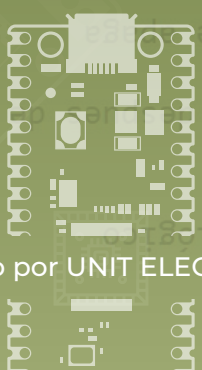
GUÍA DE APRENDIZAJE

UNIT PI PICO KIT_

Tan fácil como imaginarlo

UNIT ELECTRONICS



Elaborado por UNIT ELECTRONICS

ÍNDICE

INTRODUCCIÓN

¿Qué es UNIT PI PICO KIT?

El UNIT PI PICO KIT te ayudará a programar con el lenguaje de programación Microphyton desde cero; ya que es relativamente simple debido a su fácil sintaxis. Podrás desarrollar códigos desde la interfaz de Thonny y ver tus proyectos funcionando sobre la tarjeta de desarrollo Raspberry Pi Pico.

Es un kit básico para Raspberry Pi Pico que te ayudará a conocer el mundo de la electrónica armando diferentes proyectos desde la tarjeta de conexiones de Pi Pico. Entre las prácticas podrás realizar:

- Conocimiento de Interfaz Thonny
- Uso de núcleos de la Pi Pico
- Control de pines digitales, analogico y PWM
- Comunicación Serial
- Manejo de sensores y actuadores

¿Qué incluye el UNIT PI PICO KIT?

Este Kit incluye la tarjeta Raspberry Pi Pico, tarjeta de conexiones Pi Pico, variedad de componentes electrónicos y prácticas guiadas en formato digital.

¿Qué es Raspberry PI Pico?

Raspberry PI Pico es una tarjeta de desarrollo basada en el microcontrolador RP2040, el cual fue diseñado por Raspberry Pi y cuenta con dos núcleos ARM Cortex-M0+ que trabajan a 133 MHz, memoria Flash de 2 MB y con memoria RAM de 264 KB.
La tarjeta cuenta con entradas y salidas (I/O) para comunicación I2C, SPI y I/O programables (PIO). Estos admiten un sinfín de posibles aplicaciones integrando sensores, display´s, actuadores y otros módulos.

LISTA DE MATERIALES

El material que incluye es el siguiente:

Tarjeta de desarrollo: Raspberry Pi Pico

Conectores	Componentes / Módulos	Optoelectrónica
Header Macho Tira de 40 Pines 2.54mm	Buzzer Zumbador 5V Activo	Display LCD 16×2 con I2C Fondo Azul
Cable USB a MicroUSB 1m	Push Button Grande 12mmx12mm (3)	Tira Neopixel WS2812 5050 RGB LED
Paquete de Cables Dupont Cortos M-M	Potenciometro 10K 3 Pines 15mm WH148	Led Rojos 5mm(5)
Paquete Cables Dupont Cortos H-M	Resistencia de 1/4w 5% 330Ω (5)	Led Verdes 5mm(5)
140 cables para puentear 14 tamaños protoboard	Módulo Relevador de 1 Canal 5V DC	Led Amarillos 5mm(5)
Protoboard (400 puntos)	HC-SR501 Sensor de Movimiento PIR	
	Tarjeta de conexiones para Pi Pico	

PREPARACIÓN DE HARDWARE Y SOFTWARE

¿QUÉ ES PYTHON?

Python es un lenguaje de programación de alto nivel siendo versátil y de código legible.

Además de ser un lenguaje de propósito general, es decir, que puede ser utilizado para cualquier aplicación, como: desarrollo web, automatización de script, machine learning(ML), desarrollo de software, por mencionar algunos.

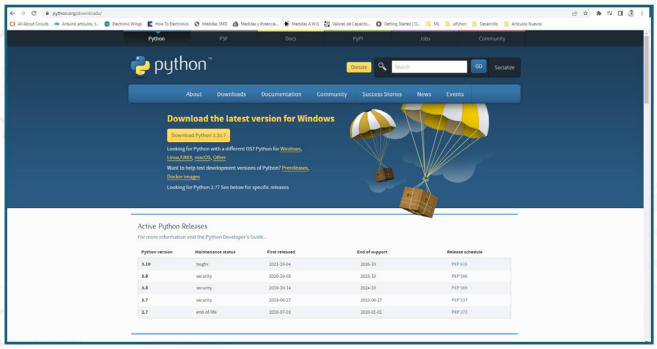
INSTALANDO PYTHON

Trabajaremos con la versión más reciente de Python 3, así tendrás acceso a los sets de instrucciones y bibliotecas de este lenguaje.

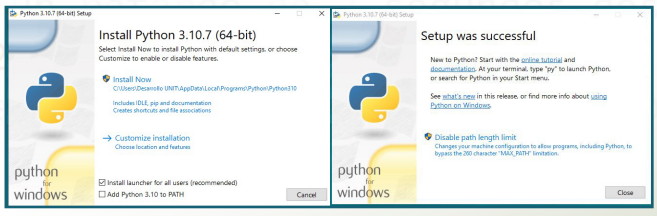
Ingresa a la siguiente liga y descarga, si ya lo tienes instalado puedes saltar al siguiente paso.

https://www.python.org/downloads/

Da clic en el botón de “Download Python” y se descargará un archivo ejecutable .exe.



Da clic en él y ejecútalo en tu computadora.



THONNY: INTÉRPRETE DE PYTHON Y FIRMWARE PARA RASPBERRY PI PICO

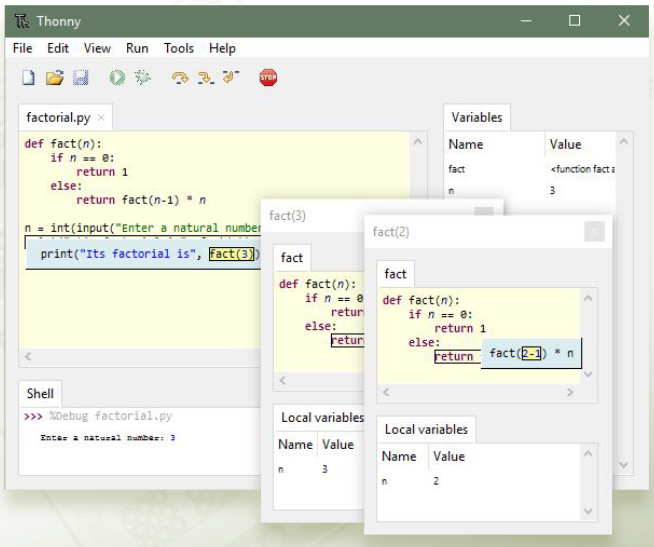
Thonny es un entorno de desarrollo IDE en lenguaje Python, es ideal para empezar en el mundo de la programación de microcontroladores con Python.

Te ofrece la mayoría de las ventajas que entrega este lenguaje de programación traducido a un nivel que el microcontrolador pueda entender: MicroPython.

INSTALANDO THONNY

Una vez instalado Python, necesitaremos el traductor que nos ayude a crear y ejecutar código en nuestra Raspberry Pi Pico.
En esta guía se hace uso de Thonny IDE siendo el más popular para tarjetas como Rasbperry Pi Pico, ESP32 y ESP8266. Ingresamos al siguiente enlace oficial para descargar el software:

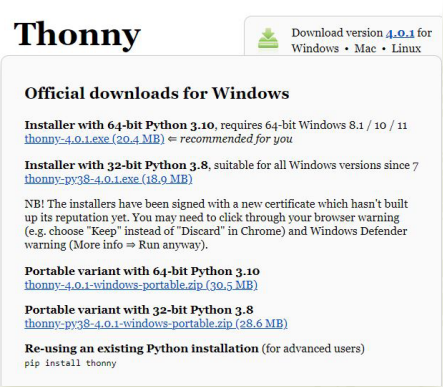
https://thonny.org/



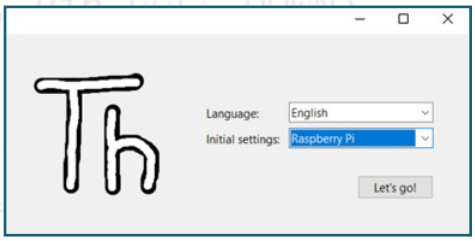
Continuación

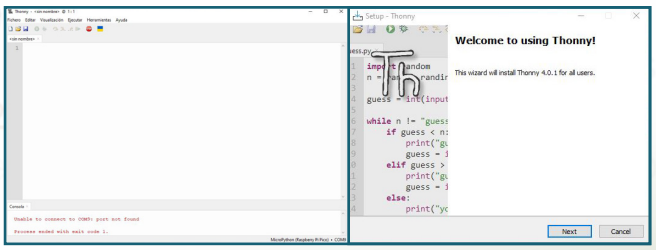
PREPARACIÓN DE HARDWARE Y SOFTWARE

Colocamos el mouse sobre el sistema operativo que manejemos y se desplegará un menú con las versiones disponibles de descarga, te recomiendo descargar la versión más reciente, incluso la misma página web te indica que es la “recomendada para ti”.



Ejecutamos el archivo .exe e instalamos Thonny en nuestro equipo, solo da clic en aceptar a las distintas ventanas que te van apareciendo durante el proceso de instalación.
Al final te aparecerá un mensaje de instalación exitosa, cuando abras Thonny, saldrá una ventana donde deberás seleccionar en configuración inicial: Raspberry Pi. Felicidades a partir de este momento puedes abrirte paso en el maravilloso mundo de MicroPython.

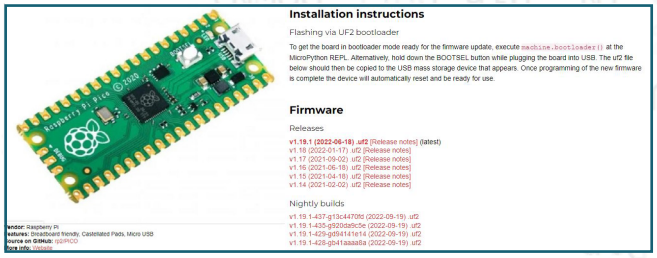




CONFIGURANDO RASPBERRY PI PICO EN THONNY

Para que nuestra placa funcione con MicroPython se debe de cargar el firmware, que es el programa que controla los circuitos de la Pi Pico para poderlos programar.
Ingresa al siguiente enlace y da clic sobre la última versión que te aparezca en la sección de “Firmware -> Releases”

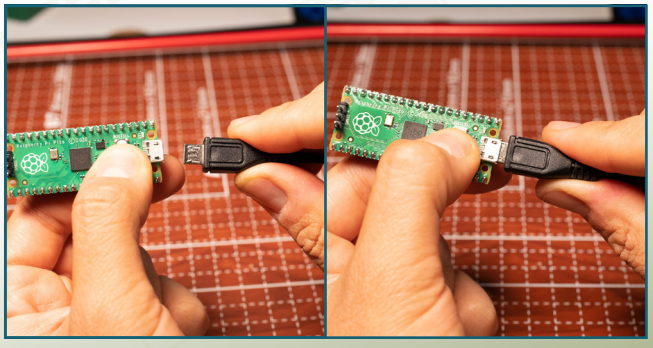
https://micropython.org/download/rp2-pico/



El siguiente paso es muy importante, se descargará un archivo .uf2, que son las instrucciones de instalación del firmware en la tarjeta:

1. Conecta el cable micro USB que incluye tu kit a uno de los puertos de la computadora.

2. Presiona el botón BOOTSEL de la placa Raspberry Pi Pico desconectada y sin dejar de presionar, conecta el cable al puerto USB de la placa.

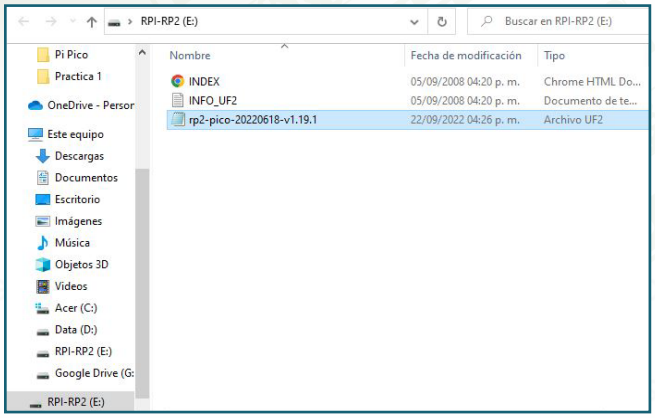


Continuación »

PREPARACIÓN DE HARDWARE Y SOFTWARE

3 En la computadora aparecerá una pestaña de unidad de almacenamiento, como si insertamos una USB en la computadora.

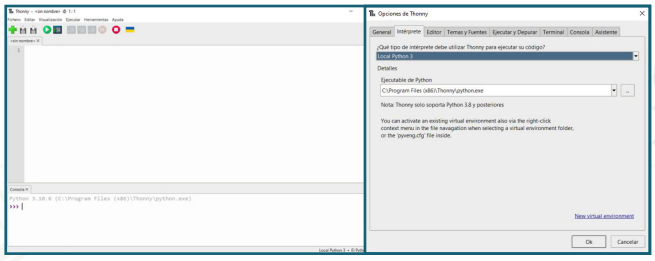
4 Arrastramos o copiamos el firmware que descargamos en esa pestaña.



5 Se abrirá una pestaña de copiado, no realices nada, el proceso se generará sólo y una vez finalizado la pestaña se cerrará de manera automática. A partir de este momento Thonny ya reconocerá tu Pi Pico.

6 Abre tu IDE de Thonny, da click en “Cambiar a modo regular” y reinicia Thonny

7 En la ventana de “Ejecutar” damos clic en “Configurar intérprete”, se deberá desplegar la siguiente ventana:



8 Dar clic en la opción: ¿Qué intérprete o dispositivo debe usar Thonny para ejecutar su código?

Dar clic en la opción “MicroPython (Raspberry Pi Pico)”.

IMPORTANTE

Puedes encontrar el código completo que utilizamos en este manual dentro de nuestro repositorio de GitHub.

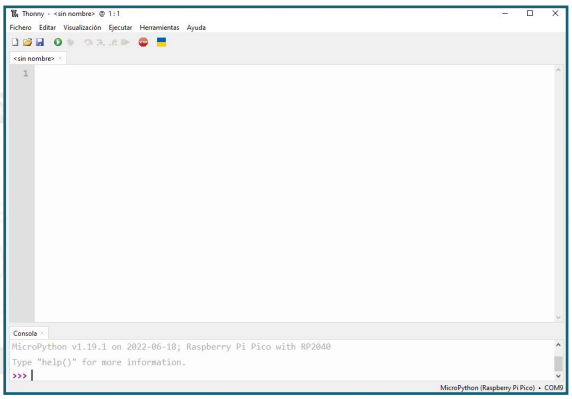
https://github.com/UNIT-Electronics/

PROYECTO 1: HELLO WORLD! INTRODUCCIÓN A MICROPYTHON

INTRODUCCIÓN

Comencemos a familiarizarnos con el entorno de Thonny y de MicroPython.
Al abrir Thonny lo primero que puedes notar es que el IDE se divide en dos secciones:

- **Editor:** Se encuentra en la parte superior y puedes escribir el código a ejecutar en la tarjeta Pi Pico.
- **Consola:** (formalmente es llamada REPL pero nos referiremos a ella como Consola para fines prácticos): Podrás comunicarte con la Raspberry Pi Pico, en ella puedes ingresar datos a la tarjeta y lo que te responda la tarjeta podrás visualizarlo en esta área.



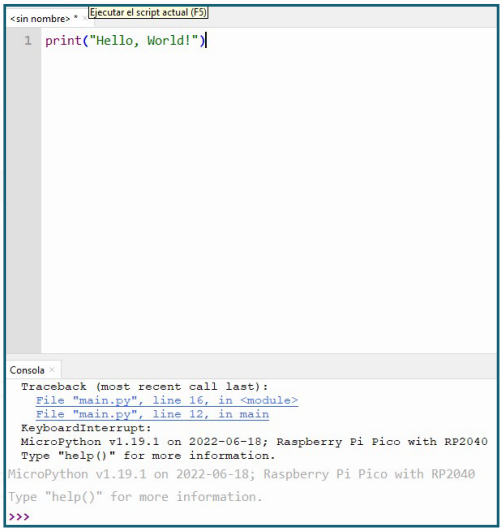
DESARROLLO

Probemos ambas secciones para que descubras su funcionamiento. Escribe el siguiente código en la sección del editor

```
print("Hello, World!")
```

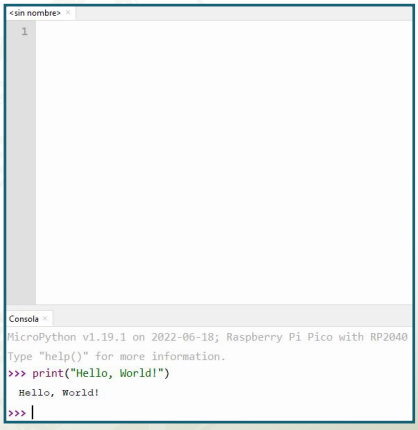
Ahora ejecutaremos el código, da clic en el botón Verde con flecha blanca de "Ejecutar el script actual".

Si todo sale en orden, el código se ejecutará y aparecerá el mensaje: "Hello, World!" en la consola, así puedes imprimir mensajes en pantalla.



Ahora escribiremos el mismo código pero en la consola. Escribe la línea de código y da un enter, ¿Qué notaste?.

El mensaje de Hello, World! Se ejecutó en la misma consola sin la necesidad de dar clic en ejecutar script.
Es otra de las ventajas de MicroPython puedes enviar código desde la consola sin pasar por el Editor.



Dato Curioso de Python

No es necesario usar punto y coma al final de cada línea para que se ejecute el código en la sección de Editor y Consola.

PROYECTO 2: DECLARACIÓN DE VARIABLES

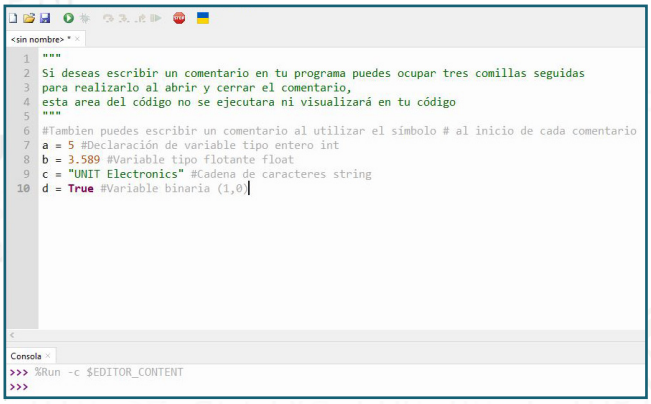
INTRODUCCIÓN

El uso de variables en Python como en los demás lenguajes de programación es muy importante.
Las variables nos ayudan a almacenar información de mediciones o alguna operación matemática. Existen distintos tipos, las más comunes son:

- **int:** Variables de números enteros
- **float:** Números con punto decimal
- **string:** Cadenas de caracteres
- **char:** Carácter
- **bool:** Binario

DESARROLLO

En la sección del Editor asignaremos valores de diferentes tipos a las variables: a, b, c y d.

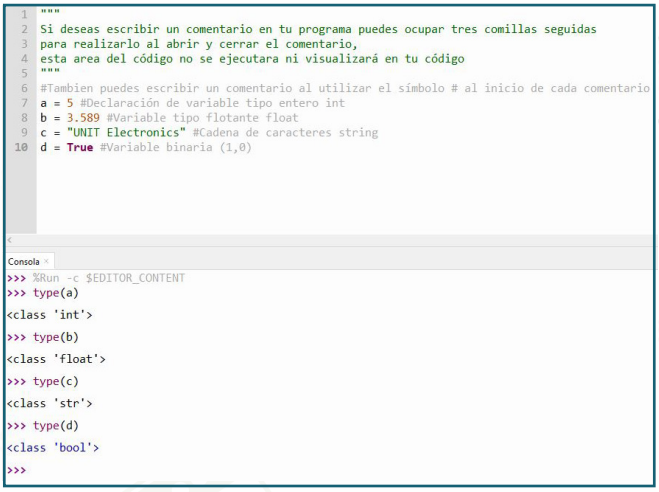



```
"""
Si deseas escribir un comentario o recordatorio en tu
programa puedes ocupar tres comillas seguidas al abrir y
cerrar el comentario,
esta área del código no se ejecutará ni visualizará en
tu código
"""
#También puedes escribir un comentario al utilizar el
#símbolo al inicio de cada comentario
a = 5 #Declaración de variable tipo entero int
b = 3.589 #Variable tipo flotante float
c = "UNIT Electronics" #Cadena de caracteres string
d = True #Variable binaria (1,0)
```

Al ejecutar el script no deberá de aparecer ningún error en la consola. Para poder comprobar que las variables tienen el valor asignado y además saber que tipo de dato le fue asignado, usaremos la función *type()*.

En la sección de la consola escribiremos la función type(), dentro de los paréntesis las variables que previamente declaramos, de la siguiente manera:

```
type (a)
type (b)
type (c)
type (d)
```



En la misma sección de la consola, se visualiza la información del tipo de variable declarada.

Dato Curioso de Python

Al declarar una variable no es necesario indicar que tipo de dato se desea guardar, basta con escribir la variable e igualarla con el dato a almacenar.

PROYECTO 3: EJECUTANDO UN LOOP (CICLO FOR)

INTRODUCCIÓN

En este proyecto se comprenderá que es un ciclo ***For*** , el cual , es una porción de código que se ejecuta de forma cíclica por un cierto número de veces.

Se compone de un valor de ***inicio*** y un valor de ***parada*** que da por finalizada la ejecución de código.

La sintaxis de un ciclo ***For*** es la siguiente:

```
for i in range (a, b, c):
>>>> Acción a realizar de forma cíclica
```

Donde:

i = variable que llevará la cuenta de cuántos ciclos se han cumplido

a = Valor de inicio de ciclo

b = Valor de parada

c = Opcional, indica cuántos valores se avanzan por ciclo

Es muy importante que dentro del ciclo exista una sangría o 4 espacios (>>>>) antes de cada línea, ya que esto es lo que indica que ese código se ejecuta dentro del ***For.***

DESARROLLO

Realizaremos una rutina del ciclo For en la sección de Editor.

```
print("Inicio del Ciclo")
for i in range(0,10):

    #No olvides insertar un espacio o sangría
    #en las líneas que se ejecutarán dentro del For
    print("Número de ciclo: ", i)

print("Fin del ciclo")
```

En el programa se visualiza un conteo del 0 al 9 en la ***consola.*** La sección del código que está dentro del For se repite 10 veces, por lo cual, no se imprime el número 10.
Si se quisiera imprimir hasta "Número de ciclo: 10", el rango de la rutina tendría que ser:

```
for i in range(0,11)
```

También como podrás observar el print(*"Fin del ciclo"*) solo se ejecuta una vez, esto es porque no lleva ningún espaciado al principio de la línea.

DESAFÍO

Otra forma de utilizar el ciclo For es indicando cuántos números avanzan en cada ciclo. En el ejemplo, el conteo se hizo de uno por uno, pero podemos contar de distintas formas.
Ejecuta los siguientes códigos en Thonny y observa lo que ocurre en la Consola.

```
print("Ciclo for de uno en uno")
for i in range(0,10):

    #No olvides insertar un espacio o sangría
    #en las líneas que se ejecutarán dentro del for
    print("Número de ciclo: ", i)

print("Ciclo for de dos en dos")
for i in range(0,10,2):
#Este ciclo contará de dos en dos cada ejecución
    print("Número de ciclo: ", i)

print("Ciclo for descendente")
for i in range(10,0,-1):
#Este ciclo se ejecutará de forma descendente
    print("Número de ciclo: ", i)
```

Dato Curioso de Python

Basta con colocar dos puntos (:) al final de la condición para indicar que las siguientes líneas pertenecen al ciclo For

PROYECTO 4: INGRESO DE CARACTERES FUNCIÓN INPUT()

INTRODUCCIÓN

Hasta el momento hemos asignado valores a las variables y esta información la podemos ver en la sección de consola, pero también podemos ingresar valores para que nuestro programa trabaje con ellos.

Esto se realiza a través de la función ***input(),*** la cual es capaz de recibir caracteres. Esta función lleva el siguiente formato:

```
x= input(“Ingrese valores desde el teclado”)
```

Como podrás observar para declarar la función input() es necesario almacenar el dato que estamos ingresando por teclado.

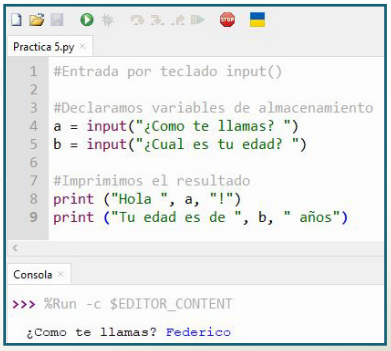
DESARROLLO

A continuación declaramos dos variables que guardarán información:

- una cadena de caracteres
- un número entero.

```
#Entrada por teclado input()

#Declaramos variables de almacenamiento
a = input("¿Cómo te llamas? ")
b = input("¿Cuál es tu edad? ")

#Imprimimos el resultado
print ("Hola ", a, "!")
print ("Tu edad es de ", b, " años")
```

En consola aparecerá el mensaje de **"¿Cómo te llamas?"**, se detiene la ejecución hasta que escribas la respuesta y presiones la tecla **enter,** posteriormente continuará el código.



Después el mensaje **“¿Cuál es tu edad?”** aparecerá, ingresa tu edad y presiona enter.

Al final la consola imprimirá tu nombre con tu edad.

DESAFÍO

Intenta colocar tu edad en carácter...
¿Qué tipo de variables son a y b?

PROYECTO 5: CICLO WHILE

INTRODUCCIÓN

El ciclo While es una porción de código que se ejecuta de forma indefinida hasta que una condición (que tu defines) resulta ser falsa. Al cumplirse, el ciclo finaliza y continúa la ejecución del resto del código.

Al igual que el ciclo For después de declarar la condicional se colocan dos puntos (:) y las líneas de código debajo de ellas que tengan 4 espacios o una sangría son las pertenecientes al ciclo.

```
while (condición):
>>>>Acción a realizar de forma cíclica
```

La condición puede ser cualquiera de las siguientes:

- **x == y** (x es igual a y)
- **x != y** (x es diferente a y)
- **x < y** (x es menor que y)
- **x > y** (x es mayor que y)
- **x <= y** (x es menor o igual que y)
- **x >= y** (x es mayor o igual que y)

DESARROLLO

Realizaremos un pequeño programa que te pide ingresar una contraseña que posteriormente se validará dentro de un ciclo While.

```
contraseña = 123456 #Definimos la contraseña

#Solicitamos al usuario ingrese la contraseña
usuario = int(input ("Ingresa la contraseña "))

#Mientras la contraseña sea incorrecta se ejecutará
#el siguiente código

while usuario != contraseña:

    #Informamos que la contraseña es incorrecta
    print("¡Contraseña incorrecta!")

    #Solicitamos nuevamente la contraseña
    usuario = int(input("Ingresa la contraseña "))

#Cuando la contraseña sea correcta permitimos el
#acceso
print("Acceso permitido")
```

Imagen 01

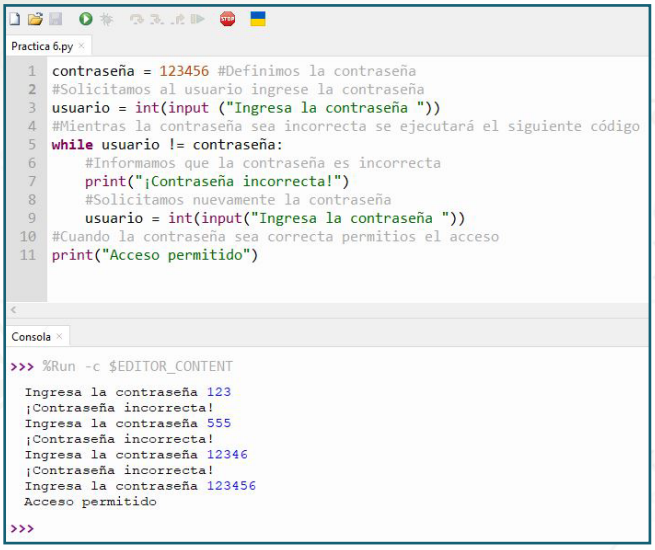


Imagen 01

Función input ()

En la función ***input ()*** observarás que lleva la siguiente sintaxis:

```
int(input("Ingresa la contraseña "))
```

Esto es porque la función input() devuelve caracteres, pero nosotros queremos trabajar con números enteros, para convertir los caracteres a números basta con declarar la función ***int()*** y se realizará la conversión. Puedes hacer lo mismo con las funciones ***float()*** y ***bool().***

PROYECTO 6: OPERACIONES MATEMÁTICAS BÁSICAS

INTRODUCCIÓN

En MicroPython puedes realizar cualquier operación básica: sumas, restas, multiplicaciones o divisiones.

Vamos a realizar una calculadora la cual al ingresar dos cifras realice todas las operaciones básicas.

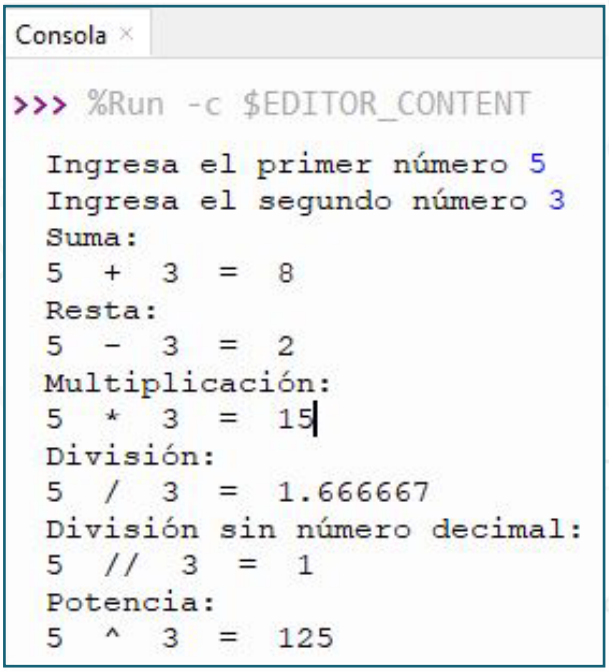
La forma de escribir el código es el siguiente:

DESARROLLO

Para realizar el siguiente código, requerimos 2 variables que el usuario tendrá que otorgar.

```
#Ingresa los dos números a utilizar
x = int (input("Ingresa el primer número "))
y = int (input("Ingresa el segundo número "))

#Suma
z = x + y
print("Suma: ")
#Imprime el resultado:
print( x, " + ", y," = ", z)

#Resta
z = x - y
print("Resta: ")
#Imprime el resultado:
print( x, " - ", y," = ", z)

#Multiplicación
z = x * y
print("Multiplicación: ")
#Imprime el resultado:
print( x, " * ", y," = ", z)

#División
z = x / y
print("División: ")
#Imprime el resultado:
print( x, " / ", y," = ", z)

#División sin número decimal
z = x // y
print("División sin número decimal: ")
#Imprime el resultado:
print( x, " // ", y," = ", z)

#Potencia
z = x ** y
print("Potencia: ")
#Imprime el resultado:

print( x, " ^ ", y," = ", z)
```

A continuación una vista en Thonny del código desplegado en la consola usando el 5 y 3 como ejemplo.



PROYECTO 7: USO DE FUNCIONES

INTRODUCCIÓN

Con ayuda de una función es posible hacer que una parte del programa se ejecute de forma ininterrumpida, ya que son bloques de código que realizan una operación.
La estructura de una función es la siguiente:

```
def NombreDeLaFuncion (x):
        Código que ejecutará
```

Donde:

x : *es la variable que utilizará la función para realizar operaciones.*

def : *indica que estamos a punto de utilizar una función.*

Las funciones requieren:

- Asignación de un nombre
- Variables con las que va a trabajar la función (puede o no requerir)

DESARROLLO

Escribe el siguiente ejemplo en Thonny:

```
#La palabra "def" se utiliza para declarar una
#función
def Calculadora (x,y):
#Los valores entre paréntesis son las variables con
#las que trabaja la función
    #Suma
    z = x + y
    print("\nSuma: ")
    print( x, " + ", y," = ", z)

    #Resta
    z = x - y
    print("Resta: ")
    print( x, " - ", y," = ", z)

    #Multiplicación
    z = x * y
    print("Multiplicación: ")
    print( x, " * ", y," = ", z)

    #División
    z = x / y
    print("División: ")
    print( x, " / ", y," = ", z)

    #División sin número decimal
    z = x // y
    print("División sin número decimal: ")
    print( x, " // ", y," = ", z)

    #Potencia
    z = x ** y
    print("Potencia: ")
    print( x, " ^ ", y," = ", z)

#Ciclo indefinido
while True:

    #Ingresa los dos números a utilizar
    x = int (input("\nIngresa el primer número "))
    #Utilizar el caracter \n agrega un espacio entre
    #renglones
    y = int (input("Ingresa el segundo número "))
    Calculadora (x,y)
```

Continuación

Ciclo while True

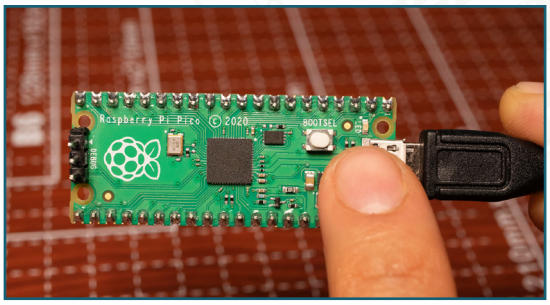
Con *while True* puedes ejecutar porciones de código de forma ininterrumpida como en este caso la función *"Calculadora"*.
Recordando que debe de colocarse al final de la rutina.

PROYECTO 8:
BLINK LED INTERNO PIN 25

INTRODUCCIÓN

En esta ocasión haremos el Hola mundo de la electrónica: Prender un led.

La tarjeta Pi Pico ya incluye un led interno, puedes encontrarlo a un costado del conector USB:



Es igual a los leds grandes de 5mm solamente que en una presentación más pequeña llamada SMD (Dispositivo de Montaje Superficial).

El primer paso será entrar a Thonny y empezar el código con la siguiente línea:

```
from machine import Pin
```

Esta línea es muy importante y se puede leer como: "desde el módulo machine importa la función **Pin".**

Las librerías son colecciones de código predefinido y machine es una función encargada de manejar y acceder a el hardware dentro de nuestra Raspberry Pi Pico.
En este ejemplo se utiliza para acceder a los pines de comunicación del microcontrolador que funcionan como entradas o salidas de datos(I/O).
La siguiente línea en nuestro programa será:

```
import time
```

time es una librería de MicroPython que gestiona todo lo relacionado con el tiempo, en este caso se utilizará para generar retardos.

Para trabajar con el pin I/O requerimos:

- Definir un objeto
- Seleccionar el pin: En nuestro caso usaremos el Pin 25 con conexión al led
- Definir la forma de trabajo: entrada o salida.

DESARROLLO

Escribe el siguiente ejemplo en Thonny:

```
#Definimos librerías
from machine import Pin
import time
#A "led" le asignamos las propiedades del Pin 25 como
#salida
led = Pin(25, Pin.OUT)

while True:
    #Esta línea de código manda un estado alto a la
    #salida
    led.value(1)
    #Esta línea de de código ingresa un retardo de 1
    #segundo
    #antes de ejecutar la siguiente instrucción
    time.sleep(1)
    #Estado bajo a la salida
    led.value(0)
    #Retardo de 0.5 segundos
    time.sleep(0.5)
```

Función *led.value()*

Esta función nos ayuda a asignar un estado al led:

1 -> Led encendido
0-> Led apagado

DESAFÍO

Ahora envía los comandos led.value() desde la consola.
Varía el tiempo de los retardos.

PROYECTO 9: BLINK LED EXTERNO

INTRODUCCIÓN

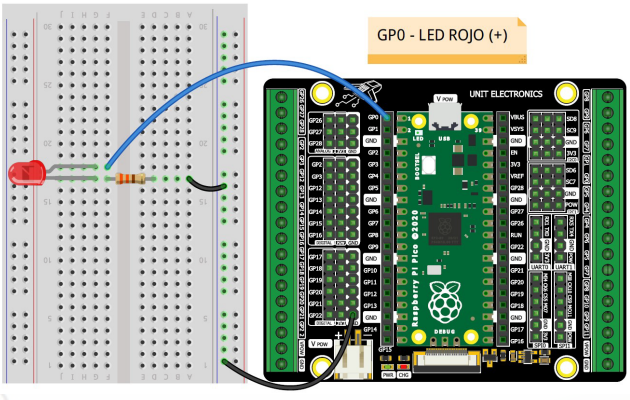
Ahora configuraremos un pin de salida como en la práctica pasada, pero utilizaremos un led externo, la configuración y control es igual que el led interno de la tarjeta, solo cambiará el número de Pin.

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Protoboard**
- **Resistencia 330 ohms**
- **Led rojo 5mm**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN

Identifica el positivo del led y conéctalo al Pin 0 de la Pi Pico, el polo negativo conéctalo a una resistencia a negativo. No olvides conectar el negativo de la protoboard a uno de los pines negativos de la tarjeta.



DESARROLLO

```
#Definimos librerías
from machine import Pin
import time
#"led" se le asigna las propiedades del Pin 0 como
#salida
led = Pin(0, Pin.OUT)

while True:
    #Esta línea de código manda un estado alto a la
    #salida
    led.value(1)
    #Esta línea de de código ingresa un retardo de 1
    #segundo
    #antes de ejecutar la siguiente instrucción

time.sleep(1)
#Estado bajo a la salida
led.value(0)
#Retardo de 1 segundo
time.sleep(1)
```

DESAFÍO

Intercambia el número de Pin utilizado y realiza las conexiones correspondientes para observar qué ocurre.

Polaridad de un led

PROYECTO 10: ENTRADAS DIGITALES (PUSH BUTTON)

INTRODUCCIÓN

Para este proyecto realizaremos un interruptor tipo Push, que proporcionará a la entrada un voltaje de 3.3V.

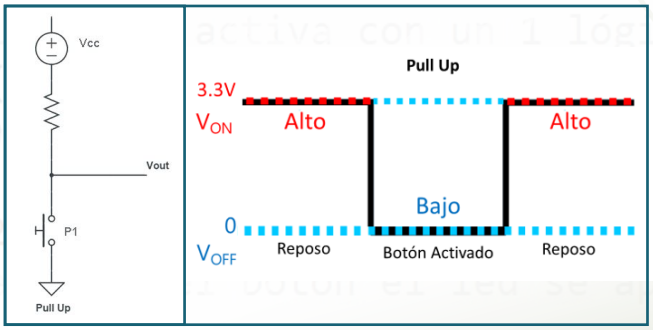
Cada vez que la Pi Pico detecte a la entrada el cambio de voltaje lanzará una alerta por la consola notificándonos que el botón fue presionado.

Una entrada digital de la Raspberry es capaz de detectar el cambio de estado de un voltaje positivo (que él microcontrolador interpreta como un 1 lógico) a 0 V (0 lógico) y viceversa. Esta característica es importante para trabajar con los sensores e interruptores llamados digitales. Cada entrada soporta un voltaje máximo de 3.3V y sobrepasar este voltaje puede causar un daño permanente a la Raspberry.

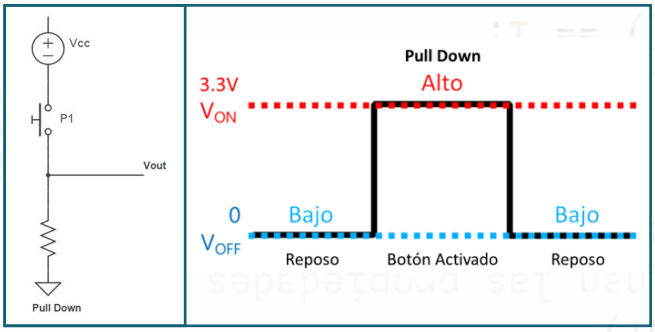
Para conectar los push button es necesario utilizar resistencias de tipo Pull Up y Pull Down. Son resistencias que establecen un estado lógico (1 o 0) de inicio.

RESISTENCIAS PULL UP y DOWN

Las resistencias Pull Up son de cualquier valor (Ω) y se conectan a un voltaje positivo (Vcc). El voltaje a través de la resistencia es igual a Vcc hasta que presionas el interruptor, durante ese momento el voltaje es 0V. Cambiando de estado de 1 a 0 durante el evento.



Las resistencias Pull Down son de cualquier valor (Ω) y se conectan a 0 V(GND). Funciona al revés, el voltaje en la resistencia es de 0V hasta que presionas el interruptor, el estado cambia a Vcc interpretando un cambio de 0 a 1 de manera binaria.



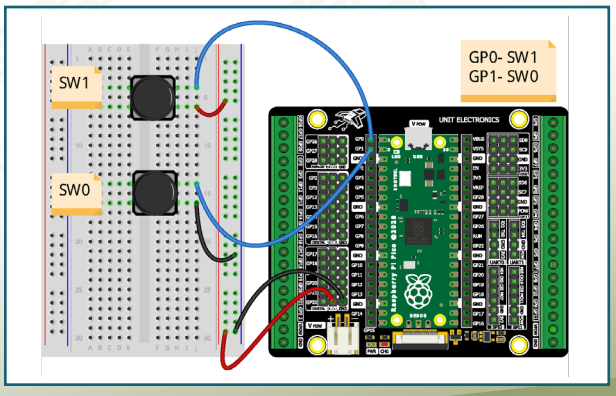
Resistencias Pull Up y Pull Down

La Raspberry Pi Pico posee internamente resistencias Pull Up y Pull Down, que pueden ser activadas a través del software. Físicamente también se puede realizar el circuito.

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Protoboard**
- **Push Button grande de 4 pines x2**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN



DESARROLLO

En este código se configuran dos pines distintos como entradas, uno como Pull Up y otro como Pull Down.

```
#Definimos librerías
from machine import Pin
import time
"""
"button" se le asigna las propiedades del Pin
0,1,2,3,etc
Pin.In se declara como un pin de entrada
Pin.PULL_DOWN declara que esa entrada funcione
como una resistencia
Pull Down
Pin.PULL_UP declara que esa entrada funcione
como una resistencia
Pull Up
"""

button1 = Pin(0, Pin.IN, Pin.PULL_DOWN)
button2 = Pin(1, Pin.IN, Pin.PULL_UP)

while True:
    #En Pull Down el pin se activa con un 1
    #lógico
    if button1.value() == 1:
        print("¡Presionaste el botón 1!")
        #Ingresamos un retardo de 0.5 segundos
        time.sleep(0.5)
    #En Pull Up el pin detecta el interruptor
    #con un 0 lógico
    if button2.value() == 0:
        print("¡Presionaste el botón 2!")
        #Ingresamos un retardo de 0.5 segundos
        time.sleep(0.5)
```

Función IF

- Son porciones de código que se ejecutan sí y solo sí se cumple la condición indicada después de la sentencia ***if,*** las condiciones son las mismas que las mostradas en el ciclo ***while.*** La sintaxis es la siguiente:
  if condición:
  >>>> Función a realizar

  A diferencia de *while* y *for,* la función *if* se ejecuta una sola vez. No olvides agregar la sangría o 4 espacios después de los dos puntos (:).

- Por cada *if* existe un retardo de medio segundo que nos ayuda a que la Pi Pico lea una sola vez el Pin de Entrada, de lo contrario la tarjeta leería muchas veces que el interruptor fue presionado.

DESAFÍOS

Reduce y aumenta el tiempo de retardo en las dos funciones para ver cómo se comporta el interruptor.

¿Qué pasa si eliminas el retardo?

PROYECTO 11: ENTRADAS Y SALIDAS EN CONJUNTO

INTRODUCCIÓN

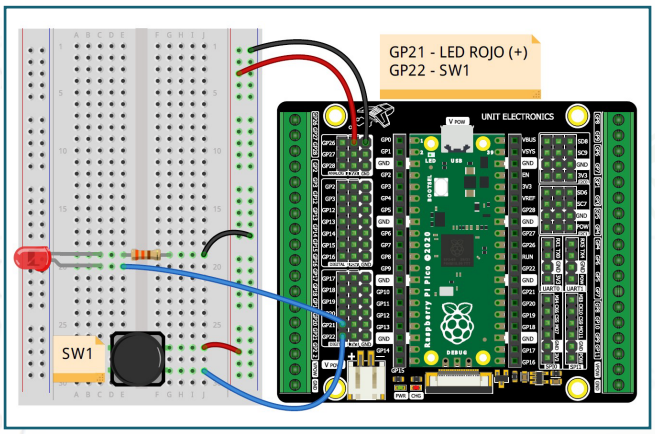
Mientras realices proyectos más grandes y complejos utilizarás más pines de la Raspberry Pi Pico, cada uno de los pines puede ser configurado como entrada o salida de manera independiente.

En esta ocasión se realizará un circuito con una entrada y salida, consta de un Push Button y un led, al apretar el botón se encenderá el led y permanecerá encendido dos segundos.

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Protoboard**
- **Led rojo 5mm**
- **Resistencia 330 ohms**
- **Push Button 4 pines**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN



DESARROLLO

En este código se configuran dos pines distintos uno como entrada en modo Pull Down para el Push Button y otro como salida para el led. Cuando el botón sea presionado se encenderá el led por dos segundos, después se apagará el led.

```
#Definimos librerías
from machine import Pin
import time
#"led" se le asigna las propiedades del Pin 21
como salida
led = Pin(21, Pin.OUT)
#"button" se le asignan las propiedades del Pin
22 como entrada conectada a un #resistor Pull
Down
button = Pin(22, Pin.IN, Pin.PULL_DOWN)

while True:
    #En Pull Down el pin se activa con un 1
    #lógico
    if button.value() == 1:
        led.value(1)
    #Ingresamos un retardo de 2 segundos después
    #de presionar el led
        time.sleep(2)
    #Al dejar de presionar el botón, después de
    #2 segundos el led se apaga
    led.value(0)
```

DESAFÍOS

Enciende el led interno de la Pi Pico con un botón

Haz que el led parpadee mientras presionas el botón y que se apague si no está presionado.

PROYECTO 12: SEMÁFORO CON LEDS

INTRODUCCIÓN

Esta práctica es el inicio de un sistema de tráfico que incluya un semáforo para autos y un semáforo peatonal, ambos sistemas en la vida real trabajan a la par para hacer fluido el tránsito de automóviles y peatones.

El diseño del funcionamiento clásico es:

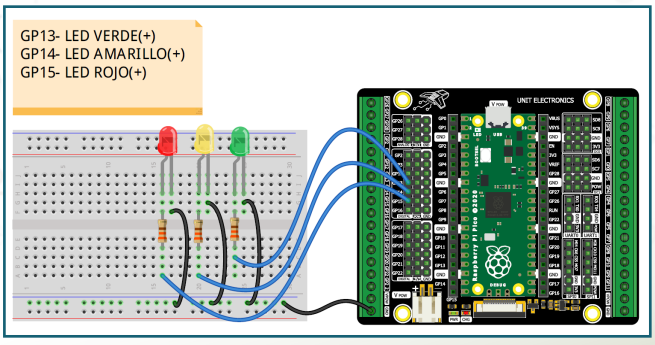
- La luz verde permanece encendida un cierto número de segundos.
- Posteriormente pasa a luz amarilla por un breve tiempo.
- Finaliza con la luz roja.

Este ciclo se repetirá indefinidamente mientras se ejecuta el programa.

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Protoboard**
- **Resistencias de 330 ohms x3**
- **Led Rojo 5mm**
- **Led Verde 5mm**
- **Led Amarillo 5mm**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN



DESARROLLO

En el siguiente código, los leds pasan de estar apagados a encendidos en intervalos de tiempo por medio de la función time.sleep proveniente de la librería time. Este funcionamiento está dentro de un bucle while , comenzando la secuencia de la siguiente forma:

1. Los 3 led apagados
2. Encendido de Led Rojo
3. Encendido de Led Verde
4. Encendido de Led Amarillo
5. Reinicio de ciclo

```
#Definimos librerías
from machine import Pin
import time

#Declaramos los 3 pines que vamos utilizar como
#salidas
#Un pin por cada color del semáforo

led_verde = Pin(13, Pin.OUT)
led_amarillo = Pin(14, Pin.OUT)
led_rojo = Pin(15, Pin.OUT)

#Declaramos los valores iniciales de cada salida
#En este caso se inicia en 0 para evitar que se
#inicien todos encendidos
led_rojo.value(0)
led_amarillo.value(0)
led_verde.value(0)

while True:
    #Luz roja encendida por 5 segundos, verde y
    #amarillo apagadas
    led_rojo.value(1)
    time.sleep(5)

    #Verde encendido por 5 segundos, rojo y amarillo
    #apagado
    led_rojo.value(0)
    led_amarillo.value(0)
    led_verde.value(1)
    time.sleep(5)

    #Amarillo encendido por 2 segundos, verde y rojo
    #apagado
    led_verde.value(0)
    led_amarillo.value(1)
    time.sleep(2)
    #Apagamos la luz amarilla para reiniciar el ciclo
    led_amarillo.value(0)
```

Estados de inicio variable

Los pines led_rojo, led_amarillo y led_verde; se configuran todos con un valor 0. Esto es muy importante ya que asignamos un estado de inicio a los pines de salida.
Además para la utilidad de este proyecto, sería peligroso que el semáforo iniciaran en rojo o verde, es mejor que inicien apagados y encender el rojo inmediatamente.

PROYECTO 13: SISTEMA DE TRÁFICO

INTRODUCCIÓN

Continuando con el proyecto anterior realizaremos un sistema más complejo, se sumará al semáforo de vehículos una alerta sonora para indicar a los peatones que es seguro cruzar.
Este proyecto contará con un botón de activación donde:

- El semáforo realizará su ciclo de manera normal hasta que se active el botón, donde tendrá que continuar el cambio de luces hasta el color rojo.
- Posteriormente dará un tiempo extra para que los peatones puedan transitar, sonando al mismo tiempo un buzzer.
- Finalizado este tiempo continuará el color rojo un par de segundos más por seguridad y seguirá el ciclo de manera normal hasta que el botón sea presionado nuevamente.

Para esta práctica tendremos dos programas ejecutándose de manera simultánea:

- Uno llevará el ciclo del semáforo
- El segundo esperará a que el botón sea presionado

¿Cómo puede ejecutar dos programas al mismo tiempo? ¿Se utilizarán dos módulos Raspberry para este proyecto? No precisamente.

NÚCLEOS DE TRABAJO DE LA RASPBERRY PI PICO

Como se vio al principio del manual, la Pi Pico trabaja con el chip RP2040 que tiene dos núcleos de procesamiento. Esto significa que puede realizar dos tareas distintas al mismo tiempo sin que una tarea influya en el tiempo o ejecución de la otra.
Nos apoyaremos de esta característica con el uso de la librería *_thread*.

```
import _thread
```

La librería _thread genera "hilos" de ejecución, en el caso de la Pi Pico se generan dos hilos cada uno con un programa independiente:

- El hilo principal es donde se ejecuta el control de luces.
- El hilo secundario estará muestreando permanentemente si el botón es presionado, ambos corren al mismo tiempo.

Entre ambos hilos se puede compartir información, pero necesitan un tipo de variable para hacerlo: *globales*.

A diferencia de las **variables locales** que están disponibles para su uso en la sección de código que se declaran, las **variables globales** se visualizan o modifican en cualquiera de los hilos, se declaran de la siguiente manera:

```
global nombre_Variable
```

¿Y cómo se declara un hilo dentro del programa?

Lo primero será crear la función que queremos ejecutar en un segundo proceso. Es posible declarar este hilo en cualquier parte del programa y con el uso de la biblioteca ***_thread*** se indica dónde se inicia esta parte del código.

No se detendrá la ejecución del código principal, se creará un nuevo proceso ejecutando ambos hilos a la vez. Se utiliza la siguiente sintaxis:

- `_thread.start_new_thread(nombre_funcion, ())`

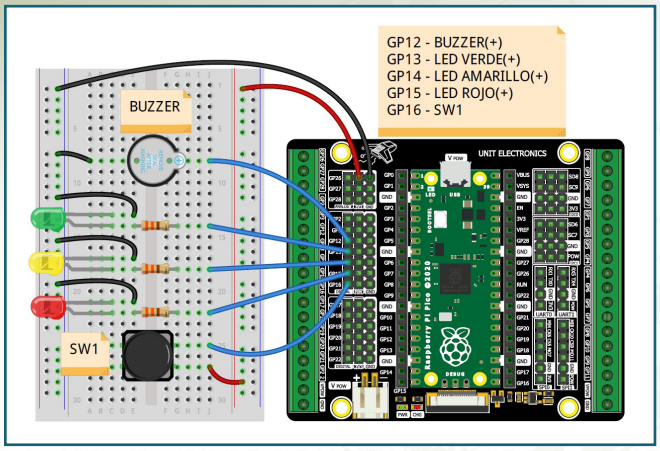
_thread.start_new_thread es la instrucción que ejecuta una función en el segundo núcleo, dentro de los paréntesis externos se define el nombre de la función y los paréntesis internos llevan las variables que serán usadas.

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Protoboard**
- **Buzzer activo**
- **Push button 4 pines**
- **Led Verde 5mm**
- **Led Amarillo 5mm**
- **Led Rojo 5mm**
- **Resistencias 330 ohms x3**
- **Cables dupont**

Continuación »

DIAGRAMA DE CONEXIÓN



DESARROLLO

En el siguiente código, tendremos dos programas ejecutándose de manera simultánea: uno llevará el ciclo del semáforo y otro esperará a que el botón sea presionado para hacer sonar el buzzer.

```
#Definimos librerías
from machine import Pin
import time

#Esta librería nos ayuda a utilizar los 2
#núcleos de la Pi Pico
import _thread

#Declaramos los 4 pines que vamos utilizar como
#salidas
#Un pin por cada color del semáforo y otro para
#el buzzer
led_rojo = Pin(15, Pin.OUT)
led_amarillo = Pin(14, Pin.OUT)
led_verde = Pin(13, Pin.OUT)
buzzer = Pin(12, Pin.OUT)

#Declaramos nuestro pin de entrada
boton = Pin(16, Pin.IN, Pin.PULL_DOWN)

#Declaramos los valores iniciales de cada
#salida
#En este caso se inicia en 0 para evitar que se
#inicien todos encendidos
led_rojo.value(0)
led_amarillo.value(0)
led_verde.value(0)
buzzer.value(0)

#Declaramos una variable global para que se
#ejecute en ambos núcleos
global boton_on

#valor inicial de variable global
boton_on = False

#Función que se ejecutará en el segundo hilo
def activacion_boton_thread():

    #Declaramos la variable global para
    #modificarla
    global boton_on

    #Ciclo infinito
    while True:

    # Si se presiona el boton, boton_on se marca
    #como verdadero
    if boton.value() == 1:
    boton_on = True

#Instrucción que declara que la función
#anterior se ejecutará en el segundo hilo de la
#Pi Pico
_thread.start_new_thread(activacion_boton_thre
ad, ())

#Ciclo infinito
while True:

    #Como solo queremos leer el valor de la
    #variable no se vuelve a declarar
    if boton_on == True:

      #Si el botón está activado se ejecuta el
      #sistema peatonal
      led_rojo.value(1)

      #Alarma auditiva
      for i in range(0, 12):
          buzzer.value(1)
          time.sleep(0.2)
          buzzer.value(0)
          time.sleep(0.2)

      #Modificamos la variable global para
      #darle un valor falso
      global boton_on
      boton_on = False

#Cuando se presione el botón nuevamente
#regresará el botón a Verdadero

#Ciclo que ejecuta los colores del semáforo
led_rojo.value(1)
time.sleep(5)
led_rojo.value(0)
led_amarillo.value(0)
led_verde.value(1)
time.sleep(5)
led_verde.value(0)
led_amarillo.value(1)
time.sleep(2)
led_amarillo.value(0)
```

Continuación »

PROYECTO 13: SISTEMA DE TRÁFICO

Variable global

Al principio de cada función o fragmento de código donde utilizamos la variable *boton_on* se declara que es una variable global, ¿Por qué pasa esto?

En cualquier punto del programa solo están disponibles para lectura, pero no puedes modificarlas. Si quieres asignarles un nuevo valor tienes que volver a declararlas como variables globales al principio de la función. Esto se ejemplifica en la función *while True*

```
while True:
    if boton_on == True:
        led_rojo.value(1)
        for i in range(10):
            buzzer.value(1)
            time.sleep(0.2)
            buzzer.value(0)
            time.sleep(0.2)
        global boton_on
        boton_on = False

    led_rojo.value(1)
    time.sleep(5)

    led_rojo.value(0)
    led_amarillo.value(0)
    led_verde.value(1)
    time.sleep(5)
```

Solo leemos el valor de boton_on , no es necesario volver a declarar la variable.

Como deseamos modificar el valor de boton_on se declara se declara nuevamente como una función global.

PROYECTO 14: PRUEBA DE REFLEJOS

INTRODUCCIÓN

¿Qué tal diseñar tu propio sistema que detecte quien pulso más rápido un botón? Al estilo de los famosos programas de concursos quien apriete el botón más rápido gana el turno. Pues bien, en esta práctica desarrollaremos la base para este sistema.

Para desarrollarlo utilizaremos una nueva librería llamada *random* , esta nueva librería puede generar números aleatorios los cuales utilizaremos para que el banderazo de inicio no sea un tiempo ya establecido.

Esta función asigna dos valores, con los cuales MicroPython calculará un número aleatorio entre ellos:

```
random.uniform(x, y)
```

Donde:

x es el número menor que puede aparecer
y es el número mayor que puede aparecer

Ahora bien, necesitaremos estar al pendiente cuando el interruptor sea presionado. El microcontrolador puede estar haciendo otras tareas, pero en cuanto se presione el botón se tendrá la marca del tiempo.
Para esta tarea se utilizará algo llamado **solicitudes de interrupción** (*IRQ* por sus siglas en inglés) otro concepto nuevo que en esta práctica aprenderemos.

¿Qué son las solicitudes de interrupción IRQ?

Imagina que estás charlando con tus amigos sobre un viaje que acabas de realizar, pretendes contarles desde el inicio de las vacaciones hasta el final, pero a mitad de tu narración un amigo te interrumpe para preguntarte de un momento de la historia en la que se distrajo, tú le cuentas a tu amigo la parte que le interesa y regresas a contar el resto de la historia que hizo falta.

Pues bien, esta relación se puede aplicar a los microcontroladores: la Pi Pico está ejecutando un programa de manera ordenada línea por línea, pero ingresa una orden que interrumpe el ciclo de ejecución, la Pi Pico se dirige a atender esa orden que lo interrumpió y una vez que la finaliza regresa al programa principal a finalizar la ejecución del resto de código que hizo falta.

Esto es a lo que se le llama solicitudes de ***interrupción IRQ*** y las utilizaremos en nuestro programa para calcular el tiempo en que tardas en presionar el botón.

Para declarar una interrupción primero necesitamos construir la función controladora (del inglés *"handler"*) que notificará cuando la interrupción sea activada. La función que se ejecuta en la interrupción se le llama ***"callback"***. Y se ve de la siguiente forma:

```
def funcion_handler (pin):
```

Como observarás se define como una función normal, se le asigna un nombre y entre paréntesis le indicamos que la función se activa por Pin. Dentro de la función tenemos la instrucción:

```
variable.irq(handler = None)
```

- ***variable*** sería el nombre que asignaste al pin donde estará conectado tu botón
- ***irq*** indica que es una interrupción

handler = None desactiva la interrupción, de lo contrario MicroPython considerará que la interrupción siempre está siendo activada.

La última línea de código ingresamos la instrucción:

```
variable.irq (trigger, handler = funcion_
handler)
```

Donde:

- ***variable*** es el nombre que se asigna al pin donde estará conectado el botón.
- ***irq*** indica que es una interrupción.
- ***trigger*** señalaremos si se activara la interrupción con flanco de subida o de bajada
- ***handler*** define la función que se ejecutará.

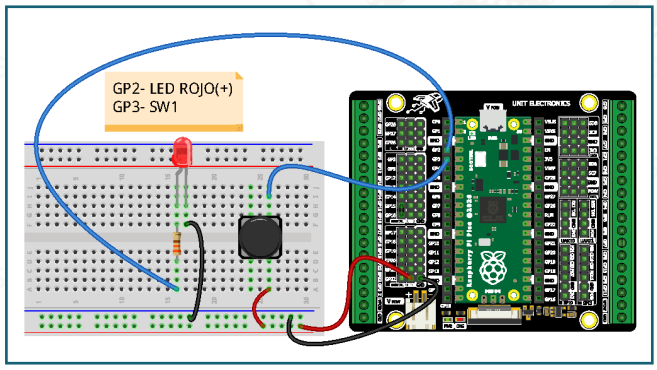
Continuación »

PROYECTO 14: PRUEBA DE REFLEJOS

MATERIAL

- Raspberry Pi Pico
- Tarjeta de conexiones para Pi Pico
- Protoboard
- Resistencia 330 ohms
- Led rojo 5mm
- Cables dupont

DIAGRAMA DE CONEXIÓN



DESARROLLO

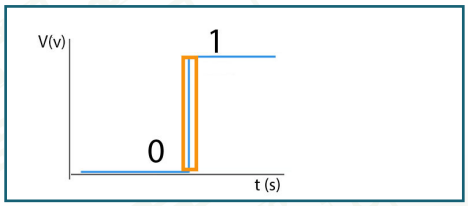
Tendremos al inicio un led encendido por un tiempo aleatorio, en cuanto se apague el led debes de presionar un botón y el programa te devolverá el tiempo que tardaste en apretarlo. Prueba tus reflejos y sincroniza tu vista con tu dedo.

```
#Definimos librerías

from machine import Pin
import time

#Librería que genera números aleatorios:
import random

#Definimos un puerto de entrada y uno de salida
led = Pin(2, Pin.OUT)
boton = Pin(3, Pin.IN, Pin.PULL_DOWN)

#Función Callback si se activa la interrupción:
def control_boton(pin):
    #Desactivamos la interrupción:
    boton.irq(handler=None)
    #Función que calcula el tiempo transcurrido entre
    #ambas variables:
    timer_reaction = time.ticks_diff(time.ticks_ms(),
    timer_start)
    #Imprimimos el tiempo de reacción transcurrido
    print("Tiempo de reacción: " + str(timer_reaction)
    + " milisegundos")

print("\nIniciamos!!!")

#Encendemos el led, señal de que inicia el juego
led.value(1)

#Usamos la función random para generar un retardo
#indeterminado
time.sleep(random.uniform(5, 10))
print("\n¡Presiona ahora!")

#Apagamos el led, señal de que presiones el botón
led.value(0)

#Guardamos el tiempo en que se apago el led
#será nuestro tiempo inicial  de la prueba
timer_start = time.ticks_ms()

#Llamamos a la interrupción
boton.irq(trigger=Pin.IRQ_RISING,
handler=control_boton)
"""
Para correr de nueva cuenta el programa presiona el
botón de "Stop"
y vuelve a ejecutar el script
"""
```

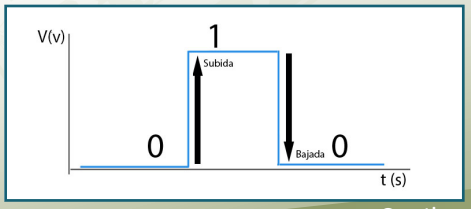
Datos relevantes

Flancos de subida y de bajada

En la forma clásica los estados de una señal digital son el 0 y 1, donde 1 representa un valor de tensión positivo y 0 con la masa del circuito (GND). Entre un estado y otro existe un flanco, el cual marca la transición, pasando de un estado bajo a un estado alto.



Un flanco de subida (rising) es el que pasa de estar en nivel bajo a estar en nivel alto, por otro lado el flanco de bajada (falling) pasa de un nivel alto a bajo.



Continuación »

La interrupción que usamos en el código se ejecuta dependiendo del flanco. En la última línea del código se ejecuta la interrupción y dentro del paréntesis se especifica lo siguiente:

```
trigger=Pin.IRQ_RISING
```

Ese comando indica que la interrupción se ejecutará cuando exista un disparo (trigger) ocasionado por **un flanco positivo (subida)** (Pin.IRQ_RISING) si quisiéramos que la interrupción se disparará por flanco negativo **(bajada)** tendrías que definirlo así:

```
trigger=Pin.IRQ_FALLING
```

También puede dispararse con un flanco positivo y negativo, en ese caso el comando sería:

```
trigger= Pin.IRQ_RISING | Pin.IRQ_FALLING
```

Función *print()*

En la función print() podemos concatenar o unir varias cadenas de caracteres para imprimir una sola cadena esto se puede hacer dos formas usando un símbolo de coma (,) o un símbolo de más (+)

```
a= "Hola"
b= " mundo"
print(a, b)
print(a + b)
```

Si ejecutas este código con ambas funciones obtendrás el mismo resultado.

PROYECTO 15:
PRUEBA DE REFLEJOS PARA 2 JUGADORES

INTRODUCCIÓN

Ya observamos cómo es posible a través de las interrupciones generar un programa que mida tu tiempo de reacción, ahora ¿Qué tal agregar dos interrupciones para tener dos concursantes enfrentándose por saber quién tiene mejores reflejos?

Pues bien, la Raspberry tiene en su interior más de una interrupción y como ya se mencionó declararemos dos interrupciones, observarás que el funcionamiento del programa es muy parecido al anterior:

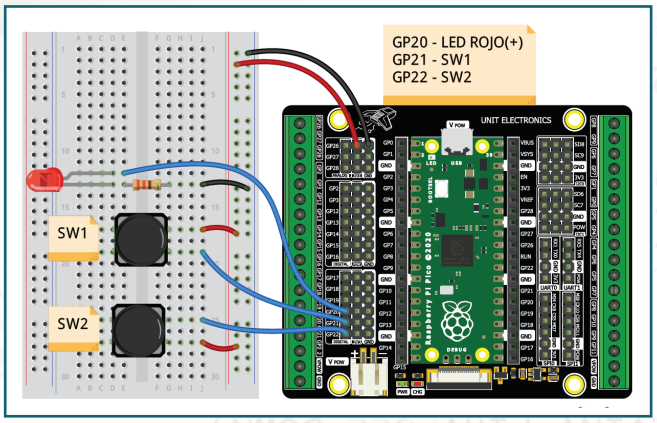
- Tienes dos pines de entrada que activan una interrupción al detectar un flanco positivo.
- Estas dos interrupciones, independientes se pueden activar a la par, pero con ayuda de una variable descubriremos cuál fue la primera en activarse.

Utilizarás dos botones, uno para cada participante identificados como jugador Izquierdo y Derecho.
Al momento de realizar tus conexiones verifica que los botones estén bien localizados para saber cuál es el Jugador Izquierdo y cual el Jugador Derecho.

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Protoboard**
- **Resistencia 330 ohms**
- **Led rojo 5mm**
- **Cables dupont**

DIAGRAMA DE COMUNICACIÓN



DESARROLLO

```
#Definimos librerías
from machine import Pin
from time import sleep, ticks_ms
import random

#Definimos nuestras entradas y salida, las entradas
#están configuradas como Pull Down
led = Pin(20, Pin.OUT)
boton_derecho = Pin(21, Pin.IN, Pin.PULL_DOWN)
boton_izquierdo = Pin(22, Pin.IN, Pin.PULL_DOWN)

#Variable de apoyo para saber quien presionó el botón
boton_presionado = None

#Función callback para activar la interrupción:
def funcion_interrupcion(pin):

    #Desactivamos interrupción 1
    boton_derecho.irq(handler=None)
    #Desactivamos interrupción 2
    boton_izquierdo.irq(handler=None)
    #Definimos la variable como global para poder
    #modificarla
    global boton_presionado
    #La variable "boton_presionado" almacenará el
    #valor del pin donde se
    #produjo primero la interrupción
    boton_presionado = pin

print("\nIniciamos!!!")

#Encendemos el led, señal de que inicia el juego
led.value(1)

#Usamos la función random para generar un retardo
#indeterminado
sleep(random.uniform(5, 10))
print("\n¡Presiona ahora!")

#Apagamos el led, señal de que presiones el botón
led.value(0)
```

Continuación »

PROYECTO 15: PRUEBA DE REFLEJOS PARA 2 JUGADORES

```
#Definimos las 2 interrupciones del programa, llevan
#a la misma función

#Se definen como flancos de subida
boton_izquierdo.irq(trigger     =     Pin.IRQ_RISING,
handler= funcion_interrupcion)
boton_derecho.irq(trigger = Pin.IRQ_RISING, handler=
funcion_interrupcion)

#Mientras esperamos a que una de las interrupciones
#se dispare no hacemos nada
while boton_presionado is None:
    sleep (1)

#Si la interrupción es por Botón Derecho se imprime
#en la consola al ganador
if boton_presionado is boton_derecho:
    print("Felicidades: Jugador DERECHO")

#Si la interrupción es por Botón Izquierdo se imprime
#en la consola al ganador
elif boton_presionado is boton_izquierdo:
    print("Felicidades: Jugador IZQUIERDO")
```

funcion_interrupcion

Las interrupciones tienen la función asociada: handler.Esta función puede contar con más de una interrupción, en este proyecto se contaron con dos.

El siguiente es un ejemplo con dos funciones handler:

```
#Definimos librerías
from machine import Pin
#Definimos nuestras entradas y salida, las entradas
#están configuradas como Pull Down
boton_derecho = Pin(21, Pin.IN, Pin.PULL_DOWN)
boton_izquierdo = Pin(22, Pin.IN, Pin.PULL_DOWN)

#Función callback para activar la interrupción:
def funcion_interrupcion_izq(pin):
    #Desactivamos interrupción 1
    boton_izquierdo.irq(handler=None)
    print("Izquierda")

def funcion_interrupcion_der(pin):
    #Desactivamos interrupción 2
    boton_derecho.irq(handler=None)
    print("Derecha")
```

```
#Definimos las 2 interrupciones del programa
#cada interrupción llama a una función distinta

#Se definen como flancos de subida
boton_izquierdo.irq(trigger     =     Pin.IRQ_RISING,
handler=funcion_interrupcion_izq)

boton_derecho.irq(trigger     =     Pin.IRQ_RISING,
handler=funcion_interrupcion_der)
```

DESAFÍO

Realiza el mismo programa, pero indica cuanto tiempo tardo el ganador en presionar el botón.

PROYECTO 16: SENSOR DE PRESENCIA

INTRODUCCIÓN

Los sensores son muy importantes en la programación de microcontroladores, ya que son dispositivos con los que podemos obtener variables físicas y procesarlas a un nivel que la Pi Pico pueda comprender: **un voltaje.**

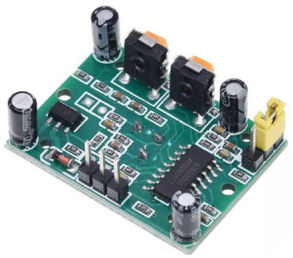
SENSOR PIR

Para esta práctica usaremos un sensor PIR ,el cual es la base en los sistemas de luz automática y de alarmas, debido a que detecta la presencia por medio del movimiento con base a la radiación infrarroja en un área determinada.

Hay una gran variedad de sensores PIR en el mercado, nosotros utilizaremos el Sensor

PIR HC-SR501.

Top View

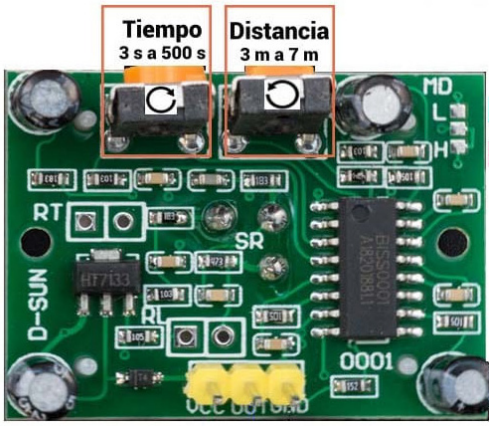


Bottom View

FUNCIONAMIENTO DEL HC-SR501 PIR

El HC-SR501 al detectar movimiento entrega una señal de voltaje de 3.3V a la salida adecuada para los pines de la Raspberry Pi Pico; a pesar de que el movimiento sea constante no se activa todo el tiempo.

El PIR cuenta con dos perillas de ajuste debajo del sensor, con ellas podremos ajustar el Tiempo y Distancia de trabajo.



Estas perillas son importantes, ya que hay un tiempo de retardo antes de volver a detectar movimiento puede ajustarse desde 3 segundos hasta 500 segundos.

Por ejemplo con 10 segundos:

- El sensor detectará una presencia y activará la salida.
- Permanecerá así por 10 segundos
- Si existen más movimientos, no se detectarán sino hasta pasados el tiempo configurado.
- Después de ello se activará con un nuevo movimiento.

La otra perilla ajusta la distancia de detección que va desde los 3 m hasta 7 m a un ángulo de detección de 110°.

Lo último que debes de saber de este módulo, es que cuenta con dos modos de configuración que puedes activar mediante los pines que se encuentran a un costado del sensor marcados como H y L.

L
Disparo Único

H
Disparo Repetido

Continuación

JUMPER L : 1 Solo Disparo, cuando ocurre una detección de movimiento (el cual llamaremos 'evento'), la salida del sensor se activa durante el tiempo que se haya ajustado a través del potenciómetro correspondiente.

Por ejemplo, que el tiempo de activación es de 60 segundos. Si durante esos 60 segundos ocurre un segundo evento, éste no será considerado.

JUMPER H: Disparos repetitivos, cada evento será detectado generando un nuevo tiempo de activación. Con 60 segundos al ocurrir el primer evento, la salida se activa.

Si transcurridos 30 s ocurre un segundo evento, entonces se sumarán 60 s al tiempo transcurrido, dando un total de 90 s continuos con la salida activa. Y así, cada evento adicional, sumará un tiempo de 60 s.

Pasados 3s el dispositivo regresa a su funcionamiento normal.

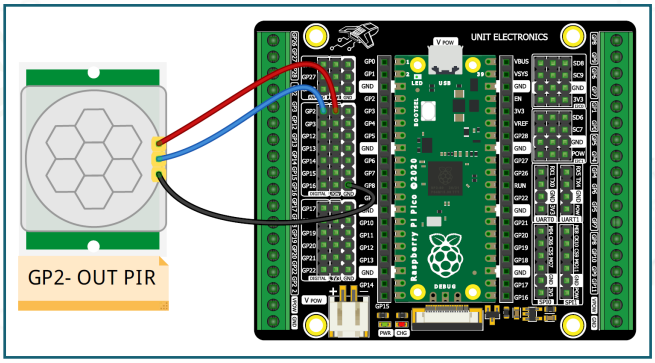
MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Sensor PIR HC-SR501**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN

Los pines de conexión son los siguientes:

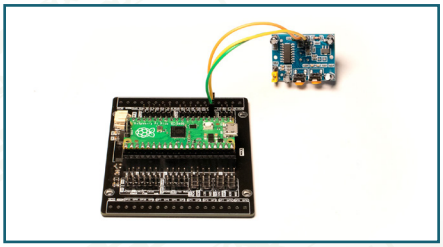
- GND se conecta a GND de nuestra tarjeta adaptadora Pi Pico.
- TTL o OUT es la salida del sensor y se conecta directamente a uno de los pines digitales de la Pi Pico.
- VCC es necesario conectarlo a uno de los pines de POW de la tarjeta, esto es porque requiere al menos 4V para funcionar.



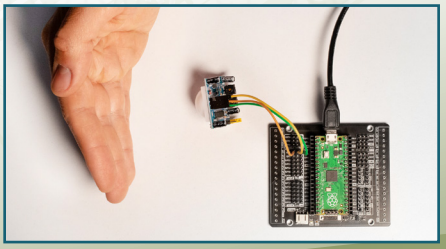
DESARROLLO

- Definiendo un pin de entrada para leer el sensor.
- Y una interrupción

Ejecuta el siguiente código en la Pi Pico, arma el circuito del lado del BOTTOM VIEW. Coloca el sensor de forma opuesta a ti como se observa en la imagen y trata de apuntar el módulo a una dirección en donde no haya movimiento para que sea estable.

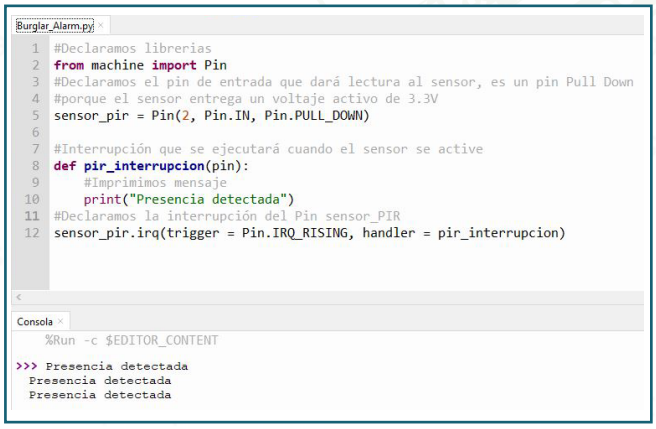


Compila el programa, pasa tu mano frente al sensor y observa lo que ocurre en la consola.



Continuación »

```
#Declaramos librerías
from machine import Pin
#Declaramos el pin de entrada que dará lectura al
#sensor, es un pin Pull Down
#porque el sensor entrega un voltaje activo de 3.3V
sensor_pir = Pin(2, Pin.IN, Pin.PULL_DOWN)

#Interrupción que se ejecutará cuando el sensor se
#active
def pir_interrupcion(pin):
    #Imprimimos mensaje
    print("Presencia detectada")

#Declaramos la interrupción del Pin sensor_PIR
sensor_pir.irq(trigger = Pin.IRQ_RISING, handler =
pir_interrupcion)
```



Manejo de Retardos

En el programa no se incluyen retardos, se realiza internamente en el sensor y es ajustable con la perilla de tiempo.

DESAFÍOS

Cambia los ajustes de Tiempo, Distancia y tipo de Disparo del módulo para observar cómo reacciona ante cada cambio.

PROYECTO 17:
SISTEMA DE LUCES AUTOMÁTICO

INTRODUCCIÓN

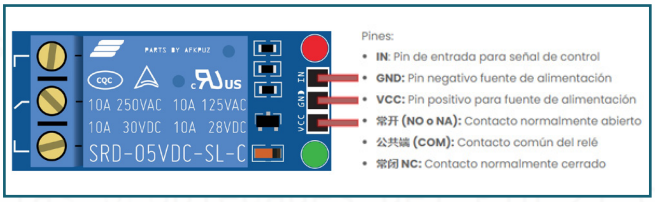
Los sensores PIR se aplican en diversos ambientes: sistemas de luces automáticas en edificios y reducir el gasto de luz. En esta práctica desarrollaremos ese sistema con un sensor de presencia PIR y un relevador.

Módulo Relevador 5V

Relevador SRD-05 VDC a 127 VAC

Un relevador es un interruptor de voltaje, útil cuando quieres controlar luces o motores que no trabajan con Corriente Directa (DC), sino con Corriente Alterna (AC) a 127VAC.

Usaremos en este proyecto el Módulo relevador 1 canal 5V y sus pines son:



Adicional usaremos un led que puedes reemplazar por un foco de 120V, si tienes conocimientos de electricidad.

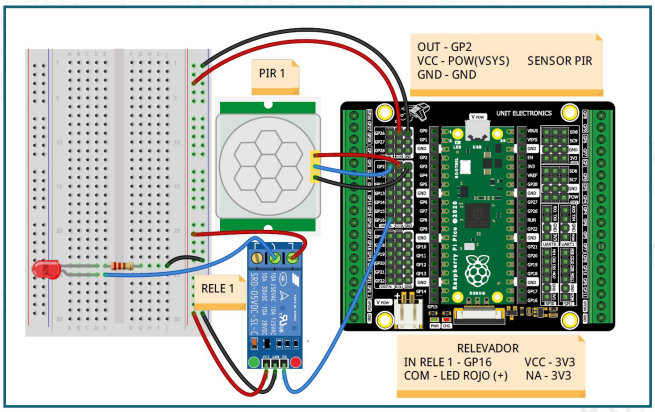
MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Protoboard**
- **Sensor PIR HC-SR501**
- **Modulo relevador 1 canal 5V**
- **Led rojo 5mm**
- **Resistencia 330 ohms**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN

Para el relevador tendrás que conectar

- La terminal Normalmente Abierta (NA) a 3.3V
- El común al pin positivo del led, no olvides agregar la resistencia de protección.
- El pin 16 de la Pi Pico como señal de activación para el relevador.
- El pin 2 de la tarjeta será la entrada para el sensor PIR.



DESARROLLO

El funcionamiento del programa es el siguiente:

- El microcontrolador se mantiene en un ciclo infinito manteniendo el relevador apagado.
- Si recibe una interrupción disparada por el sensor PIR energiza el relevador y lo mantendrá activado por un tiempo específico.
- Cuando acabe este tiempo el relevador apagará la luz hasta una nueva detección.

Al igual que la práctica anterior coloca el sensor opuesto a ti, de manera que tus movimientos no lo activen, pasa tu mano frente a él y observa lo que ocurre.

```
#Importamos librerías
from machine import Pin
import time

#Definimos el pin de entrada para el sensor
sensor_pir = Pin(2, Pin.IN, Pin.PULL_DOWN)

#Pin de salida para el relevador
rele = Pin(16, Pin.OUT)

#Valor inicial del relevador
rele.value(1)
```

Continuación »

```
"""
Funcionamiento del relevador:

1 -> Rele desactivado
0 -> Rele activado
"""
#Interrupción que se ejecutará cuando el sensor se
#active
def pir_interrupcion(pin):
    print("Presencia detectada")
    rele.value(0)
    #Activamos el relevador
    time.sleep(10)
    #10 segundos de activación del rele
    rele.value(1)
    #Apagamos el rele

    #Declaramos la interrupción del Pin sensor_PIR
    sensor_pir.irq(trigger = Pin.IRQ_RISING, handler=
    pir_interrupcion)

while True:
    #Mientras se activa la interrupción el rele
    #permanece desactivado
    rele.value(1)
```

Relevador SRD-05 VDC

El relevador SRD-05 VDC tiene un funcionamiento distinto, ya que con un voltaje positivo el relevador permanece apagado, mientras que con 0V el relevador se activa.
En el mercado existen diferentes relevadores, revisa cuál es su forma de trabajo antes de aplicarlo a tus proyectos.

PROYECTO 18:
SISTEMA DE ALARMA VISUAL

INTRODUCCIÓN

Ya se exploró el uso del sensor PIR como sistema de luces automático, en esta práctica utilizaremos nuevamente el PIR HC-SR501 y la Raspberry pero ahora como alarma con luces led, es decir, al detectar una presencia cercana la Pi Pico activa un led que funcionará como alarma visual.

Este es el principio de las alarmas antirrobo comerciales, la luz puede alertarte que un intruso ha entrado a un área vigilada.

Para este proyecto se utilizará la función toggle()para hacer parpadear el led. Sin esta función se declara de la siguiente forma:

```
for i in range (25):
    led.value(1)
    time.sleep(1)
    led.value(0)
    time.sleep(1)
```

Lo cual realizará un ciclo que ejecuta 25 veces un led encendiendo y apagando cada segundo.

Ahora lo sustituiremos con:

```
for i in range (50):
    led.toggle()
    time.sleep(1
```

Que realizaría lo mismo pero con una menor cantidad de código.

Lo que toggle() realiza es alternar entre estados, si el Pin está en 1 lógico lo convierte en 0 lógico y viceversa. Esta instrucción está dentro de la librería Pin y funciona de la siguiente forma:

```
from machine import Pin
variable = Pin(x, Pin.OUT)
variable.toggle()
```

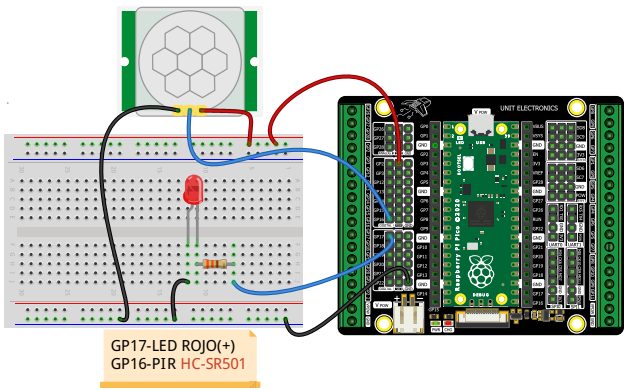
Donde:

variable = nombre que asignaremos al pin de salida.
x = número de puerto que usaremos como salida.

MATERIAL

- Raspberry Pi Pico
- Tarjeta de conexiones para Pi Pico
- Sensor PIR HC-SR501
- Resistencia 330 ohm
- Led rojo 5mm
- Protoboard
- Cables dupont

DIAGRAMA DE CONEXIÓN



DESARROLLO

El funcionamiento será el siguiente:

- La Raspberry mantiene en ciclo de encendido y apagado de un led de manera continua con tiempo de 2 s.
- Se mantiene el ciclo anterior hasta que recibe una interrupción proveniente del sensor, en ese momento se ejecuta un ciclo donde el led prende y apaga a mayor velocidad indicando que se detectó un movimiento.
- Si no se detecta nuevamente movimiento el led parpadea de manera normal como en el primer paso.

Continuación »

```
#Declaramos librerías
from machine import Pin
import time

#Definimos el pin de entrada para el sensor
sensor_pir = Pin(16, Pin.IN, Pin.PULL_DOWN)

#Pin de salida para el led
led = Pin(17, Pin.OUT)

#Valor inicial del led
led.value(0)

#Interrupción que se ejecutará cuando el sensor
#se active
def pir_handler(pin):
    print("Presencia detectada")
    #Ciclo For para hacer parpadear el led, se
    #repite 50 veces
    for i in range (50):
       #Instrucción para cambiar el estado del
       #led
       led.toggle()
       #Instrucción para pausar 100mS la
       #compilación del código
       time.sleep_ms(100)

#Activamos la interrupción
sensor_pir.irq(trigger    =    Pin.IRQ_RISING,
handler=  pir_handler)

#Menú principal
while True:

    #Instrucción para cambiar el estado del led
    led.toggle()
    time.sleep(2)
```

Funciones para Retardos

Las funciones de retardo pueden variar en tiempo:

- time.sleep(): a segundos
- time.sleep.ms(): con milisegundos

PROYECTO 19:
ALARMA DE PRESENCIA CON SIRENA

INTRODUCCIÓN

Para finalizar el uso del sensor PIR agregaremos una sirena con el buzzer que incluye tu kit de Raspberry Pi Pico.
El funcionamiento del programa será el mismo, pero dentro del ciclo For para el encendido del led, añadiremos otro ciclo For para el buzzer, de la siguiente manera:

```
for i in range (50):
    led.toggle()
    for j in range (25):
        buzzer.toggle()
        time.sleep_ms(3)
```

- El primer ciclo For se ejecuta normalmente cambiando el estado del led de encendido a apagado o viceversa.
- Después ingresa al ciclo For interio r donde se cambia de estado al Pin del Buzzer cada 3 milisegundos, ejecutándose 25 veces hasta culminar el proceso.
- Al término del segundo For, regresa al ciclo exterior para ejecutarse una segunda vez repitiendo el proceso anterior de nueva cuenta. Esto lo realizará 50 veces hasta culminar de ejecutar el primer For

Pareciera que es un proceso muy tardado, pero recuerda que todas las instrucciones se ejecutan en unos pocos milisegundos.

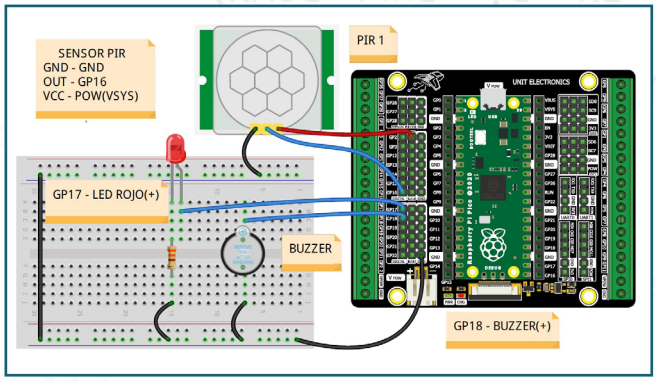
Ciclos o Bucles Anidados

Al incluir un Ciclo dentro del bloque de otro Ciclo se le conoce en programación como Ciclos o Bucles Anidados. El ciclo externo comienza su ejecución hasta llegar al ciclo interno, mientras que el ciclo externo no avanza hasta que el ciclo interno termina.

MATERIAL

- Raspberry Pi Pico
- Tarjeta de conexiones para Pi Pico
- Sensor PIR HC-SR501
- Resistencia 330 ohms
- Led rojo 5mm
- Buzzer Activo
- Protoboard
- Cables dupont

DIAGRAMA DE CONEXIÓN



DESARROLLO

Recuerda colocar el sensor PIR opuesto a ti, para evitar que se active con tus movimientos de forma constante.

```
#Importamos librerías
from machine import Pin
import time

#Definimos el pin de entrada para el sensor
sensor_pir = Pin(16, Pin.IN, Pin.PULL_DOWN)

#Definimos pines de salida
led = Pin(17, Pin.OUT)
buzzer = Pin(18, Pin.OUT)

#Interrupción que se ejecutará cuando el sensor
#se active
def interrupcion_PIR (pin):
    print("Presencia detectada")
```

Continuación »

```
    #Ciclo para prender y apagar rápidamente
    #el led
    for i in range (50):
        #Instrucción que cambia de estado
        #el pin del Led
        led.toggle()

        for j in range (25):
           #Instrucción que cambia de estado
           #el pin del Buzzer
           buzzer.toggle()
           #Retardo de 3 milisegundos
           time.sleep_ms(3)

#Declaramos la interrupción del Pin
#sensor_PIR
sensor_pir.irq(trigger = Pin.IRQ_RISING,
handler = interrupcion_PIR)

while True:

       #Mientras se activa la interrupción
       #el led parpadeará cada 2 segundos
       led.toggle()
       time.sleep(2)
```

Ciclos For

En el For anidado que controla el buzzer no utiliza la letra i para el incremento Utiliza la letra j porque cada incremento es independiente.
Cada ciclo lleva una cuenta distinta.

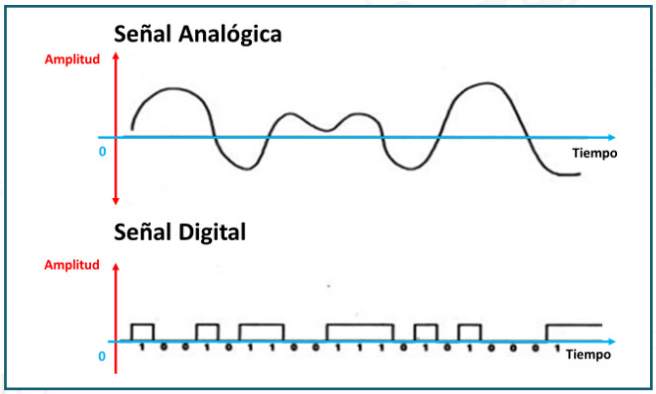
PROYECTO 20: CONVERTIDOR ADC CON POTENCIÓMETRO

INTRODUCCIÓN

En esta práctica diseñaremos un medidor de voltaje a través de un potenciómetro, para ello trabajaremos con el Convertidor Analógico Digital (ADC) que toma la señal analógica y la convierte en una señal digital que el microcontrolador puede interpretar.

SEÑALES ANALÓGICAS

Una señal analógica es una señal que puede tomar una gran cantidad de valores (0 V a 3.3V para Pi Pico) que cambian de forma continua.



Para trabajar con estas señales, el microcontrolador posee un módulo interno: Convertidor Analógico Digital (ADC), posee dos características importantes: **resolución y número de canales.**

RESOLUCIÓN ADC

La resolución es la cantidad de bits que el convertidor genera en cada lectura analógica, por ejemplo, la Pi Pico tiene una resolución de **12 bits,** lo cual significa que el convertidor entrega valores desde **0** hasta **4095** convertido a números decimales.

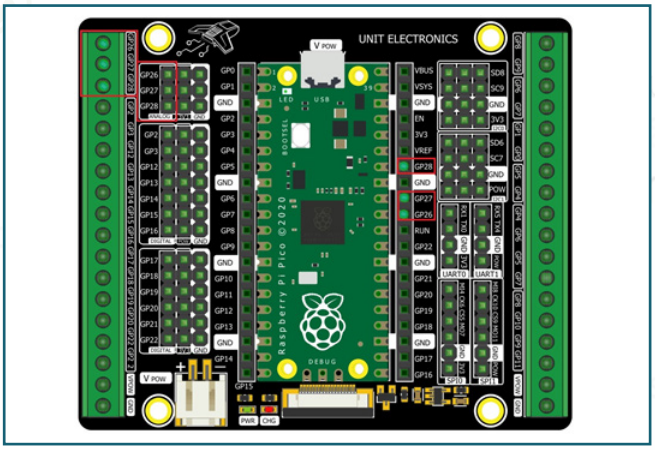
Pero estamos programando en MicroPython con una resolución de **16 bits,** por lo que el ADC puede entregar valores desde **0** hasta **65,535.**

NÚMERO DE CANALES

Es la cantidad de señales que puede aceptar y convertir a la vez. En el caso de la Pi Pico tiene 3 canales:

- GP26 (canal 0)
- GP27 (canal 1)
- GP28 (canal 2)

Existe un cuarto canal que está conectado a un sensor de temperatura interno.



Además introduciremos una nueva librería importada desde machine: ADC, se declara:

```
from machine import ADC
```

Esta librería es la que incluye los métodos necesarios para configurar el Convertidor Analógico Digital de la Pi Pico, para declarar un pin como analógico se realiza lo siguiente:

```
variable = ADC(x)
```

Donde:

variable: Es el espacio para asignar los atributos del ADC.

x: Es el canal o Pin Analógico que queremos leer, por ejemplo, para el Pin 26 podemos definirlo como *ADC(26)* o *ADC(0)* ya que el pin 26 corresponde al canal 0 del convertidor ADC, ambas formas son válidas.

Usando ADC requerimos recibir datos a 16 bits, para ello se requiere del método read_u16() el cual toma la lectura analógica y retorna un número entero.

Continuación »

Posteriormente encontrarás que se realizan dos operaciones matemáticas dentro del código, esta es la famosa relación de "La regla de 3", si al obtener 3.3V obtenemos la lectura 65,535 ¿Qué voltaje obtendremos con una medición distinta?

$$\frac{65535 \rightarrow 3.3}{X \rightarrow voltaje}$$

$$voltaje = \frac{(X)(3.3)}{65535}$$

- Dividimos el voltaje máximo que podemos obtener (3.3V) entre el número de bits máximo (65535) .
- El valor obtenido se multiplica por la lectura del ADC (X en la ecuación).
- El resultado final es el voltaje que circula a través del potenciómetro.

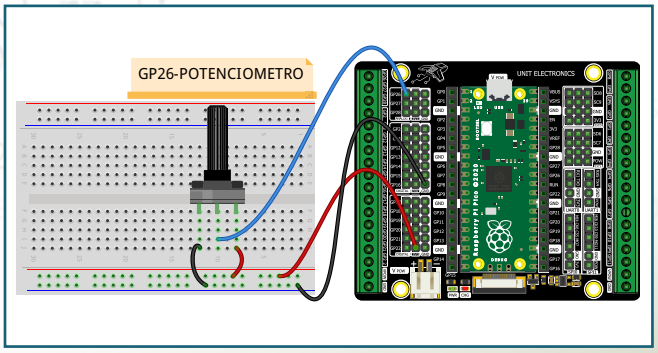
En el código viene expresado de la siguiente manera:

```
conversion_factor = 3.3 / (65535)
voltaje = potenciometro.read_u16()
* conversion_factor
```

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Potenciómetro 10K ohms**
- **Protoboard**
- **Cables conexión**

DIAGRAMA DE CONEXIÓN



DESARROLLO

Compila el siguiente código en Thonny, una vez que lo ejecutes mueve la perilla del potenciómetro y observa que valores obtienes en la Consola, si cuentas con un multímetro puedes realizar mediciones de voltaje para comparar el valor que obtienes.

```
#Añadiremos la nueva librería ADC
from machine import Pin, ADC
import time


#Asignaremos a Potenciómetro la lectura ADC del
#pin 26
potenciometro = ADC(26)

#Constante de conversión de ADC a Voltaje
conversion_factor = 3.3 / (65535)


while True:
    #¿Cuántos bits obtienes de lectura?
    print("\nValor en bits: ",
    potenciometro.read_u16())

    #Ecuación para obtener la lectura de voltaje
    voltaje = potenciometro.read_u16() *
    conversion_factor

    #Imprimimos el voltaje leído en la consola
    print(voltaje, "V")

    #Cada 0.5s se realizará una nueva lectura
    time.sleep(0.5)
```

Continuación »

PRÁCTICAS

Aplicación de la "Regla de 3"

Este término se refiere a una relación matemática muy útil, así puedes obtener relaciones entre el ADC y la variable que estás midiendo.

Valor 0Ω en el Potenciómetro

Lo normal es que nunca obtengas una medición de 0 V por más que lleves a la perilla al mínimo, esto es porque internamente el potenciómetro nunca llega a 0 ohms,
Debido al material eléctrico con el que se construyen.

DESAFÍOS

¿Conoces la ley de ohm?

Si es así prueba tus conocimientos desarrollando un medidor de resistencia, en lugar de medir voltaje mide la resistencia variable del potenciómetro

Continuación »

PROYECTO 21: TERMÓMETRO AMBIENTAL

INTRODUCCIÓN

El microcontrolador de la Raspberry Pi Pico posee internamente un sensor de temperatura conectado al canal 4 del ADC. Al variar la temperatura varía el voltaje de salida.

De igual forma que la práctica anterior se aplica una Regla de Tres para obtener el factor de conversión:

```
conversion_factor = 3.3 / (65535)
voltaje = potenciometro.read_u16() * conversion_factor
```

Pero queremos obtener temperatura, no voltaje. Para ello utilizamos la ecuación especificada para el sensor de temperatura de la Pi Pico:

```
temperatura = 27 - (voltaje - 0.706)/0.001721
```

Esta ecuación viene indicada en la datasheet u hoja técnica del RP2040 y es específica para este sensor.

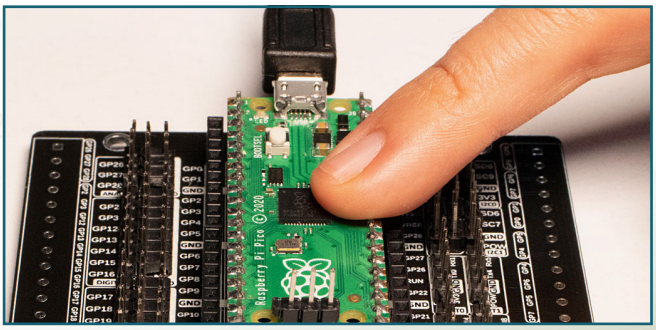
MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**

DESARROLLO

Compila el siguiente código en Thonny, no necesitas ningún componente externo para medir la temperatura.
Las primeras mediciones serán la temperatura ambiente, también puedes tocar o acercar objetos al chip RP2040 y observar como la temperatura varía.

Ten cuidado al medir temperaturas fuera del rango de -20°C a 70°C para no dañar este u otros componentes de la placa.



```
#Añadiremos la nueva librería ADC
from machine import Pin, ADC
import time

#Asignaremos a sensor_temp la lectura ADC
#del canal 4
sensor_temp = ADC(4)

#Constante de conversión de ADC a Voltaje
conversion_factor = 3.3 / (65535)

while True:
    #Ecuación para obtener la lectura de
    #voltaje
    lectura = sensor_temp.read_u16() *
    conversion_factor

    #Ecuación para convertir el voltaje en
    #temperatura
    temperatura = 27 - (lectura -
    0.706)/0.001721

    #Imprimimos en consola la temperatura
    print("\nTemperatura ambiente: ",
    temperatura, " °C")

    #Cada 2s se realizará una nueva lectura
    time.sleep(2)
```

Datasheet RP2040

La datasheet es un documento donde viene la información de uso de cualquier componente electrónico, es generada y distribuida por el fabricante.

Ejemplo: **Datasheet RP2040**

Continuación »

PROYECTO 22: PWM

INTRODUCCIÓN

En esta práctica aprenderemos a configurar una señal PWM (Modulación por Ancho de Pulso) con nuestra tarjeta Raspberry Pi Pico.

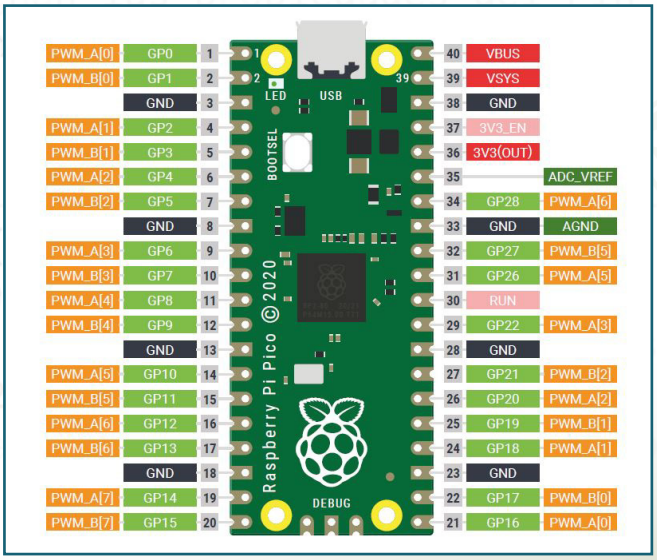
SEÑALES PWM

Las señales de salida PWM (siglas en inglés de Modulación por Ancho de Pulso) son un tipo de señal digital que varía el ancho o tiempo en que la salida se mantiene en alto (3.3V).

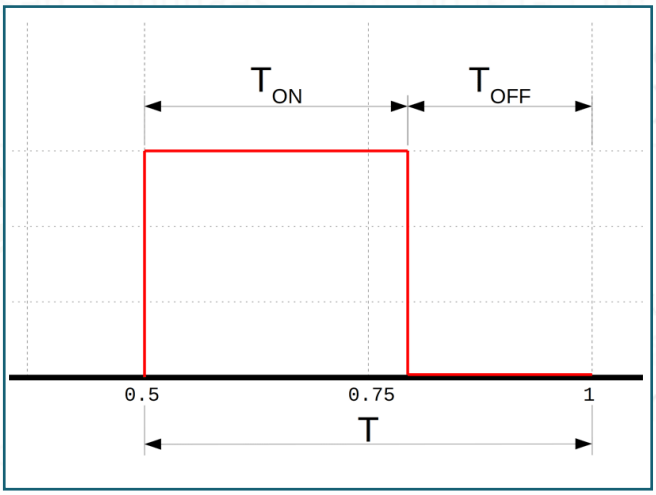
Se utiliza en diversas aplicaciones, pero con Pi Pico es común utilizarlo en:

- Control de velocidad para motores de DC.
- Variar el brillo de un led.
- Control de servomotores.

La Pi Pico cuenta con salidas PWM en cada uno de los Pines GPIO. Están divididas en 8 bloques y cada bloque cuenta con dos salidas independientes. El número representa el bloque PWM conectado a ese pin, la letra representa cual salida del bloque es usada (0 o 1).



Existen un par de conceptos que tendremos que definir para aprender a usar estas señales:Tiempo en alto (TON), Tiempo en bajo (TOFF) , Frecuencia y Ciclo útil



Ton: Tiempo que la señal permanece activa con un voltaje de 3.3V

Toff: Tiempo que la señal permanece a 0 V o apagada.

Frecuencia: Es el periodo de tiempo que la señal tarda en repetirse y se mide en Hertz [Hz].

Hertz

Un Hert significa cuantas veces se repite un ciclo durante un segundo, ejemplo una señal de 60Hz se repite 60 veces en un segundo.

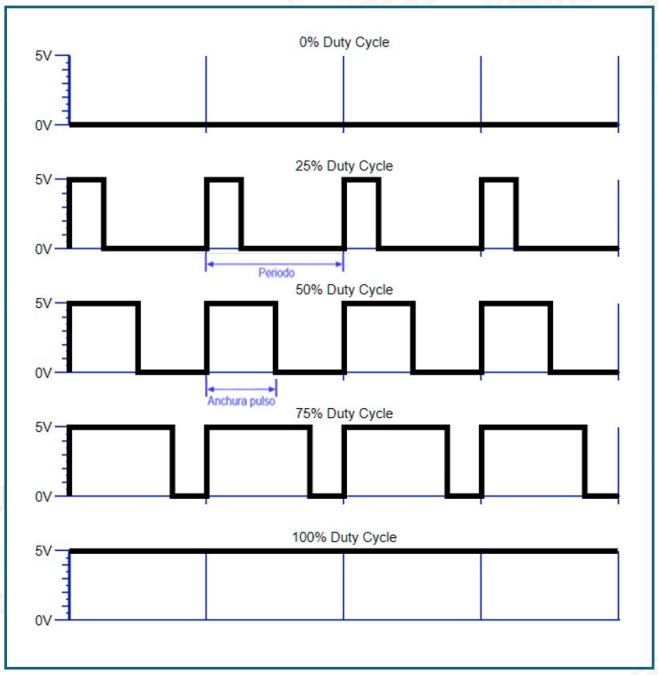
Continuación »

PRÁCTICAS

Ciclo útil: También llamado ciclo de trabajo es un porcentaje que indica la proporción de tiempo que la señal se encuentra en alto, contra el tiempo que está en bajo.

Por ejemplo una señal con un ciclo de trabajo del 50% , la mitad del tiempo la señal estará en alto y la mitad en bajo.

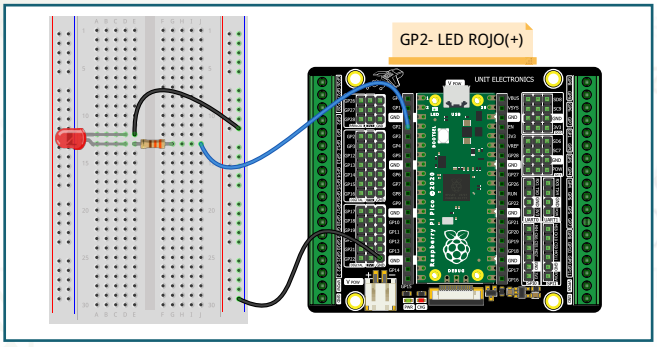
Una señal de 100% de ciclo útil se encuentra todo el tiempo en estado alto. Este parámetro se conoce como Duty Cycle.



MATERIAL

- Raspberry Pi Pico
- Tarjeta de conexiones para Pi Pico
- Resistencia 330 ohms
- Led rojo 5mm
- Cables conexión

DIAGRAMA DE CONEXIÓN



DESARROLLO

Observaremos con este código el funcionamiento de la señal PWM por medio de la intensidad de iluminación del Led.

```
#Añadiremos la nueva librería PWM
from machine import Pin, PWM
import time

#Al objeto 'led' le asignamos las
#propiedades de PWM
#Se declara al pin 2 como salida PWM
led = PWM(Pin(2))

#Frecuencia de trabajo en Hz
led.freq(1000)

while True:
    #Ciclo para aumentar poco a poco el
    #brillo del led
    for i in range (0,65536,10):
        #Asignamos al ciclo útil el valor
        #de i
        led.duty_u16(i)
        #Retardo de 1 milisegundo
        time.sleep(0.001)
    #Retardo de 2 segundos
    time.sleep(2)
```

Ciclo For con Función duty_u16

El For lleva una cuenta de 0 hasta 65,535 ¿Por qué estos números?
Para la instrucción duty se agrega la palabra _u16 para contar con 16 bits (de forma decimal 65,535), por lo tanto:

- Ciclo útil de 0%: duty_u16(0)
- Ciclo útil del 100%: duty_u16(65535)
- Ciclo es del 50%: duty_u16(65535/2)

Observaremos que el brillo del led aumenta paulatinamente, gracias al aumento del duty cycle dentro del ciclo For.

DESAFÍOS

- Crea un programa que paulatinamente disminuye el brillo del led y se repita.
- Crea un programa que aumente el brillo del led y cuando llegué al máximo disminuya lentamente.

PROYECTO 23:
DIMMER LED CON POTENCIÓMETRO

INTRODUCCIÓN

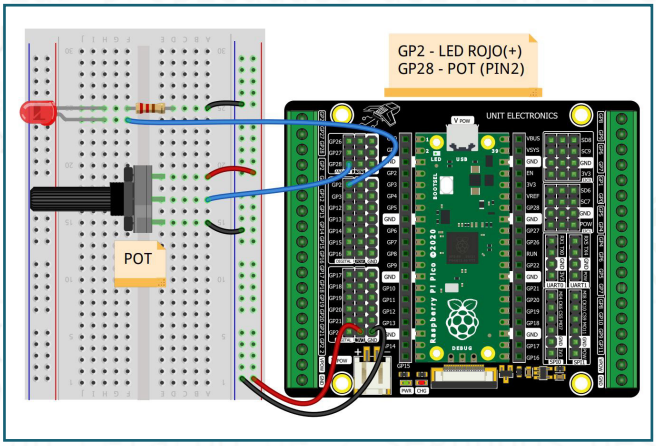
Como se observó en la práctica anterior una de las funciones PWM es el control de brillo de un led, de una manera más comercial puedes encontrar atenuadores de luz llamados Dimmer.

Los dimmer cuentan con una perilla o potenciómetro con el que puedes ajustar la intensidad a la que brilla una lámpara o tira led. En este proyecto imitaremos el funcionamiento con la Pi Pico.

MATERIAL

- Raspberry Pi Pico
- Tarjeta de conexiones para Pi Pico
- Resistencia 330 ohms
- Led rojo 5mm
- Potenciómetro 10K ohm
- Cables dupont

DIAGRAMA DE CONEXIÓN



DESARROLLO

Compila el siguiente código en Thonny, con el circuito armado mueve el potenciómetro a ambos lados, observa como aumenta y disminuye el brillo del led.

```
#Añadiremos la nueva librería PWM, también
#utilizaremos ADC y Pin
from machine import Pin, ADC, PWM
import time

#Asignaremos a potenciómetro la lectura
#ADC del pin 28
potenciometro = ADC(28)

#Led se le asignan las propiedades de PWM
#al Pin 2
led = PWM(Pin(2))

#Frecuencia de trabajo
led.freq(1000)

while True:
    #Ciclo útil del PWM, es el valor del
    #Potenciómetro
    led.duty_u16(potenciometro.read_u16())
```

Señales PWM

Puedes encontrar señales PWM en muchas aplicaciones, por ejemplo:

- Control de luces
- Control de velocidad de motores
- Fuentes de voltaje
- Luces RGB
- Ventiladores de computadora
- Control de servomotores

PROYECTO 24:
COMUNICACIÓN SERIAL I2C

INTRODUCCIÓN

Para esta práctica utilizaremos un protocolo de comunicación que está dentro de la Pi Pico:I2C (Inter – Integrated Circuit por sus siglas en inglés) , que configuraremos para enviar y/o recibir datos.

PROTOCOLOS DE COMUNICACIÓN

Para enviar mensajes o información entre un módulo y nuestra tarjeta Pi Pico es necesario que ambos entiendan el tipo de información que están compartiendo, en otras palabras, que hablen el mismo idioma.

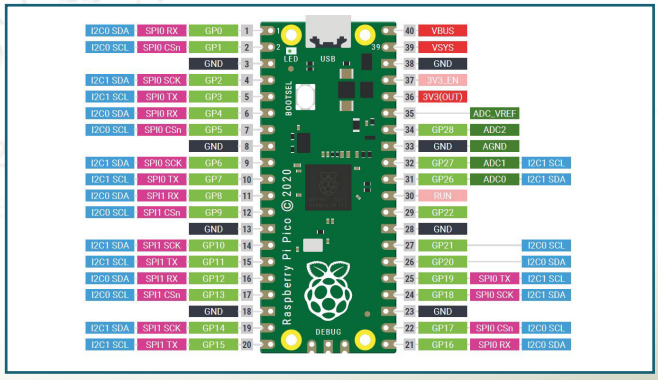
La Raspberry Pi Pico posee tres tipos de comunicación: UART, SPI e I2C. En este manual usaremos la comunicación I2C ya que es el protocolo más popular que existe.

COMUNICACIÓN I2C

Con I2C permite comunicar hasta 128 dispositivos con tan solo dos cables y compartiendo la misma tierra GND.

En la siguiente imagen puedes encontrar en azul los pines que tiene la Pi Pico y que pueden trabajar con I2C, están divididos en dos módulos I2C0 e I2C1. En tu tarjeta de conexiones puedes encontrar dos bloques dedicados a I2C:

- SD8 (GPI8) y SC9 (GPIO9) del módulo I2C0.
- SD6 (GPIO6) y SC7 (GPIO7) correspondientes a I2C1.

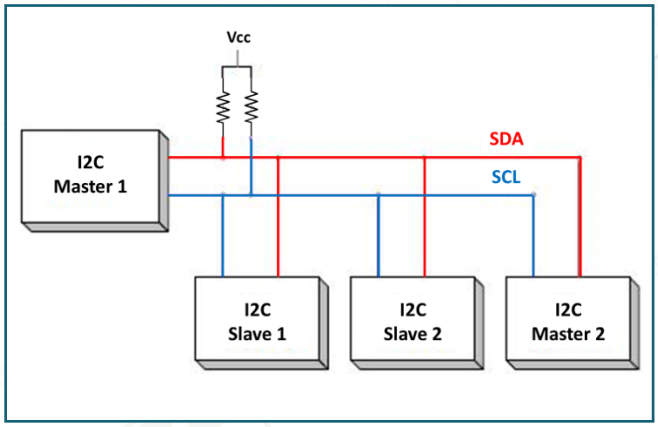


Los dos cables que se utilizan en la comunicación son llamados SDA y SCL.

- SDA es el canal de datos
- SDL es un reloj, internamente ayuda a que los dispositivos trabajen juntos.

El modo de funcionamiento es por medio de "maestros" y "esclavos":

- Maestro: Es el encargado de coordinar las comunicaciones y enviar órdenes a los dispositivos esclavos.
- Esclavo: Están al pendiente de las órdenes del maestro, cuando reciben una orden la realizan y si lo requiere regresan información al maestro.



Dirección I2C

¿Cómo se comunica el **Maestro** con cada uno de los Esclavos si existe una gran cantidad de dispositivos conectados a los mismos cables?
Esto se resuelve con Direcciones. El **Maestro** llama a la dirección del dispositivo con el que se quiere comunicar y sólo comparte información con él, el resto de los dispositivos no interfieren a menos que el **Maestro** llame a su dirección.

Continuación »

Uso de I2C en Microphyton

Para programar los puertos I2C en MicroPython debes de declarar la librería I2C del módulo Machine, con la siguiente declaración:

```
from machine import I2C
```

Después declaramos los pines que utilizaremos como SDA y SCL, por ejemplo, utilizaremos del bloque I2C1 los pines 6 y 7:

```
sda_pin = Pin (6)
scl_pin = Pin (7)
```

Y finalmente se declara el objeto al que le atribuimos las propiedades de I2C.

```
variable = I2C (bloque, sda = sda_pin, scl = scl_pin, freq)
```

Donde:

Variable es el nombre de la variable a la que le asignaremos las propiedades del I2C

Bloque es 0 o 1, se declara como 0 si los pines que utilizaremos pertenecen a I2C0 o es 1 si pertenece a I2C1.

sda_pin y scl_pines son los pines que declaramos previamente. Siempre se igualan a sda y scl respectivamente.

freq es la frecuencia máxima de trabajo para el reloj scl.

Para obtener la dirección I2C de cualquier dispositivo conectado a tu tarjeta debes de utilizar el siguiente comando:

```
variable.scan()[0]
```

Donde:

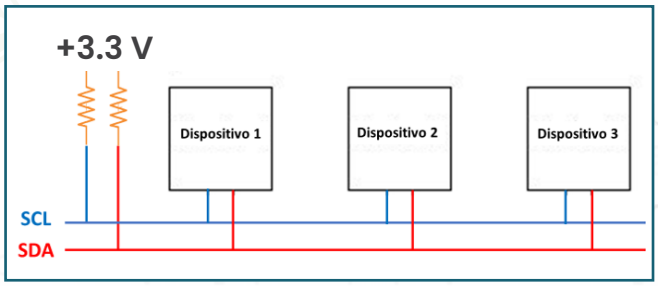
variable es el nombre que se le asigna a las propiedades de I2C

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Pantalla LCD 16x2 con I2C**
- **Cables dupont**

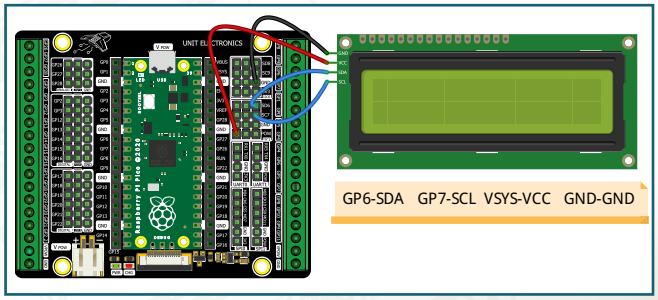
DIAGRAMA DE CONEXIÓN

Para el correcto funcionamiento del sistema I2C debe de conectarse una resistencia de 4.7KΩ a 3.3V en cada una de las líneas de comunicación. Afortunadamente muchos de los dispositivos que trabajan con I2C ya incluyen estas resistencias internamente, en esta práctica utilizaremos el display LCD y si, ya incluye internamente esta conexión.



Debes de conectar la LCD a un pin POW de tu tarjeta de conexiones, ya que la fuente de voltaje de 3.3V no será suficiente para encender la pantalla que trabaja de 4.2 a 5 V.

Ten cuidado al realizar las conexiones de los pines SDA y SCL, si los inviertes, la comunicación no funcionará.



Continuación

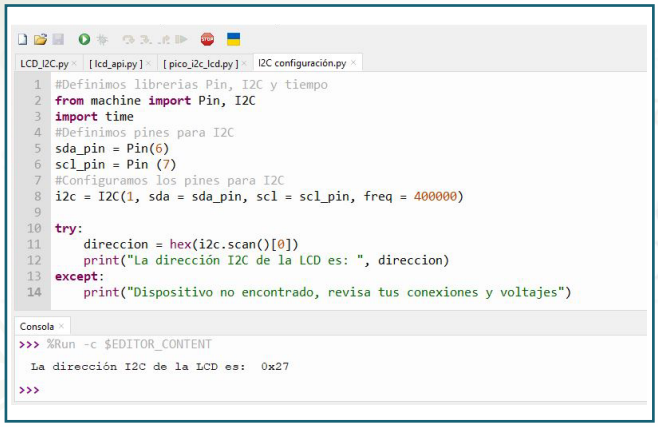
DESARROLLO

El siguiente código tendrá como funcionalidad:

- SD8 (GPI8) y SC9 (GPIO9) del módulo I2C0.
- SD6 (GPIO6) y SC7 (GPIO7) correspondientes a I2C1.

Si no tienes nada conectado a los pines que configuraste o los conectas de forma errónea te aparecerá un mensaje de advertencia. Si esto ocurre revisa tus conexiones, que la LCD está energizada correctamente y que los pines sean los correctos.

```
#Definimos librerías Pin, I2C y tiempo
from machine import Pin, I2C
import time

#Definimos pines para I2C
sda_pin = Pin(6)
scl_pin = Pin (7)

#Configuramos los pines para I2C
i2c = I2C(1, sda = sda_pin, scl = scl_pin,
freq = 400000)

try:
    #Obtenemos la dirección I2C de la LCD
    direccion = hex(i2c.scan()[0])
    print("La dirección I2C de la LCD es:
    ", direccion)
except:
    #Si hay un error se te notificará en
    #la consola
    print("Dispositivo no encontrado,
    revisa tus conexiones y voltajes")
```



Función hex()

Observa la función *i2c.scan()[0]* se encuentra dentro de otra función *hex()*, esto es porque la función *hex* convierte en hexadecimal cualquier número que se coloque dentro de ella.

Para los microcontroladores es más común este sistema y podrás reconocerlo porque el número empieza con un "0x".

Función try

Las declaraciones *Try* y *Except* se utilizan para evitar errores dentro de la ejecución del programa.
Si ejecutamos la función *i2c.scan()* y el programa no reconoce el dispositivo conectado se marcará un error, ya que no tiene nada que escanear.
Con la función try al marcarse un error en esas líneas de código, el programa saltará a la instrucción *except*, que desplegará un mensaje indicando que no hubo comunicación.

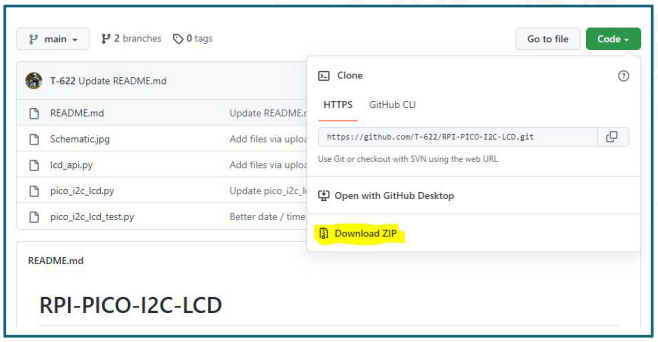
PROYECTO 25: PANTALLA LCD 16X02

INTRODUCCIÓN

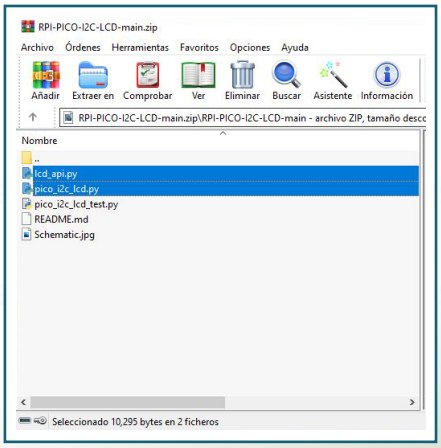
Para esta práctica utilizaremos la pantalla LCD, la cual requiere de una librería externa, que explicaremos cómo implementarla.
En el siguiente enlace podrás descargarla.

https://github.com/T-622/RPI-PICO-I2C-LCD

El procedimiento para instalar la librería es el siguiente:

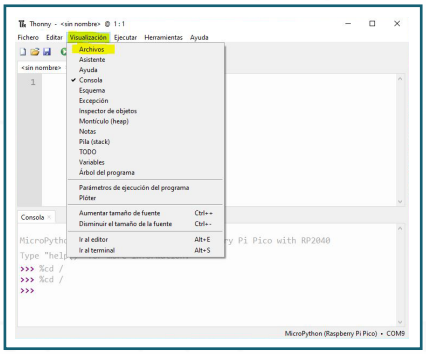


- Da clic en el botón verde llamado “Code”, se desplegará un menú
- Selecciona “Download Zip”
- Se descargara un archivo .zip, es necesario contar con un lector de .zip para que puedas ver el contenido del archivo
- Dentro de la carpeta encontrarás dos archivos llamados lcd_api.py y pico_i2c_lcd.py son los necesarios para utilizar la LCD, descárgalos en tu computadora

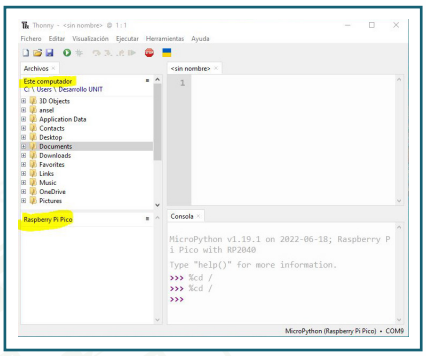


Ahora cargaremos estos dos archivos a la memoria de la Raspberry Pi Pico:

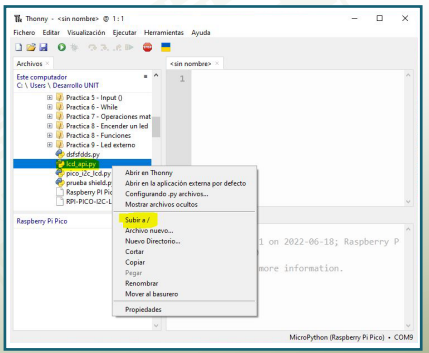
- Ingresa al IDE Thonny y da clic en “Visualización”, en la pestaña que se despliega selecciona: “Archivos”



- Se te desplegarán dos ventanas del lado izquierdo, en ellas se muestran los archivos que tienes cargados en tu computadora y en la parte de abajo estarán los archivos de la Pi Pico

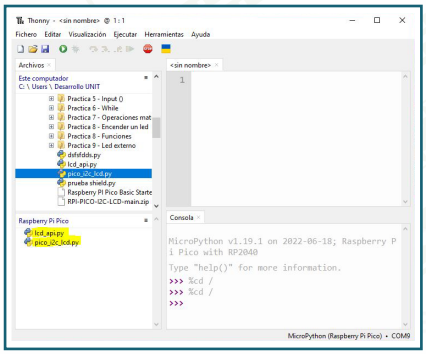


- En la ventana de archivos de tu computadora, busca la carpeta donde descargaste lcd_api.py y pico_i2c_lcd.py, localiza estos documentos y da clic derecho sobre cada uno



Continuación

- En el menú que se desplegará da clic en “Subir a/” y automáticamente se copiará a tu Pi Pico
- Realiza los pasos anteriores con el archivo pico_i2c_lcd.py. Ahora tu ventana de archivos de Raspberry mostrará ambos archivos como se muestra en la siguiente imagen. Esto significa que copiaste las librerías de manera correcta



Listo, ya podemos utilizar el display LCD en nuestra práctica. La librería se encarga de hacer las configuraciones necesarias y traducir los comandos de MicroPython a un lenguaje que el display intérprete. Las funciones que utilizaremos serán:

Envía una cadena de caracteres a la pantalla.

```
- lcd.putstr (“Cadena de caracteres”)
```

Si queremos imprimir una variable utilizamos el siguiente comando:

```
- lcd.putstr (str (Variable))
```

Para borrar todo el contenido de la pantalla ocupamos:

```
- lcd.clear()
```

Podemos imprimir datos en la pantalla empezando desde cualquier punto de ella, solo tenemos que dar la columna y fila donde queremos iniciar como si fuera una coordenada, utilizando el siguiente comando:

```
- lcd.move_to(Col, Row)
```

Como cualquier biblioteca, debemos declararla al inicio del programa con las propiedades que necesitamos:

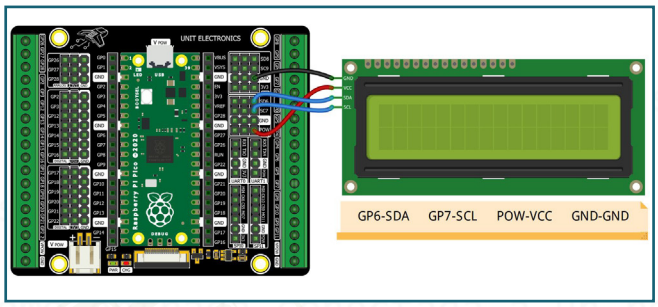
```
- from lcd_api import LcdApi
- from pico_i2c_lcd import I2cLcd
```

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Pantalla LCD 16x2 con I2C**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN

No olvides conectar el pin de alimentación VCC de la LCD a un Pin POW de la tarjeta de conexiones.



DESARROLLO

```
#Definimos librerías Pin, I2C y tiempo
from machine import I2C,Pin
import time

#Librerías necesarias para utilizar la
#LCD:
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

#Definimos pines para I2C
sda_pin = Pin(6)
scl_pin = Pin (7)
```

Continuación

```
#Configuramos los pines para I2C
i2c = I2C(1, sda = sda_pin, scl = scl_pin,
freq=400000)

#Obtenemos la dirección I2C de la LCD
I2C_DIREC = i2c.scan()[0]

#Definimos el tipo de LCD que ocuparemos
#Si ocupas un LCD de otras dimensiones,
#aquí debes de modificarlas
I2C_FILAS = 2
I2C_COLS = 16

#Configuramos la LCD con las bibliotecas
#que descargamos
lcd = I2cLcd (i2c, I2C_DIREC, I2C_FILAS,
I2C_COLS)

lcd.putstr("H")
#Imprimimos una letra
time.sleep(0.5)
#Retardo de 1/2 segundo

lcd.putstr("o")
time.sleep(0.5)

lcd.putstr("l")
time.sleep(0.5)

lcd.putstr("a")
time.sleep(0.5)

#Imprimimos caracteres, observa el espacio
#al inicio de la palabra
lcd.putstr(" mundo")
time.sleep(2)

#Ciclo para imprimir un conteo del 0 al 15
for i in range (0, 16):
    #Coordenada de inicio (columna, fila)
    lcd.move_to(7, 1)
    #Imprimimos el conteo, utilizando la
    #función str()
    lcd.putstr(str(i))
    time.sleep(1)

#Borramos la pantalla
lcd.clear()
```

Observa que si quieres escribir desde el principio de la pantalla no requieres dar las coordenadas de inicio.

Existen más instrucciones que puedes utilizar en la pantalla, esta es la lista que incluye la librería. Puedes enviar cada comando desde la consola a la Pi Pico para que observes cómo trabaja cada una:

Instrucciones para pantalla LCD

- lcd.show_cursor() / lcd.hide_cursor() - Muestra / Oculta el cursor de la pantalla lcd (Barra blanca)
- lcd.blink_cursor_on() / lcd.blink_cursor_off() - Activa / desactiva el cursor parpadeante al imprimir
- lcd.backlight_on() / lcd.backlight_off() - Enciende / apaga la luz de fondo de la pantalla LCD
- lcd.display_on() / lcd.display_off() - Enciende/apaga la pantalla completa
- lcd.custom_char(Num, bytearray([HEX chars])) - Num puede ser cualquier número entero 0 - 8 (Escribiendo en ubicaciones CGRAM) simplemente usado para numeración.

  Los caracteres HEX simplemente se crean usando este enlace:
  https://maxpromer.github.io/LCD-Character-Creator/
  Proporcionará una cadena de caracteres hexadecimales que pueden reemplazar los "caracteres hexadecimales" en el comando de ejemplo.

PROYECTO 26:
JUEGO DE REFLEJOS CON PANTALLA LCD

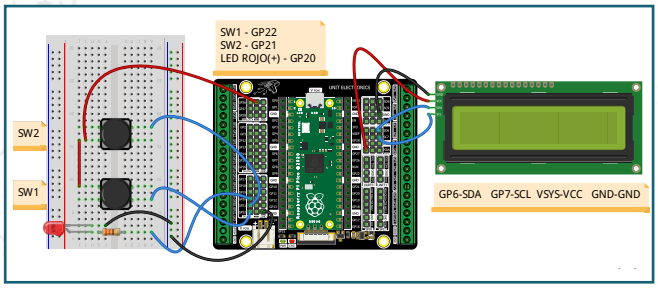
INTRODUCCIÓN

¿Recuerdas el juego de reflejos de la práctica 15? Es momento de ir al siguiente nivel, retomaremos este juego incorporando la pantalla LCD donde se te notificará cuando debas de presionar el botón y se mostrará al jugador ganador.

MATERIAL

- **Raspberry Pi Pico**
- **Resistencia 330 ohms**
- **Push botón 4 pines**
- **Led rojo**
- **Pantalla LCD 16x2 con I2C**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN



DESARROLLO

```
#Definimos librerías
from machine import Pin, I2C
from time import sleep
import random

#Librerías necesarias para utilizar la LCD:
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

#Definimos nuestras entradas y salidas
led = Pin(20, Pin.OUT)
sda_pin = Pin(6)
scl_pin = Pin (7)
boton_derecho = Pin(21, Pin.IN, Pin.PULL_DOWN)
boton_izquierdo = Pin(22, Pin.IN,
Pin.PULL_DOWN)

#Configuramos los pines para I2C
i2c = I2C(1, sda = sda_pin, scl = scl_pin,
freq=400000)

#Obtenemos la dirección I2C de la LCD
I2C_DIREC = i2c.scan()[0]

#Definimos el tipo de LCD que ocuparemos
#Si ocupas un LCD de otras dimensiones, aquí
#debes de modificarlas
I2C_FILAS = 2
I2C_COLS = 16

#Configuramos la LCD con las bibliotecas que
#descargamos
lcd = I2cLcd (i2c, I2C_DIREC, I2C_FILAS,
I2C_COLS)

#Variable de apoyo para saber quien presionó el
#botón
boton_presionado = None

#Función callback para activar la interrupción:
def funcion_interrupcion(pin):

    #Desactivamos interrupción 1
    boton_derecho.irq(handler=None)

    #Desactivamos interrupción 2
    boton_izquierdo.irq(handler=None)

    #Variable global para botón
    global boton_presionado

    #Obtenemos el pin de la primera interrupción
    boton_presionado = pin

    #Ciclo para prender y apagar la pantalla
    for i in range(6):

        # Apaga la luz de fondo de la pantalla
        lcd.backlight_off()
        sleep(0.1)

        # Enciende la luz de fondo de la
        # pantalla
        lcd.backlight_on()
        sleep(0.1)

#Ciclo indeterminado de juegos
while True:

    #Limpiamos pantalla
    lcd.clear()

    #Inicia la pantalla en Col: 2, Fila: 0
    lcd.move_to(2, 0)
    lcd.putstr("Iniciamos!!!")

    #Encendemos el led, señal de que inicia el
    #juego
    led.value(1)

    #Usamos la función random para generar un
    #retardo indeterminado
    sleep(random.uniform(5, 10))
```

Continuación »

```
#Limpiamos pantalla
lcd.clear()

#Apagamos el led, señal de que presiones el
#botón
led.value(0)
lcd.putstr("Presiona ahora!")

#Interrupciones del programa, se definen
#como flancos de subida
boton_izquierdo.irq(trigger =
Pin.IRQ_RISING, handler=
funcion_interrupcion)
boton_derecho.irq(trigger = Pin.IRQ_RISING,
handler= funcion_interrupcion)

#Mientras esperamos a que una de las
#interrupciones se dispare no hacemos nada

while boton_presionado is None:
    sleep (1)

lcd.clear()
lcd.putstr("Ganador: Jugador")

#Si la interrupción es por Botón Derecho
if boton_presionado is boton_derecho:
    lcd.move_to(4, 1)
    lcd.putstr("DERECHO")

#Si la interrupción es por Botón Izquierdo
elif boton_presionado is boton_izquierdo:
    lcd.move_to(3, 1)
    lcd.putstr("IZQUIERDO")

sleep(10)
#Esperamos 10 segundos a reiniciar el juego

#Valor de botón para reiniciar el juego
boton_presionado = None
```

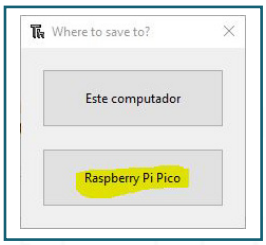
DATOS RELEVANTES

Agregando una pantalla LCD, ya no dependemos directamente de la computadora para conocer el ganador, podríamos agregar una conexión USB a 5V, conectar una power bank o una batería LiPo y llevar tu juego de reflejos a todas partes.

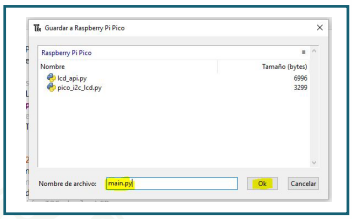
Solo tenemos un inconveniente, al desconectar la Pi Pico de la computadora se borra el programa compilado de la memoria de la tarjeta, esto se resuelve de manera sencilla:

- Dirígete a la pestaña 'Fichero', da clic sobre ella.

- Selecciona la opción 'Guardar como...'
- En la ventana que se abrirá selecciona Raspberry Pi Pico.



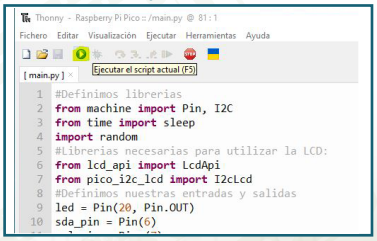
- Ahora tendrás un recuadro donde podrás encontrar los programas que se habían almacenado anteriormente:



- Da clic en la barra de 'Nombre de archivo:' e ingresa el nombre de:

  main.py

- Da clic en ok y ahora ejecuta nuevamente el script, el programa se compilara normalmente.



Si todo salió correctamente desconecta la Raspberry de tu computadora y conectala a un eliminador de 5V o una power bank y el programa funcionará de forma automática, sin la necesidad de hacerlo desde Thonny.

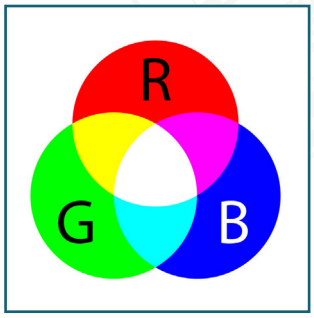
Para agregar alguna fuente externa es importante conocer el consumo de nuestro proyecto para que todos los dispositivos tengan la alimentación de tensión adecuada a sus especificaciones.

PROYECTO 27: NEOPIXEL

INTRODUCCIÓN

Neopixel es un encapsulado que posee 3 leds (Rojo, Verde y Azul) junto a un circuito de control, el cual nos ayuda a trabajar con los 3 colores utilizando solamente un pin de datos.

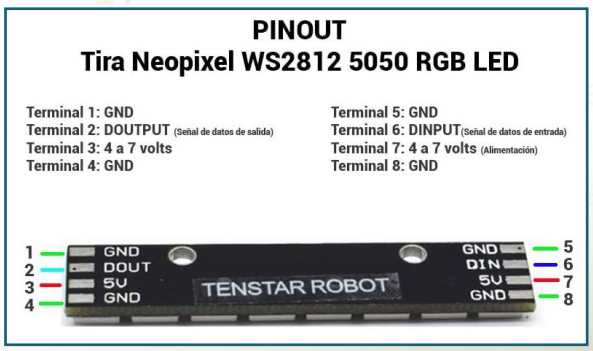
Son muy populares ya que con cada led puedes generar la cantidad de 16,777,216 de colores. Esto es debido a que cada color varía su brillo de manera independiente, lo cual te permite que se generen distintas tonalidades.



También puedes modificar el color de cada Neopixel de forma independiente, por ejemplo: configurar cada led de un solo color o que solo un par estén encendidos y el resto permanezcan apagados.

En el kit se incluye una pequeña tira neopixel de 8 leds, todos los LEDs se controlan desde una sola línea de datos. Y se puede conectar más de una tira en serie conectando los pines de entrada DIN y salida DOUT.

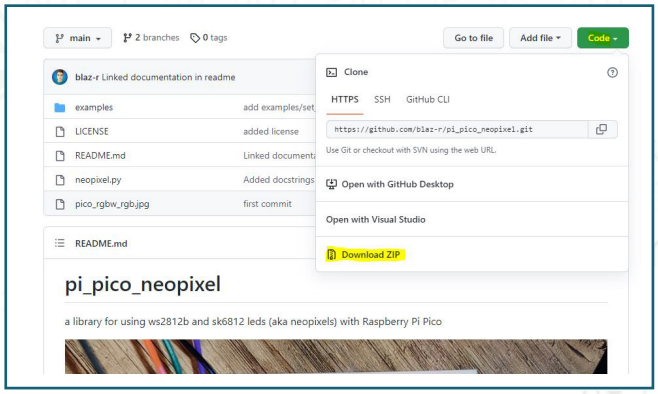
En la siguiente imagen encontrarás los pines de conexión del módulo:



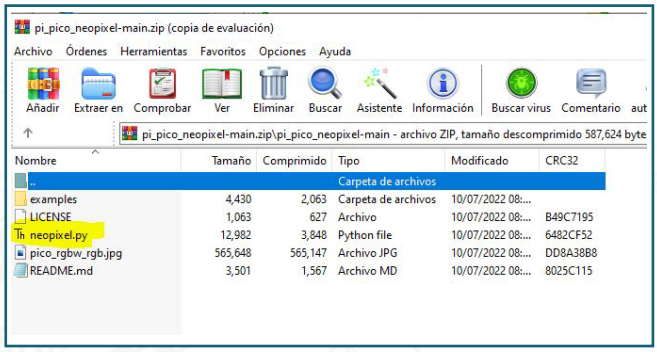
Al igual que con las Pantallas LCD utilizaremos una librería, te recordaremos rápidamente cómo instalar la librería en tu Raspberry.

Ingresa al siguiente enlace y descarga el archivo .zip:

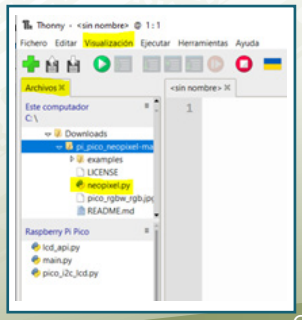
https://github.com/blaz-r/pi_pico_neopixel



Descomprime el archivo "neopixel.py" de la carpeta descargada.



Sube el archivo a tu Pi Pico desde Thonny: Visualización -> Archivos -> Este computador -> neopixel.py



Continuación »

- Da clic derecho sobre 'neopixel.py' y selecciona "subir a/".

Este proceso se tiene que realizar en cada Raspberry que utilices, tendrás que cargar la biblioteca nuevamente si cambias de tarjeta.

Ahora abre un nuevo documento e inicia el archivo con las siguientes líneas:

```
from neopixel import Neopixel
```

Las funciones que utilizaremos en este programa serán:

```
Neopixel(NUM_LEDS , 0, LED_PIN, "GRB")
```

Donde:

- NUM_LEDS es el número de Neopixeles que posee la tira.
- 0 es el ID de máquina de estado, es una configuración interna del microcontrolador, te recomiendo dejarlo en cero.
- LED_PIN es el pin de datos de la tira
- "GRB" es el tipo de led que utilizaremos, hace referencia a que es de tres colores (Green – Red – Blue)

```
variable.brightness(brillo)
```

Ajusta el brillo del led, puedes enviarle valores desde 1 hasta 255. Colocar un 1 pondría al mínimo el brillo y 255 sería al máximo.

```
variable.set_pixel(i, color)
```

Coloca un color deseado en un led particular.

- i es el número de led que quieres configurar
- color es el valor que quieres configurar en dicho led (1 a 255)

```
variable.show()
```

Envía el dato a la tira Neopixel.

Lo último que necesitamos para configurar nuestra tira de led es definir los colores en los que encenderá cada uno. Esto se realiza ajustando el tono de cada uno de los 3 colores RGB, el tono lo puedes ajustar utilizando PWM, no te preocupes no es necesario configurarlo de eso se encarga la librería.

Pero si debes de recordar que el PWM puede tomar valores entre 0 y 255, la biblioteca reconoce el color dependiendo de donde se encuentre posicionado (Rojo, Verde, Azul). Así por ejemplo los tres colores primarios son:

```
Rojo = (255, 0, 0)
Verde = (0, 255, 0)
Azul = (0, 0, 255)
```

Y colores intermedios pueden ser:

```
Amarillo = (255, 150, 0)
Violeta = (138, 43, 226)
Indigo = (75, 0, 130)
```

El blanco y negro se pueden conseguir encendiendo al 100% los 3 colores o apagándolos por completo:

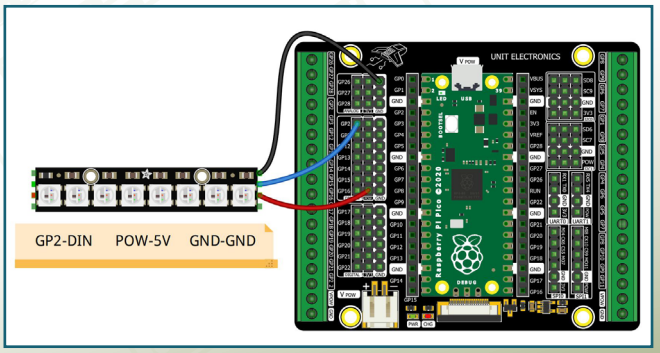
```
Blanco = (255, 255, 255)
Negro = (0, 0, 0)
```

MATERIAL

- **Raspberry Pi Pico**
- **Tarjeta de conexiones para Pi Pico**
- **Tira neopixel 8 leds**
- **Cables dupont**

DIAGRAMA DE CONEXIÓN

Conecta el voltaje de alimentación a los pines POW y asegúrate de conectar GP2 al pin DIN del módulo.



Continuación

DESARROLLO

```
import time
#Librería para utilizar la Neopixel
from neopixel import Neopixel

#Variables de configuración
NUM_LEDS  = 24 #¿Cuantos leds tiene tu tira?
LED_PIN = 2          #Pin de conexión

#Declaramos el objeto con propiedades de
#Neopixel
pixels = Neopixel(NUM_LEDS , 0, LED_PIN, "GRB")
velocidad = 0.1 #Cambio de pixel, en segundos
brillo = 10 #Brillo del led, 1 a 255

#Definimos colores
red = (255, 0, 0)
orange = (255, 165, 0)
yellow = (255, 150, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
indigo = (75, 0, 130)
violet = (138, 43, 226)


#Cadena donde se almacenan los colores
colors_rgb = (red, orange, green, yellow, blue,
indigo, violet)

#Brillo del led
pixels.brightness(brillo)

#Ciclo de trabajo
while True:
    #Este ciclo cambia de color
    for color in colors_rgb:
        #Este ciclo cambia de pixel
        for i in range(NUM_LEDS ):
            #Definimos el led y el color que
            #deseamos
            pixels.set_pixel(i, color)
            #Retardo
            time.sleep(velocidad)
            #Mandamos a la tira led el comando
            pixels.show()
```

Librería Neopixel

Si quieres conocer el resto de funciones que posee esta librería puedes visitar el siguiente enlace, el creador de la biblioteca explica las distintas funciones que puedes realizar con la Neopixel y como se configuran, además de añadir más ejemplos útiles:

https://github.com/blaz-r/pi_pico_neopixel/wiki/Library-methods-documentation

Puedes consultar el resto de colores que puedes crear en la neopixel dentro del siguiente enlace:

https://github.com/blaz-r/pi_pico_neopixel

DESAFÍOS

Cambia las variables de velocidad y brillo para que veas la respuesta del módulo Neopixel.

Cambiar el número de leds totales por un número menor que 8

¿Recuerdas la práctica 20? Puedes añadir a esta práctica dos potenciómetros para variar la intensidad y velocidad con la que enciende el led.

UNIT
ELECTRONICS

KIT elaborado por el equipo de UNIT

Siguenos en nuestras redes sociales

@unitelectronics