# Práctica 2:
# Instalación de Micropython en el microcontrolador RP2040

Fundamentos de Sistemas Embebidos

Autor: José Mauricio Matamoros de Maria y Campos

## 1. Objetivo

El alumno aprenderá a instalar Micropython, en una Raspberry Pi Pico (RP2040).

## 2. Introducción

La presente práctica resume los pasos a seguir para instalar microPython en el chip microcontrolador RP2040 En particular se controlará el encendido del LED centinela de la tarjeta Raspberry Pi Pico, así como en la lectura de la temperatura del controlador. Para este propósito se utilizará la interfaz de desarrollo integrada Thonny.

### 2.1. El microcontrolador RP2040

Las tarjetas Raspberry Pi han sido diseñadas con un objetivo en particular: la educación. En todo el mundo se utilizan estas tarjetas enseñar a los estudiantes a desarrollar proyectos en las áreas de ciencias de la computación, ingeniería, automatización con hardware, e Internet de las cosas usando un gran número de lenguajes de programación [1, 2].

En contraste con sus hermanas mayores que son computadoras completas que integran un puerto de propósito general, la Raspberry Pi Pico no es una computadora, sino una tarjeta microcontroladora muy poderosa y económica Su popularidad se debe en gran parte a que puede llegar a costar tan sólo $4.^{00}$USD, mientras que su chip, el RP2040, se puede conseguir en el mercado por $1.^{00}$USD [1, 2]. La Pico es simplemente una tarjeta integrada del tamaño de una barra de goma de mascar que integra un adaptador de corriente para programarse vía USB (el RP2040 opera a 3.3V mientras que la especificación USB opera a 5VDC) con acceso a los 40 pines del RP2040 [1, 2].

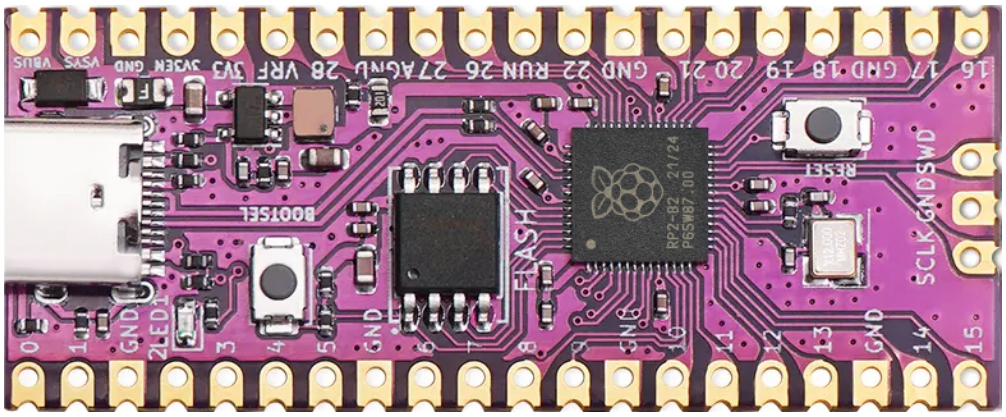



Figura 1: Raspberry Pi Pico

Lo anterior es relevante porque la Pico se ha estado posicionando durante los últimos años como uno de los microcontroladores de medio perfil más utilizados en el mercado, superando al Arduino en muchos aspectos [1, 2]. Por ejemplo, el RP2040 integra los siguientes componentes de hardware, entre otros:

- DOS nucleos (Dual Core ARM Cortex-M0+) a 133MHz
- 264kB de memoria SRAM integrada
- Controlador DMA (Direct Memory Access)
- 30 pines de propósito general (4 con sporte analógico)
- 4 canales ADC de 12 bits
- Sensor de temperatura integrado conectado al ADC
- 16 canales PWM
- Depuración de código en chip vía SWD
- 2 puertos UARTs
- 2 controladores SPI
- 2 controladores $I^2C$
- 8 máquinas de estado PIO
- Controlador USB 1.1 con soporte para modos dispositivo y maestro
- 2 PLLs integrados para generar los relojes interno y USB
- Bus QSPI dedicado para interfaz con memoria Flash de hasta 16MB
- Soporte de periféricos extendido vía PIO (Programmable IO)
- Regulador de voltaje integrado

## 2.2. MicroPython

Creado por Damien P. George y Paul Sokolovsky (entre otros), MicroPython fue diseñado para ser una versión ligera y eficiente del lenguaje Python 3 instalable en un pequeño microcontrolador. Dado que Python es un lenguaje interpretado y, por lo tanto, en general más lento que los lenguajes compilados (véase Subsección 2.3), MicroPython fue diseñado para ser lo más eficiente posible para que pueda ejecutarse en microcontroladores que normalmente son más lentos y tienen mucha menos memoria que una computadora personal típica [1, 3].

Otro aspecto es que las tarjetas microcontroladores como Arduino requieren que el código sea compilado en otra computadora para producir un código objeto ejecutable que debe ser cargado en dicha tarjeta. En contraste, dado que MicroPython es un intérprete ejecutándose directamente en el hardware, puede prescindirse del paso intermedio de compilación: ¡se puede ejecutar el código fuente directamente en el hardware! Esto permite a los fabricantes de hardware construir tarjetas pequeñas y económicas con MicroPython embebido en el chip y que puedan ejecutar virtualmente cualquier programa tanto en la fase de desarrollo como en la de producción [1, 3].

MicroPython permite crear programas simples, eficientemente especificados y fáciles de entender. Además, incluye las siguientes características [1, 3]:

- **Intérprete interactivo:** MicroPython habilita en la tarjeta física una consola interactiva especial a la que puede acceder conectándose via USB.
- **Bibliotecas estándar de Python:** MicroPython es compatible con muchas de las bibliotecas estándar de Python. En general, uno puede confiar en que alrededor del del 80 % de las bibliotecas más utilizadas estarán disponibles.
- **Bibliotecas a nivel de hardware:** MicroPython tiene bibliotecas integradas que le permiten acceder al hardware directamente para encender o apagar pines, leer datos analógicos, leer datos digitales, controlar hardware (PWM) y más.
- **Extensible:** MicroPython también es extensible. Ésta es una gran característica para usuarios avanzados que necesitan implementar alguna biblioteca compleja de bajo nivel (en C/C++) e incluir la nueva biblioteca en MicroPython.

La mayor limitación de MicroPython deriva de su facilidad de uso: aunque MicroPython está altamente optimizado el código se interpreta sobre la marcha lo que agrega una penalización de desempeño y memoria por parte del intérprete. Esto significa que los proyectos que requieren un alto grado de precisión, un muestreo de datos a alta frecuencia, o comunicaciones a alta velocidad (por ejemplo USB), pueden no ejecutarse lo suficientemente rápido. Estos problemas suelen superarse con el uso de bibliotecas en C/C++ optimizadas para manejar la comunicación de bajo nivel. Por otro lado, MicroPython también requiere de un poco más de memoria que el código nativo. Normalmente, esto no es un problema, aunque debe tomarse en cuenta si el programa va a extenderse con características nuevas. Los programas más grandes que usan muchas bibliotecas podrían consumir más memoria de la disponible. Finalmente, como se mencionó anteriormente, MicroPython no implementa todas las funciones de todas las bibliotecas de Python 3 [1, 3].

## 2.3. Lenguajes compilados $^{\text{v}}/_{\text{s}}$ lenguajes interpretados

Los lenguajes compilados requieren de un programa externo llamado compilador para convertir el código fuente de una forma legible por humanos a una forma ejecutable binaria que pueda ser entendido y ejecutado con máxima eficiencia por el procesador. Por otro lado, los lenguajes interpretados no se compilan, sino que se evalúan línea a línea sobre la marcha con un programa llamado intérprete. Python 3 proporciona un ejecutable de Python que es a la vez un intérprete y una consola que le permite ejecutar el código a medida que se escribe.

Por lo tanto, los lenguajes compilados son mucho más rápidos que los lenguajes interpretados porque el código está preparado (y optimizado) para su ejecución en el microprocesador sin requerir de pasos intermedio en tiempo real para procesar el código antes de la ejecución.

## 2.4. Convertidor Analógico—Digital y sensado de temperatura

El RP2040 cuenta con un convertidor analógico-digital de 12 bits con cinco canales, cuatro de los cuales están anclados a los pines A0–A3 (GPIO 27–29) y un canal especial cuyo único propósito es medir la temperatura del integrado.
De acuerdo con la hoja de especificaciones del RP20240:

> El sensor de temperatura mide el voltaje $V_{BE}$ de un diodo bipolar conectado al quinto canal del convertidor AD (AINSEL=4). Normalmente $V_{BE} = 0.706V$ cuando $T = 27^oC$ con una pendiente de $-1.721\frac{mV}{^oC}$ [4].

Con esta información, y suponiendo linearidad en la respuesta, se puede utilizar la ecuación de la recta $y = mx+b$ para extrapolar el valor de temperatura del integrado de la siguiente manera:

$$\begin{aligned} y &= mx + b \\ &= m\left(x - x_0\right) + y_0 \end{aligned}$$

o, sustituyento la variable $x$ por la temperatura $T$ y $y$ por el voltaje $v$ y los valores $m = -1.721\frac{mV}{^oC}$, $T_0 = 27^oC$ y $v_0 = 0.706V$:

$$\begin{aligned} v &= m\left(T - T_0\right) + v_0 \\ &= -1.721\frac{\text{mV}}{^\circ\text{C}}\left(T - 27^\circ\text{C}\right) + 0.706\text{V} \\ &= \left(-1.721\frac{\text{mV}}{^\circ\text{C}} \times T\right) + \left(27\cancel{^\circ\text{C}} \times 1.721\frac{\text{mV}}{\cancel{^\circ\text{C}}}\right) + 0.706\text{V} \\ &= -1.721T\frac{\text{mV}}{^\circ\text{C}} + 46.467\text{mV} + 706\text{mV} \\ &= -1.721T\frac{\text{mV}}{^\circ\text{C}} + 752.47\text{mV} \end{aligned}$$

despejando a $T$:

$$\frac{v}{-1.721\frac{\text{mV}}{^\circ\text{C}}} = \frac{(T)\cancel{\left(-1.721\frac{\text{mV}}{^\circ\text{C}}\right)}}{\cancel{-1.721\frac{\text{mV}}{^\circ\text{C}}}} + \frac{752.47\cancel{\text{mV}}}{-1.721\frac{\cancel{\text{mV}}}{^\circ\text{C}}}$$

finalmente:

$$T = \frac{-1}{1.721\frac{\text{mV}}{^\circ\text{C}}} \cdot v + 437.23^\circ\text{C} \tag{1}$$

Así, bastará con convertir el valor reportado por el convertidor analógico-digital a su equivalente en milivoltios para obtener la temperatura del circuito integrado.

## 2.5. Thonny

Thonny es un IDE[1] de código abierto para Python desarrollado por la Universidad de Tartu, Estonia, pensado para ser amigable con el principiante. Además de las capacidades estándar de construcción y ejecución de programas, cuenta con un soporte extendido para animación de programas, permitiendo ilustrar los conceptos de variables, control de flujo, evaluacíón de expresiones, llamada a funciones, recursión, referencias y montículos, objetos (incluyendo clases y funciones como valores), datos compuestos (listas, diccionarios y conjuntos) y operaciones de entrada/salida sobre archivos [5, 6].

Thonny puede llevar registro de las acciones del usuario con suficiente detalle para reproducir el proceso de elaboración de un programa, permitiéndole a los estudiantes elegir el nivel de detalle con el que desean analizar la reproducción [5, 6].

Una de las ventajas de Thonny sobre cualquier otro editor de Python es que, a partir de la versión 4.0, incluye soporte para interactuar con varios microcontroladores como el RP2040, el ESP32 y el ESP8266; todos de amplia difusión en el mercado. Además, Thonny permite realizar operaciones en el sistema de archivos virtual del RP2040 (MicroPython convierte el RP2040 en una suerte de memoria USB de unos cuantos kB). Para este fin, basta con contectar la tarjeta controladora a la PC, iniciar Thonny y seleccionar las opciones *View* y *Files*. De igual manera Thony permite copiar archivos de y a la tarjeta controladora para ser utilizados por MicroPython [1, 3].

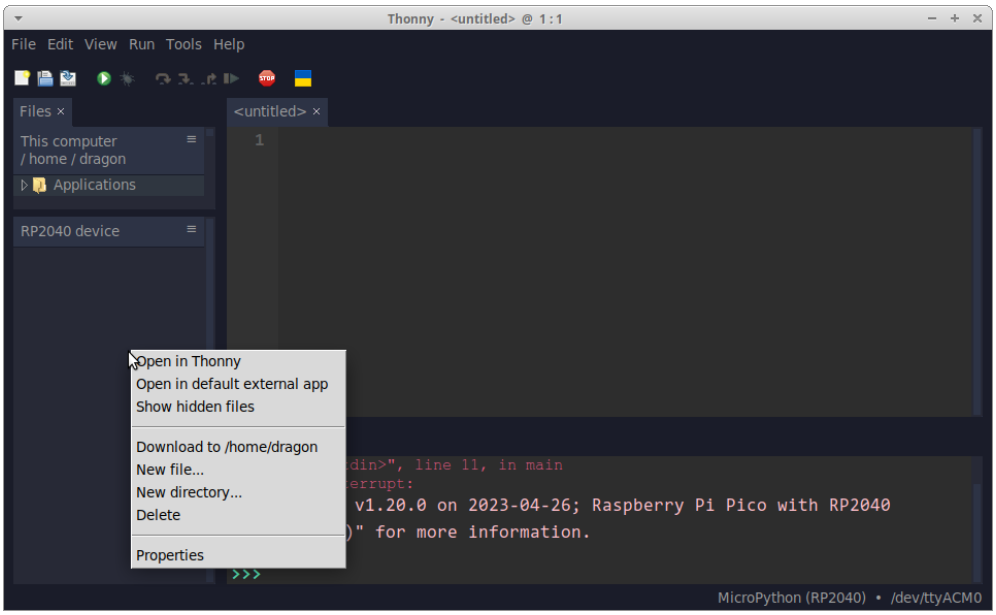



Figura 2: Thonny permite visualizar, crear, copiar, cortar, pegar y, en general, manupular archivos dentro del RP2040.

# 3. Material

Se asume que el alumno cuenta con un una computadora con sistema operativo basado en Linux (se recomienda Xubuntu) e interprete de Python instalado.

- 1 Tarjeta microcontroladora Raspberry Pi Pico o UNIT DualMCU RP2040 + ESP32 (recomendado)
- 1 cable USB-C con soporte para datos[2]
- Cables y conectores varios

[1] Entorno de desarrollo integrado, por sus siglas en inglés *Integrated Development Environment*

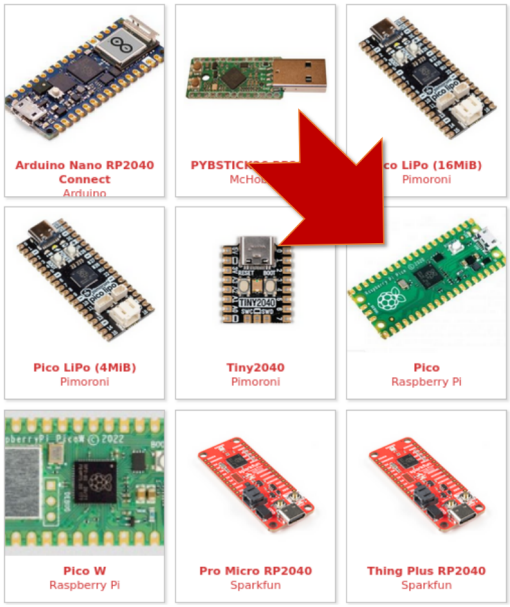
[2] El mercado ofrece un gran número de cables económicos tipo USB-C, especialmente para carga rápida de celulares, tablets y laptops. Estos cables, sin embargo, han sido diseñados para la transmisión de potencia y no de datos, por lo que pueden no contar con las líneas de transmisión de datos para programar un microcontrolador. Verifique que su cable USB-C cumple con la especificación de transporte de datos y no sólo con la de transporte de potencia.

# 4. Instrucciones

1. Descargue el último firmware disponible de MicroPython para el RP2040 siguiendo los pasos de la Subsección 4.1.
2. Cargue el firmware en el microcontrolador RP2040 tomando la Subsección 4.2 como guía.
3. Siga los pasos de la Subsección 4.3 para instalar Thonny.
4. Ejecute los programas de prueba como se indica en las Subsecciones 4.4 y 4.5
5. Analice los programas de las subsecciones 4.4 y 4.5 y realice los experimentos propuestos en la sección 5.

## 4.1. Paso 1: Descargar MicroPython

Ingrese a `https://micropython.org/download/?port=rp2` y seleccione el modelo de su tarjeta controladora. Si no está seguro, simplemente de click en la imagen correspondiente a la Raspberry Pi Pico (`https://micropython.org/download/rp2-pico/`). A continuación descargue último firmware disponible de MicroPython, por ejemplo la versión 1.20.0. Deberá ser un archivo extensión `uf2`.




(a) Selección de la Raspberry Pi Pico


(b) Firmwares disponibles

Figura 3: Descarga de la última versión de MicroPython

**Importante**

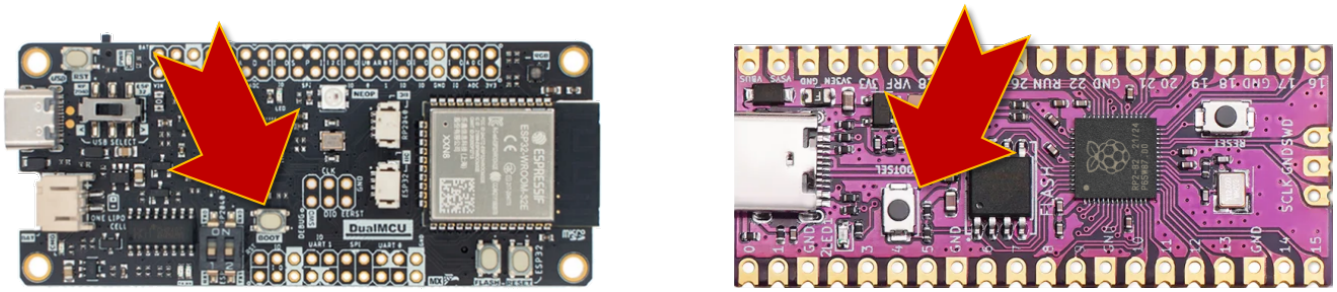
Si tiene una Raspberry Pi Pico-W descargue el firmware para la Pico-W. De otro modo no podrá usar la tarjeta inalámbrica

## 4.2. Paso 2: Instalar MicroPython en el RP2040

Antes de proceder, verifique que su cable adaptador USB-C soporta transmisión de datos.

El RP2040 viene precargado con un microfirmware especial que permite dos modos de arranque: i) el arranque normal en modo microcontrolador que comenzará la ejecución de las microinstrucciones grabadas en la memoria FLASH de programa y ii) el arranque en modo carga, que inicializará al microcontrolador como si fuera un dispositivo de almacenamiento masivo (ej. una memoria USB) donde se puede copiar el firmware compilado para el microcrontrolador. Lejos atrás quedaron los tiempos en que se requería de un dispositivo programador especial o de software complejo para interactuar con el microcontrolador.

Para arrancar el microcontrolador RP2040 en modo carga, basta con presionar el botón BOOTSEL en la tarjeta antes de energizar el microcontrolador conectándolo a una computadora, tal como muestra la Figura 4. Se requiere de un cable adaptador USB tipo C para este propósito.



(a) Boton BOOT en la UNIT DualMCU

(b) Boton BOOTSEL en la Raspberry Pi Pico

Figura 4: Descarga de la última versión de MicroPython

**Importante**

Si cuenta con una tarjeta una tarjeta UNIT DualMCU, verifique que el interruptor USB-Selector esté activado en modo RP2040. Esta tarjeta cuenta, además, con un botón RST que sirve para reiniciar el RP2040 evitando el desgaste innecesario del conector USB.

Para grabar MicroPython en el RP2040 siga los siguientes pasos:

1. Presione el botón BOOT/BOOTSEL de la tarjeta.
2. Sin soltar el botón, conecte la tarjeta a la computadora usando el cable USB-C.
3. Espere dos segundos antes de soltar el botón BOOT/BOOTSEL.
4. Espere unos segundos a que la computadora reconozca el dispositivo.[3] El volumen tendrá un nombre parecido a RPI-RP2.
5. Copie el archivo de firmware, por ejemplo el `rp2-pico-20230426-v1.20.0.uf2` al dispositivo de almacenamiento masivo (la tarjeta).

Tan pronto como la tarjeta detecte que se ha cargado un nuevo firmware, ésta se reiniciará automáticamente.

## 4.3. Paso 3: Instalación de Thonny bajo Ubuntu Linux

El motivo por el cual se aconseja el desarrollo de software y firmware para microcontroladores en sistemas basados en Unix es porque dichos sistemas ofrecen un vasto conjunto de herramientas de compilación cruzada, facilitando el desarrollo para virtualmente cualquier integrado con absoluta independencia de software para terceros, y todo de forma gratuita. Además, el software GNU cuenta con licencia de código abierto, por lo que siempre es posible estudiar y comprender en completitud cómo operan las herramientas de desarrollo para obtener el máximo desempeño posible. Así, un sistema moderno podrá compilar sin ningún problema un sistema desarrollado en 1985, cosa que no es posible en sistemas con Windows.

En un sistema con Ubuntu Linux, instalar Thonny es tan sencillo como abrir una terminal y ejecutar la línea:

```
$ sudo apt install thonny
```

[3] Si la computadora no ha reconocido la tarjeta como un dispositivo de almacenamiento masivo luego de aproximadamente medio minuto, verifique que su cable tenga soporte para transporte de datos.

**Importante**

Los sistemas POSIX cuentan con dos niveles de acceso: usuario y súper-usuario (root) y, por seguridad, sólo un súper-usuario puede instalar programas.
En una terminal, el prompt de los usuarios es un signo de pesos `$`, mientras que el de los súper-usuarios es el símbolo hash `#`. Dado que la mayoría de los sistemas POSIX no cuentan con comando `sudo` se estila especificar en nivel de acceso requerido antes del comando; así la línea

```
$ sudo apt install thonny
```

es equivalente a

```
# apt install thonny
```

¡Pero en ningún caso los caracteres `$` ni `#` se escriben!

Es necesario señalar que *aptitude*, el gestor de paquetes de los sistemas basados en Debian (como ubuntu), contiene versiones estables, robustas y bien probadas en lugar de la última versión con las últimas características. Es por esto que la versión de Thonny instalada vía `apt` podría no tener soporte para el RP2040. Por ello, el conjunto de pasos recomendado a seguir es el siguiente:

```
# apt install python3 python3-pip
$ pip install -U pip thonny
```

A diferencia de *aptitude*, *pip* instalará la versión estable más reciente.

.

Finalmente ejecute Thonny, invocándolo con la línea

```
$ thonny
```

deberá ver una ventana como la siguiente:

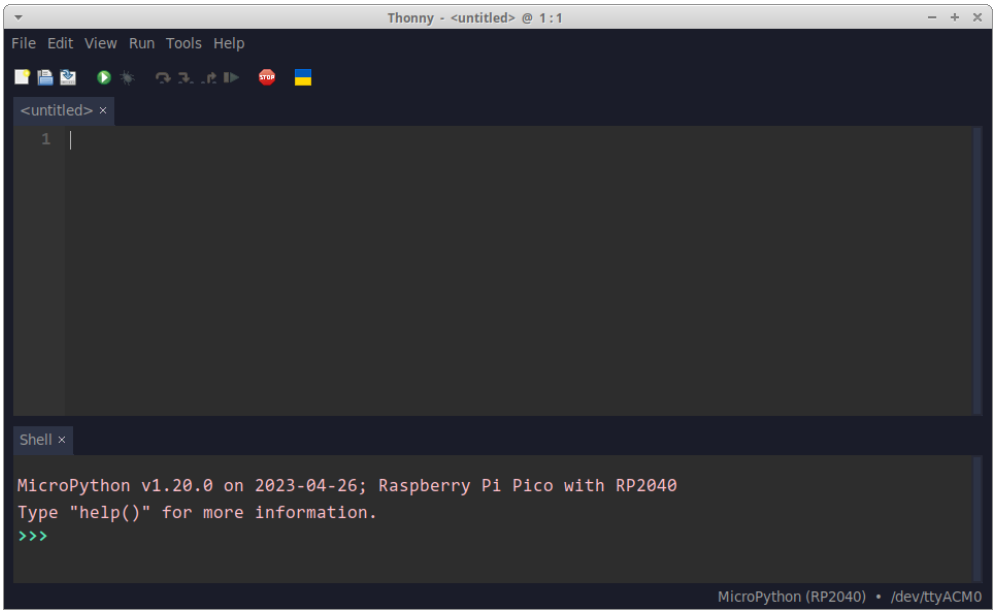



Figura 5: Thonny

Es normal que Thonny marque un error la primera vez que se ejecuta. Esto se debe a que no se ha configurado el intérprete. Para verificar que Thonny pueda operar con el RP2040, haga click en el menú *Run* y seleccione *Configure interpreter...* (ejecutar, configurar intérprete). Verá una ventana similar a la siguiente:

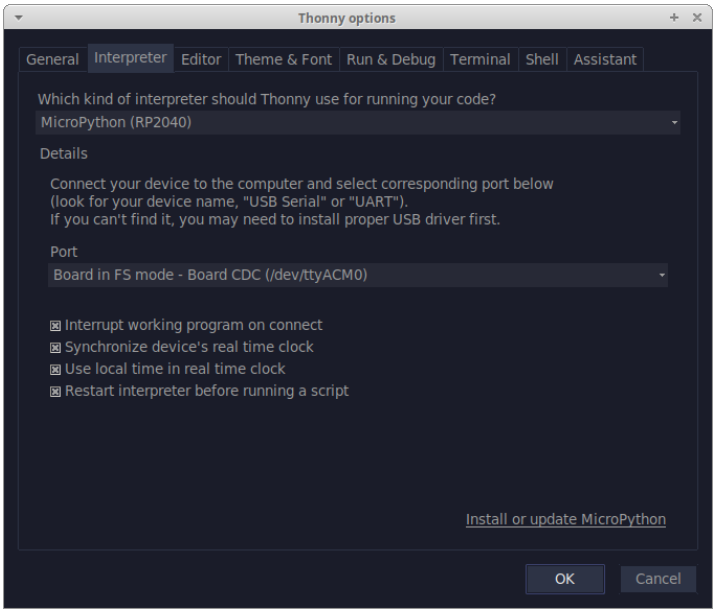



Figura 6: Selección de intérprete y tarjeta en Thonny

En el combo superior seleccione la tarjeta Raspberry Pi Pico (el RP2040). En el recuadro inferior seleccione el puerto en el que se encuentra conectada la tarjeta. Si el cuadro inferior no muestra ningún puerto, verifique que la tarjeta microcontroladora se encuentra conectada a la computadora vía el cable USB.

## 4.4. Paso 4: Led parpadeante con el RP2040

Con la tarjeta controladora configurada y conectada, basta con copiar los programas al editor de Thonny y presionar el botón de ejecución (invocable vía la tecla F5).

Ingrese el siguiente código en el editor y ejecútelo

Código ejemplo 1: `blink.py`: — Led parpadeante usando retardos

```
1 from machine import Pin      # Board IO Pin
2 from utime import sleep_ms   # Delay in milliseconds
3
4 led = Pin(25, Pin.OUT)       # Setup pin 25 (sentinel LED) as output
5
6 while(True):                 # Repeat forever
7     led.toggle()             # Toggle the led
8     sleep_ms(500)            # Wait for 500ms
```

El código de ejemplo 1 configura primero el pin 25 del RP2040 como salida y luego define un bucle infinito con la cláusula **`while`** dentro del cual se alterna el estado del pin y se realiza una espera activa de 500ms. El paquete `machine` contiene las clases que abstraen el acceso a todo el hardware del microcontrolador. Es decir, los elementos de `machine` pueden variar entre diferentes integrados, por lo que el código de MicroPython no es necesariamente portable ni agnóstico respecto a hardware a diferencia de los programas hechos en CPython.

La línea 1 del código de ejemplo 1 importa la clase `Pin` del paquete `machine`, que posteriormente será usado en la línea 4 para crear un objeto tipo `Pin`. MicroPython ocupa el paradigma de programación orientado a objetos para hacer el código más compacto y legible. Para cambiar el estado del pin basta con llamar a las funciones `on()`, `off()` o `toggle()` sin necesidad de volver a especificar el número de pin o su configuración.[4]

El estudiante avezado habrá podido notar que el código de ejemplo 1 es bastante ineficiente. La función `sleep_ms` se traducirá en una larga cadena de `NOP`s que mantendrán al procesador activo y ocupado haciendo nada, es decir,

[4]Para más información consúltese la documentación de MicroPython en https://docs.micropython.org/en/latest/library/machine.Pin.html

incapaz de realizar otras tareas. Además, a menos que se programe en ensamblador, es imposible estimar de forma precisa los tiempos de ejecución del código en lenguajes de medio y alto nivel.

Para resolver este problema, el código de ejemplo 2 hace uso de un temporizador o *timer* de SOFTWARE[5] para hacer parpadear el led (véase la línea 9 del código de ejemplo 2). En la siguiente línea el temporizador se inicializa con tres parámetros: i) `freq`, que especifica la frecuencia de operación del timer en Hertz; ii) `mode`, que indica si el timer debe correr sólo una vez (`Timer.ONE_SHOT`) o de forma periódica (`Timer.PERIODIC`) y iii) `callback`, que especifica el nombre la función que se llamará cuando el contador llegue a cero; o `blik` en nuestro caso. Para más información consúltese la documentación de MicroPython en https://docs.micropython.org/en/latest/library/machine.Timer.html

Código ejemplo 2: `blink-timer.py`: — Led parpadeante usando temporizadores

```
1  from machine import Pin      # Board IO Pin
2  from machine import Timer    # Hardware timer
3
4  def blink(timer):            # Callback function
5      led.toggle()             # Toggle the led
6  #end def
7
8  led = Pin(25, Pin.OUT)       # Setup pin 25 (sentinel LED) as output
9  timer = Timer()              # Create the Timer object
10 timer.init(freq=2.5,         # Timer frequency set to 2.5Hz
11     mode=Timer.PERIODIC,     # Timer will run endlessly (not one-shot)
12     callback=blink           # Set callback function: blink
13 )
```

Nótese que el objeto `Timer` se crea después de declararse la función `blink`. Esto es debido a que los scripts de Python se ejecutan línea a línea, por lo que el intérprete no conocerá a la función `blink` si ésta se declara después.

Por conveniencia, los códigos completos de los programas de ejemplo se encuentran en los Apéndices A y B.

## 4.5. Paso 5: Temperatura del RP2040

La ecuación (1) tiene un problema fundamental: un convertidor ADC entrega valores discretos enteros que corresponden de forma proporcional y lineal al voltaje sensado entre $V_{\text{Ref-}}$ y $V_{\text{Ref+}}$. En el caso del RP2040 $V_{\text{Ref-}} = \text{GND} = 0\text{V}$ y $V_{\text{Ref+}} = \text{VCC} = 3.3\text{V}$ y, dado que la precisión del ADC es de 12bits, los valores registrados estarían entre cero y $2^{12} - 1 = 4095$. Sin embargo, MicroPython ofrece el método `read_u16()` de las instancias de `ADC` que provee un entero no signado de 16 bits. Así, las conversiones tendrán que tomar en cuenta un rango entre 0 y 65535.

Con esto en mente y considerando que $v = \frac{3.3V}{65535}x$, se puede reemplazar $v$ por $x$ en la ecuación (1) para obtener una fórmula de conversión directa del valor reportado del ADC a un valor de temperatura.
Así, la ecuación (1) quedaría como:

$$
\begin{aligned}
T &= \left(\frac{-1}{1.721\frac{\text{mV}}{^\circ\text{C}}}\right)\left(\frac{3.3\text{V}}{65535}\cdot x\right) + 437.23^\circ\text{C} \\
&= \left(\frac{-1}{1.721\frac{\text{mV}}{^\circ\text{C}}}\right)\left(\frac{3300\text{mV}}{65535}\cdot x\right) + 437.23^\circ\text{C} \\
&= \frac{-3300\cancel{\text{mV}}}{65535 \times 1.721\frac{\cancel{\text{mV}}}{^\circ\text{C}}}\cdot x + 437.23^\circ\text{C} \\
&= \frac{-3300}{112785.74\frac{1}{^\circ\text{C}}}\cdot x + 437.23^\circ\text{C}
\end{aligned}
$$

Lo que finalmente produce la ecuación:

[5] Nótese que no se le proporciona el ID del timer a utilizar al constructor. Esto es debido a que la implementación actual del MicroPython para el RP2040 aún no tiene soporte para el uso de los timers físicos (Hardware) del integrado.

$$T = -29259 \times 10^{-6} x + 437.23 \tag{2}$$

Esta ley se refleja en el código presentado en código de ejemplo 3.

Ingrese el siguiente código en el editor de Thonny y ejecútelo

Código ejemplo 3: `temp.py`: — Temperatura del RP2040

```
from machine import ADC           # Board Analogic-to-Digital Converter
from utime import sleep_ms        # Delay function in milliseconds

def main():                       # Main function
    K = -0.029259019              # Conversion factor
    adc = machine.ADC(4)          # Init ADC on pin 4
    while(True):                  # Repeat forever
        x = adc.read_u16()        # Read ADC
        temp = x * K + 437.23     # Convert to celcius
        print(f'Temp: {temp}°C') # Print temperature
        sleep_ms(1000)            # Wait for 1000ms
#end def

if __name__ == '__main__':
    main()
```

El código de ejemplo 3 configura primero el ADC para leer del canal 4 (sensor interno de temperatura) y luego define un bucle infinito con la cláusula **`while`** dentro del cual i) se lee el valor del convertidor A/D, ii) se convierte el valor sensado a temperatura en grados celcius iii) se imprime la temperatura y, finalmente, iv) se espera durante un segundo.

Por conveniencia, los códigos completos de los programas de ejemplo se encuentran en el Apéndice C.

# 5. Experimentos

1. [6pt] Modifique el código de la subsección 4.4 para que el brillo del led centinela se vea atenuado.
2. [4pt] Modifique el código de la subsección 4.5 para que la temperatura se reporte tanto en grados Farenheit como en Centígrados.
3. [+3pt] Con base en lo aprendido, modifique el código de la subsección 4.4 atenúe el brillo del led usando un PWM en lugar de retardos.
4. [+2pt] Usando Thonny cargue un archivo `main.py` en el RP2040 para que su programa se inicie de forma automática y autónoma.

**Nota:** Anexe en su reporte las imágenes o enlaces a videos correspondientes a cada experimento como evidencia de su ejecución.

## 6. Referencias

# Referencias


[1] Charles Bell. *Beginning MicroPython with the Raspberry Pi Pico: Build Electronics and IoT Projects*. Springer, 2022.

[2] Charles Bell. Introducing the raspberry pi pico. In *Beginning MicroPython with the Raspberry Pi Pico: Build Electronics and IoT Projects*, pages 1–42. Springer, 2022.

[3] Charles Bell. Introducing micropython. In *Beginning MicroPython with the Raspberry Pi Pico: Build Electronics and IoT Projects*, pages 43–84. Springer, 2022.

[4] *RP2040 Datasheet: A microcontroller by Raspberry Pi*. Raspberry Pi, 2023.

[5] Aivar Annamaa. Thonny, a python ide for learning programming. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 343–343, 2015.

[6] Aivar Annamaa. Introducing thonny, a python ide for learning programming. In *Proceedings of the 15th koli calling conference on computing education research*, pages 117–121, 2015.

[7] Ignas Plauska, Agnius Liutkevičius, and Audronė Janavičiūtė. Performance evaluation of c/c++, micropython, rust and tinygo programming languages on esp32 microcontroller. *Electronics*, 12(1):143, 2022.

[8] Gabriel Gaspar, Peter Fabo, Michal Kuba, Juraj Dudak, and Eduard Nemlaha. Micropython as a development platform for iot applications. In *Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1 9*, pages 388–394. Springer, 2020.

# A. Programa Ejemplo: `blink.py`

src/blink.py

```
from machine import Pin      # Board IO Pin
from utime import sleep_ms   # Delay in milliseconds

led = Pin(25, Pin.OUT)       # Setup pin 25 (sentinel LED) as output

while(True):                 # Repeat forever
    led.toggle()             # Toggle the led
    sleep_ms(500)            # Wait for 500ms
```

# B. Programa Ejemplo: `blink-timer.py`

src/blink–timer.py

```
from machine import Pin      # Board IO Pin
from machine import Timer    # Hardware timer

def blink(timer):            # Callback function
    led.toggle()             # Toggle the led
#end def

led = Pin(25, Pin.OUT)       # Setup pin 25 (sentinel LED) as output
timer = Timer()              # Create the Timer object
timer.init(freq=2.5,         # Timer frequency set to 2.5Hz
    mode=Timer.PERIODIC,     # Timer will run endlessly (not one-shot)
    callback=blink           # Set callback function: blink
)
```

# C. Programa Ejemplo: `temp.py`

src/temp.py

```
from machine import ADC           # Board Analogic-to-Digital Converter
from utime import sleep_ms        # Delay function in milliseconds

def main():                       # Main function
    K = -0.029259019              # Conversion factor
    adc = machine.ADC(4)          # Init ADC on pin 4
    while(True):                  # Repeat forever
        x = adc.read_u16()        # Read ADC
        temp = x * K + 437.23     # Convert to celcius
        print(f'Temp: {temp}°C') # Print temperature
        sleep_ms(1000)            # Wait for 1000ms
#end def

if __name__ == '__main__':
    main()
```