Primeiros Passos com o Thonny e MicroPython

1. Introdução à Estrutura Básica de um Programa em MicroPython

- **Definição:** Todo programa em MicroPython começa com a importação das bibliotecas necessárias e, em seguida, a configuração e manipulação dos pinos GPIO. Inicialmente utilizaremos duas bibliotecas do micropython: ***machine; time.***

- **Importação das Bibliotecas:**
  - **Importando toda biblioteca:**

    import machine
    import time

    Exemplo
    import machine
    import time

- **Importando apenas as classes e métodos necessários:**

    from machine import Pin
    from time import sleep

    Exemplo
    from machine import Pin
    from time import sleep

  Explicação: O uso de from ... import ... permite importar apenas as partes específicas das bibliotecas que serão utilizadas, economizando memória e tornando o código mais claro.

- **Estrutura while em MicroPython**
  A estrutura de controle while é usada para repetir um bloco de código enquanto uma condição for verdadeira. Em MicroPython, assim como em outras linguagens de programação, o while permite a criação de loops que podem executar um conjunto de instruções repetidamente até que a condição definida se torne falsa ou seja interrompida manualmente.

  Sintaxe básica:

  while ***condição***:
    # bloco de código que será repetido

- **Comentários em MicroPython**
  Comentários são trechos de texto no código que são ignorados pelo interpretador, servindo para documentar e explicar o código para quem o lê. Eles são extremamente úteis para tornar o código mais compreensível, tanto para o programador que o escreveu quanto para outras pessoas que possam trabalhar com ele no futuro.
  Em MicroPython, assim como em outras linguagens de programação, os comentários são precedidos pelo símbolo #.

2. **Configurando pinos como entrada**

- **Sem PULL-UP ou PULL-DOWN:**

```
pino_entrada = Pin(14, Pin.IN)
```

Explicação: Neste exemplo, o pino GPIO 14 é configurado como entrada (Pin.IN). Sem resistores de PULL-UP ou PULL-DOWN, o estado do pino pode flutuar se não estiver conectado a uma fonte de sinal clara.

  - **Com PULL-UP:**

```
pino_entrada_pullup = Pin(5, Pin.IN, Pin.PULL_UP)
```

Explicação: Configura o pino GPIO 5 como entrada com resistor de PULL-UP interno. Isso significa que, por padrão, o pino estará em nível lógico alto (1) quando não estiver conectado a nada.

- **Com PULL-DOWN:**

Explicação: Configura o pino GPIO 16 como entrada com resistor de PULL-DOWN interno. Por padrão, o pino estará em nível lógico baixo (0) quando não estiver conectado a nada.

3. **Usando o Método sleep para Delays**

- **Introdução ao sleep:**

```
sleep(1)
```

Explicação: A função sleep pausa a execução do programa por um número específico de segundos. Neste exemplo, o programa pausa por 1 segundo.

4. ***Controle do LED conectado ao GPIO 2***

Exemplo1 - Usando os Métodos on() e off() em Alternância
Aqui, alternamos manualmente entre os métodos on() e off() dentro do loop:

```
from machine import Pin
from time import sleep

# Configuração do LED embutido no Raspberry Pi Pico
led = Pin(2, Pin.OUT)

while True:
    led.on()  # Liga o LED
    sleep(1)  # Espera por 1 segundo
    led.off()  # Desliga o LED
    sleep(1)  # Espera por 1 segundo
```

Exemplo2 - Usando o Método value() para Ligar e Desligar o LED
Aqui, vamos alternar manualmente o estado do LED utilizando value(1) para ligar e value(0) para desligar o LED.

```
from machine import Pin
from time import sleep

# Configuração do LED embutido no Raspberry Pi Pico
```

```
led = Pin(2, Pin.OUT)

while True:
    led.value(1)  # Liga o LED
    sleep(1)      # Espera por 1 segundo
    led.value(0)  # Desliga o LED
    sleep(1)      # Espera por 1 segundo
```

<u>Resumo das duas abordagens:</u>

1. **value()**: Usado para ler e definir o estado do pino manualmente.
2. **on() e off()**: Oferecem controle explícito sobre ligar e desligar o pino.

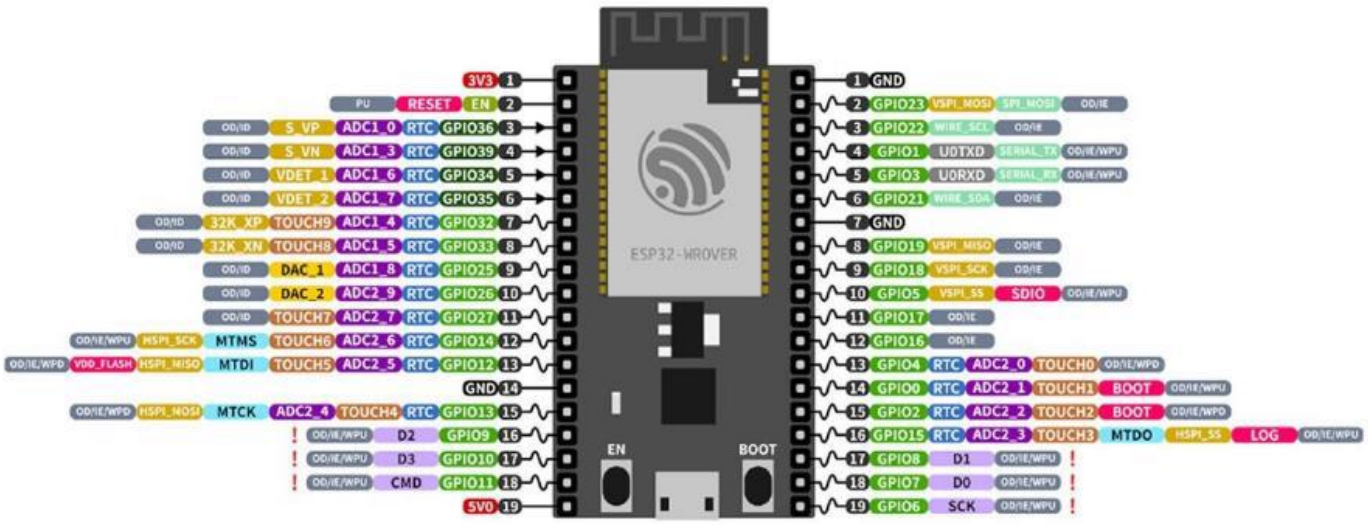
Esses métodos permitem diferentes abordagens para alcançar o mesmo objetivo de alternar o estado do LED, dependendo da necessidade de leitura ou controle explícito.

Exemplo4 – Configurar dois pinos como entrada, um com PULL-UP e outro com PULL-DOWN, e monitorar seus estados em um loop.

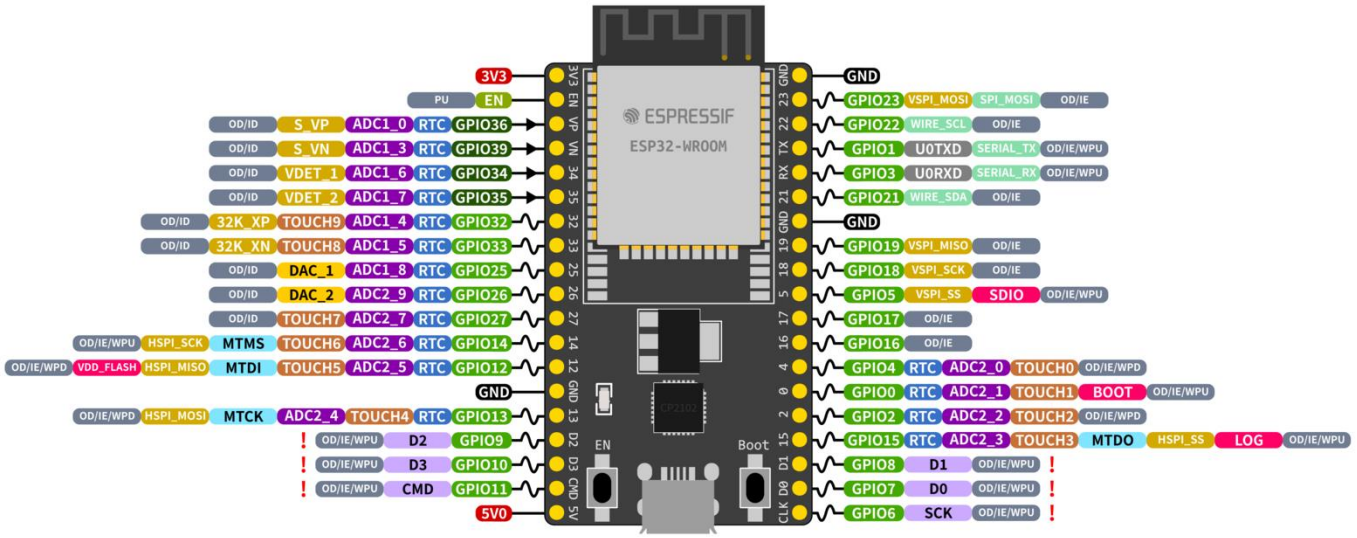
```
from machine import Pin
from time import sleep

pino_pullup = Pin(5, Pin.IN, Pin.PULL_UP)
pino_pulldown = Pin(4, Pin.IN, Pin.PULL_DOWN)

while True:
    estado_pullup = pino_pullup.value()
    estado_pulldown = pino_pulldown.value()
    print("PULL-UP:", estado_pullup, "PULL-DOWN:", estado_pulldown)
    sleep(1)
```



ESP32-DevKitC



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions

RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE

WPU: *Weak Pull-up (Internal)*
WPD: *Weak Pull-down (Internal)*
PU: *Pull-up (External)*
IE: *Input Enable (After Reset)*
ID: *Input Disabled (After Reset)*
OE: *Output Enable (After Reset)*
OD: *Output Disabled (After Reset)*