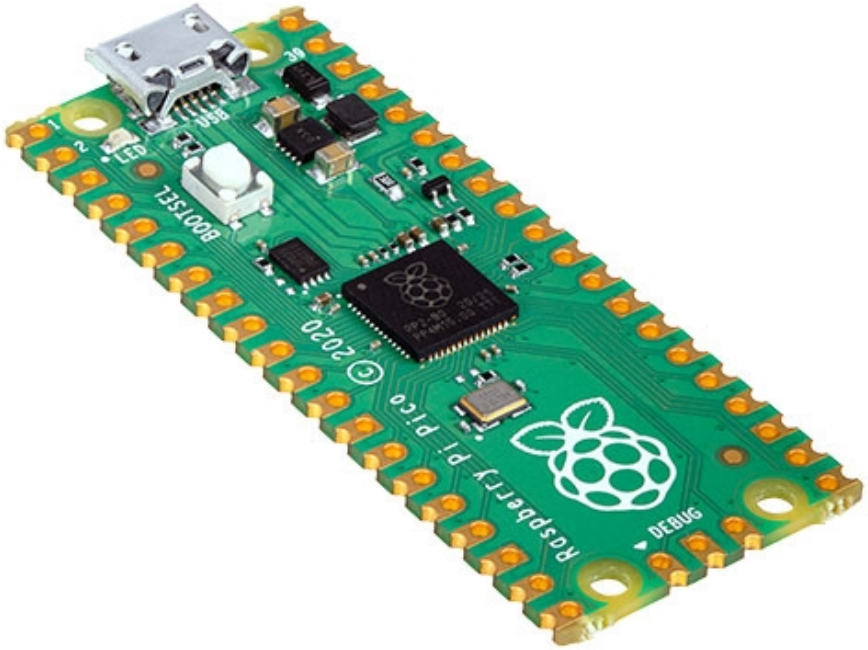
RASPBERRY PI PICO

O Raspberry Pi Pico é uma placa microcontroladora bem semelhante ao Arduino, mas podemos programá-lo na linguagem Python utilizando um computador. A placa conta com 256 KB de memória RAM, 2 MB de memória Flash, sensor de temperatura on-board e um RTC (Real Time Counter) de alta precisão. A Pi Pico possui uma GPIO de 40 pinos, sendo que 26 deles são multifuncionais. Há pinos para comunicação SPI, I2C e UART, 3 entradas ADC com precisão de 12 bits e 16 canais PWM.



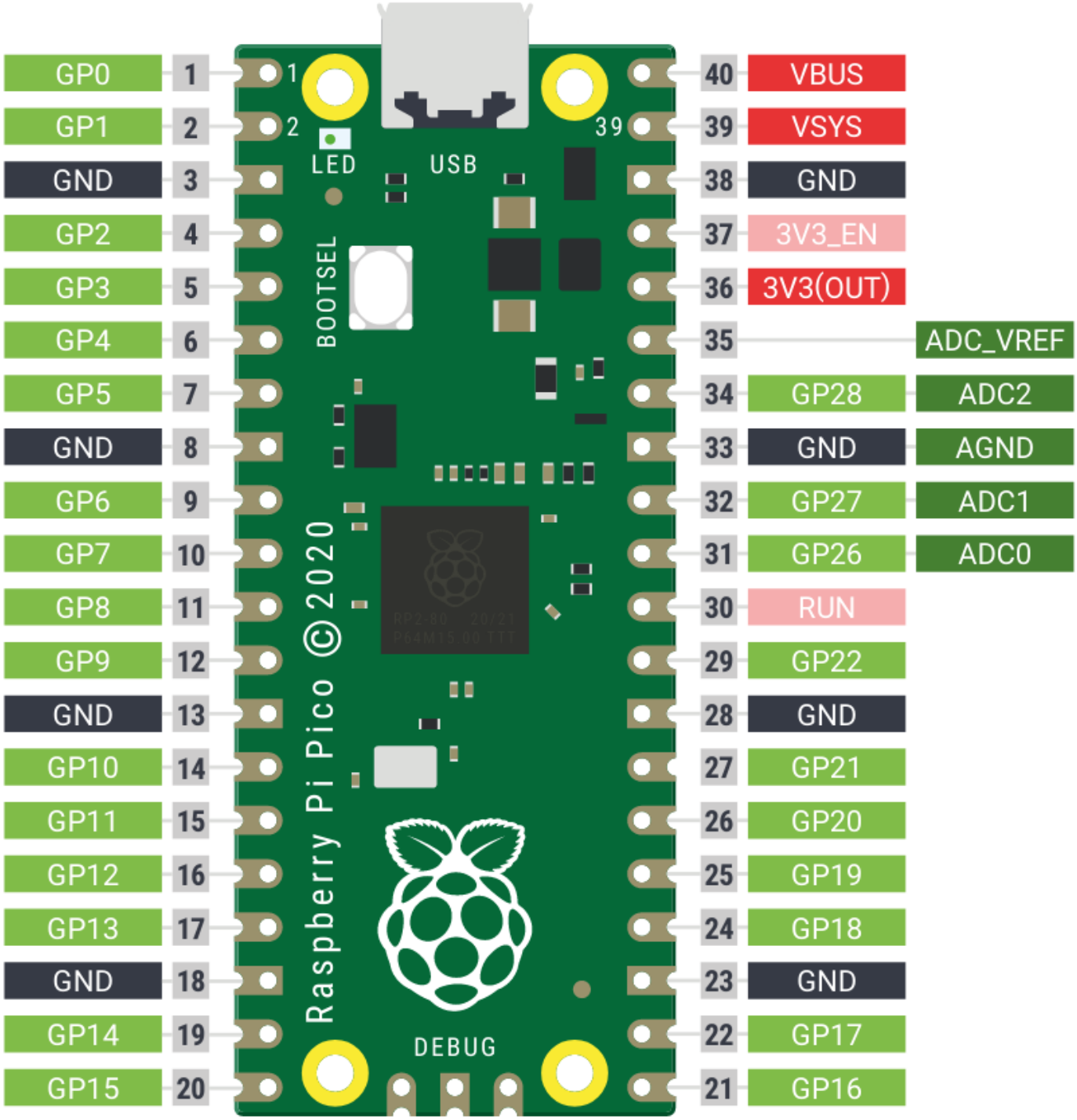
Cada pino GPIO envia até 3.3 Volts com até 16 mA de corrente. Com esta placa, você pode criar projetos em Python e C/C++, de forma simples e bastante prática.

PINAGEM

No Raspberry Pi Pico, os pinos GPIO são para envio de sinais digitais e analógicos (PWM) e são enumerados pelo nome do pino (se quisermos, por exemplo, usar o GPIO15, vamos usar o número 15 na programação).

Os pinos ADC servem para a leitura de sinais analógicos, como por exemplo a voltagem enviada por um potenciômetro que pode ser regulada. Tais pinos podem então ler uma tensão de 0 a 3.3 Volts, mas essa funcionalidade está apenas nos pinos GPIO com o ADC (vermelho).

A figura abaixo mostra a pinagem do Raspberry PICO com os pinos GPIO e ADC.



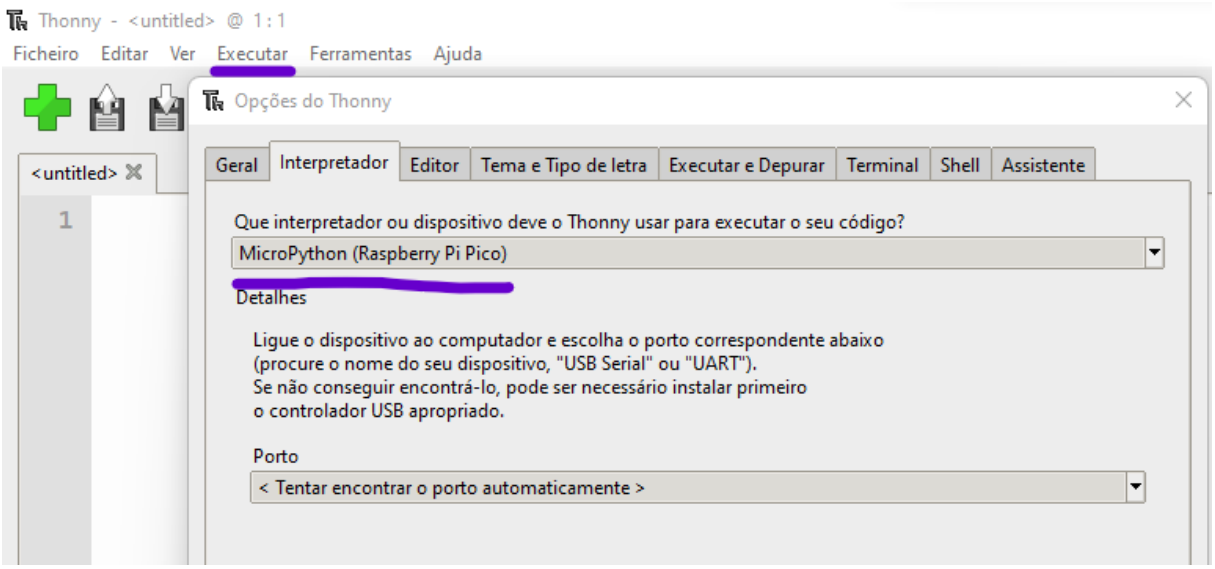
INSTALANDO O THONNY

Para programar no Raspberry Pi Pico, vamos utilizar o compilador Thonny, onde vamos programar na linguagem MICRO-PYTHON, recebendo funcionalidades extras para envio de sinais elétricos. Para instalar o software, vá no link:

- https://thonny.org/

E baixe a versão para Windows, como mostrado na figura abaixo. Após baixado, execute o instalador e instale normalmente o aplicativo para poder abri-lo, futuramente.

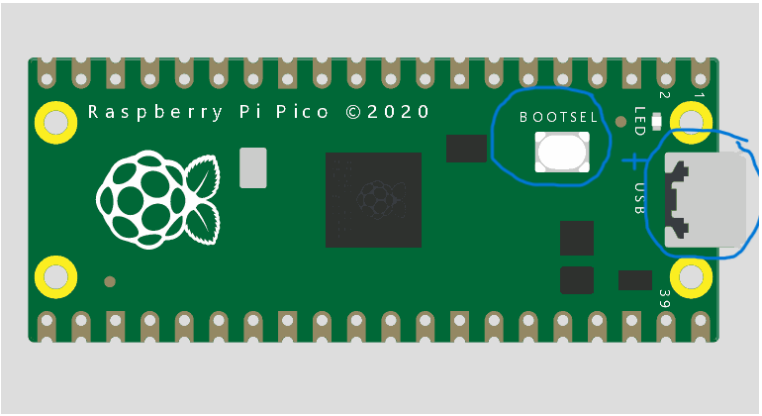
Abrindo o Thonny, vá em **EXECUTAR** e depois em **ESCOLHER O INTERPRETADOR**, onde escolheremos o **MicroPython (Raspberry Pi Pico)**



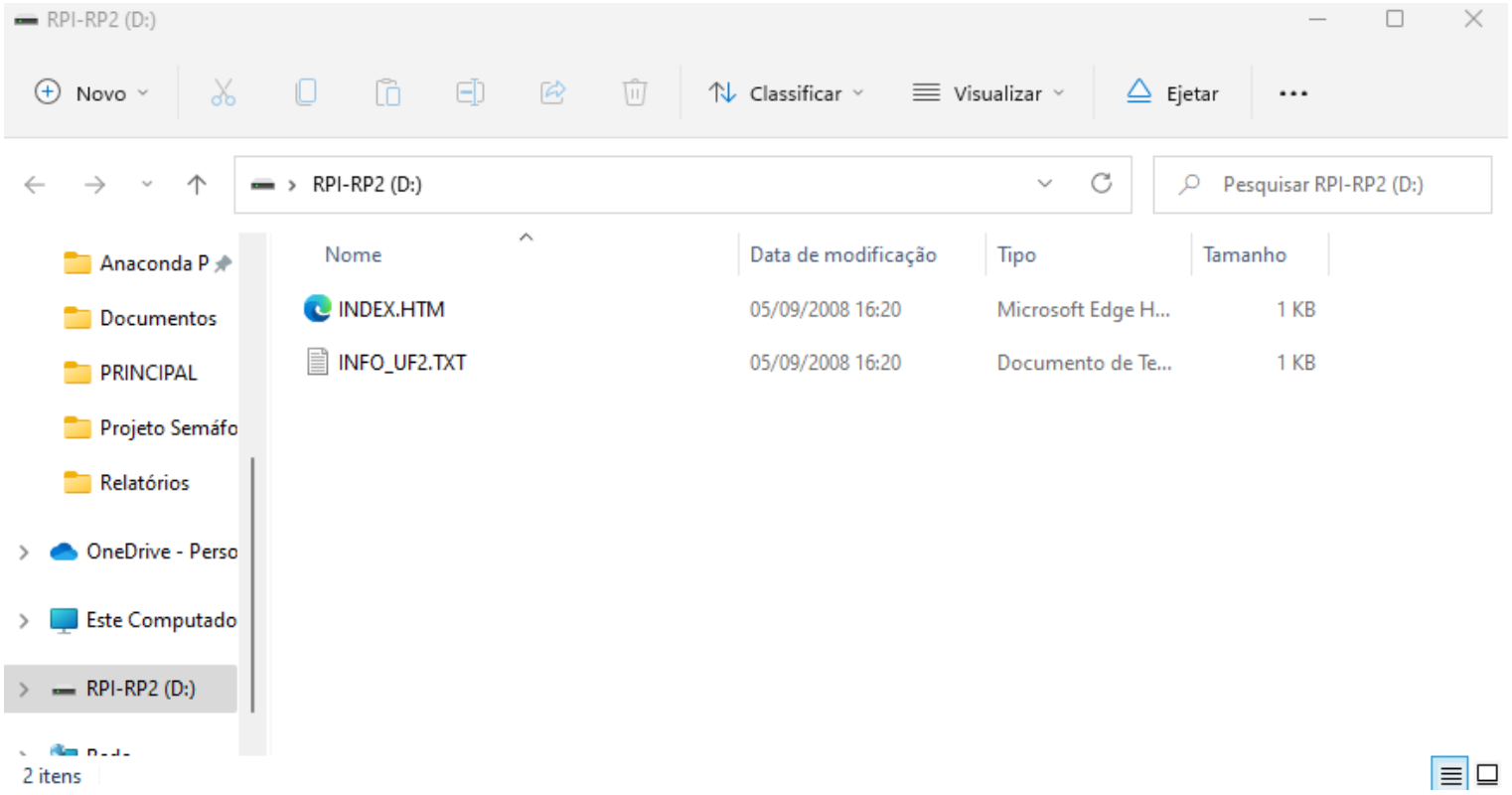
Podemos então dar OK e podemos inserir o Raspberry Pico com o cabo USB.

CONEXÃO COM O RASPBERRY

Após a instalação do Thonny e configuração do interpretador, poderemos agora conectar o cabo USB do Raspberry no computador, mas faremos isso pressionando o botão **BOOTSEL** ao mesmo tempo

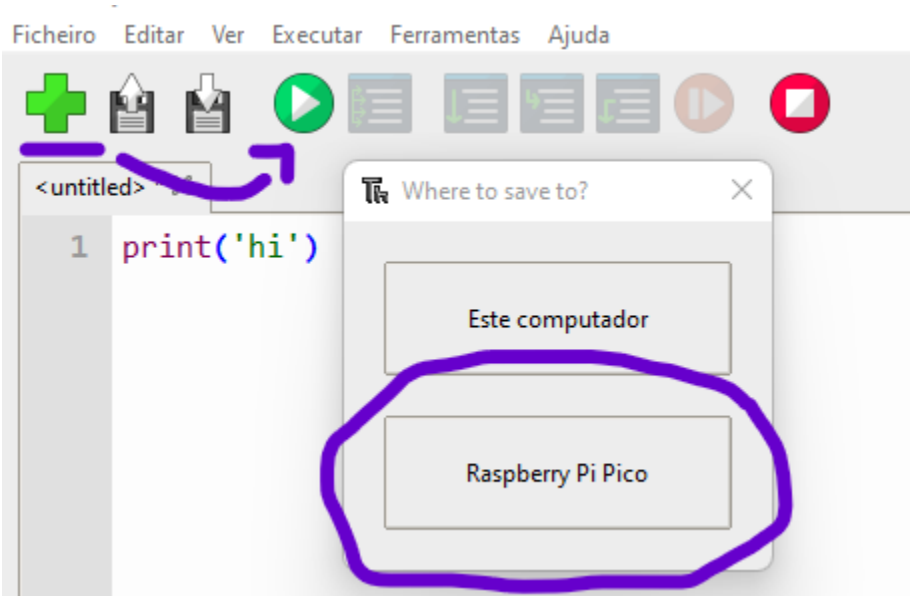


Após esse procedimento, o armazenamento dessa placa estará disponível para você acessar pelo computador como mostrado na figura abaixo

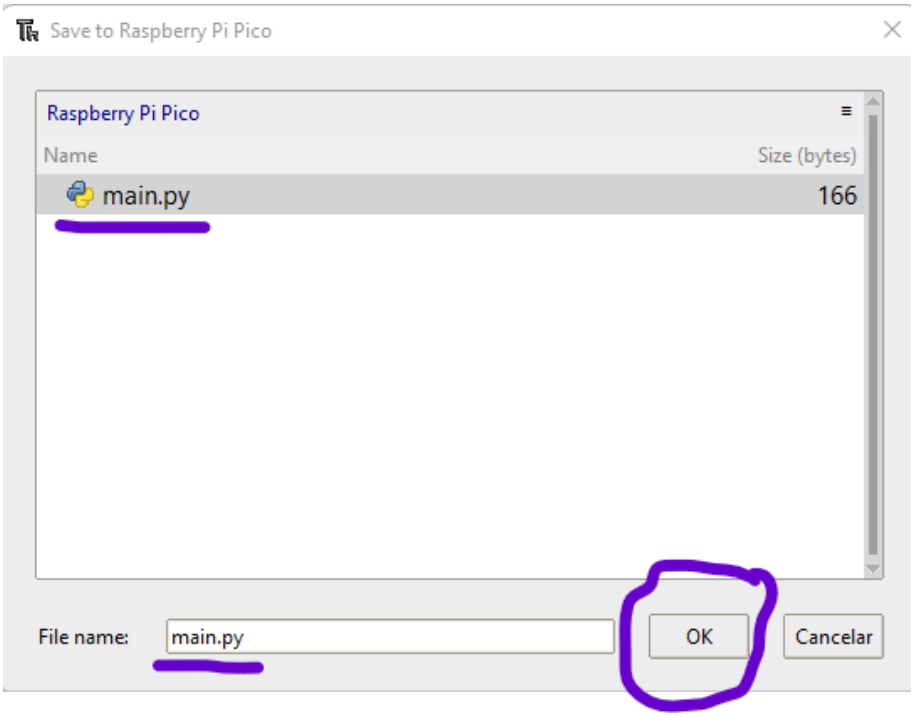


Se for a primeira vez que você está usando esse Raspberry (conectando o cabo), vamos instalar o arquivo que permitirá a leitura de códigos .py. Abra a pasta dos arquivos da placa, baixe o arquivo rp2-pico-20220117-v1.18.uf2 disponível no drive, e cole-o nessa pasta. Após isso, o Raspberry irá reiniciar e já podermos programá-lo na linguagem Micro-Python.

Agora no compilador Thonny aparecerá uma mensagem pedindo para instalar o Raspberry Pico quando o código for executado. **Para salvar um código no Raspberry**, crie um novo arquivo e execute ele, salvando na própria placa.

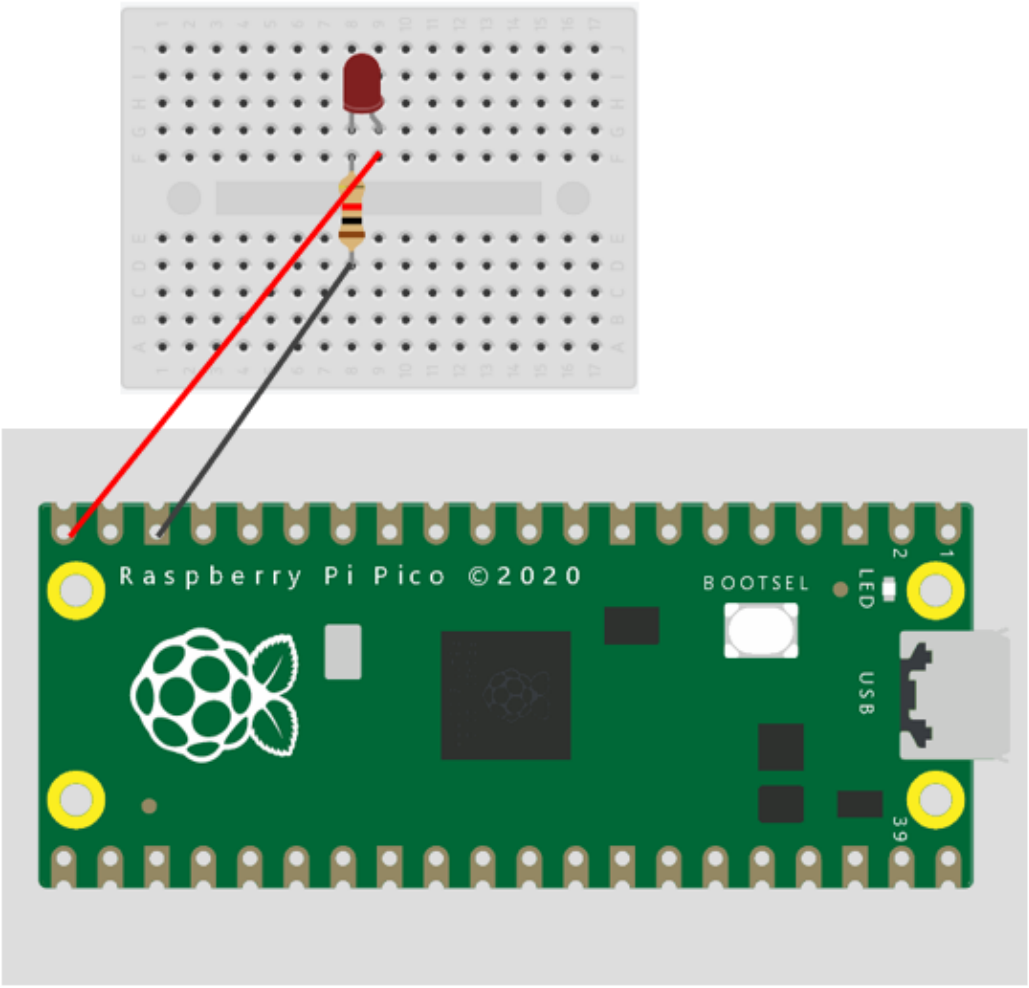


Para que o código seja lido e executado pelo Raspberry sempre que ele for ligado, salve o código em um arquivo main.py para ativar essa funcionalidade.



ACENDENDO LED

Vamos agora acender um LED com o circuito básico, onde a corrente chega do pino GPIO 15 (pino 20) com 3.3 Volts, passa pelo LED, passa pelo resistor e vai para o negativo (GND), consulta a pinagem para analisar o circuito.



O código está mostrado na figura abaixo, onde o LED acende e apaga a cada 1 segundo

```
from machine import *
from time import time, sleep

# DECLARANDO OS PINOS DE SAÍDA
LED_PADRAO = Pin(25, Pin.OUT)
LED = Pin(15, Pin.OUT)

# ATIVANDO O LED DO PICO
LED_PADRAO.value(1)
print('INICIANDO...')

# LOOP INFINITO COM O LED
while True:
    LED.value(1)
    sleep(1)

    LED.value(0)
    sleep(1)
```

USANDO OS PINOS PWM

A função PWM está presente em todos os pinos GPIO do Raspberry Pico, onde podemos enviar uma tensão específica de 0 a 3.3 Volts por um sinal analógico.

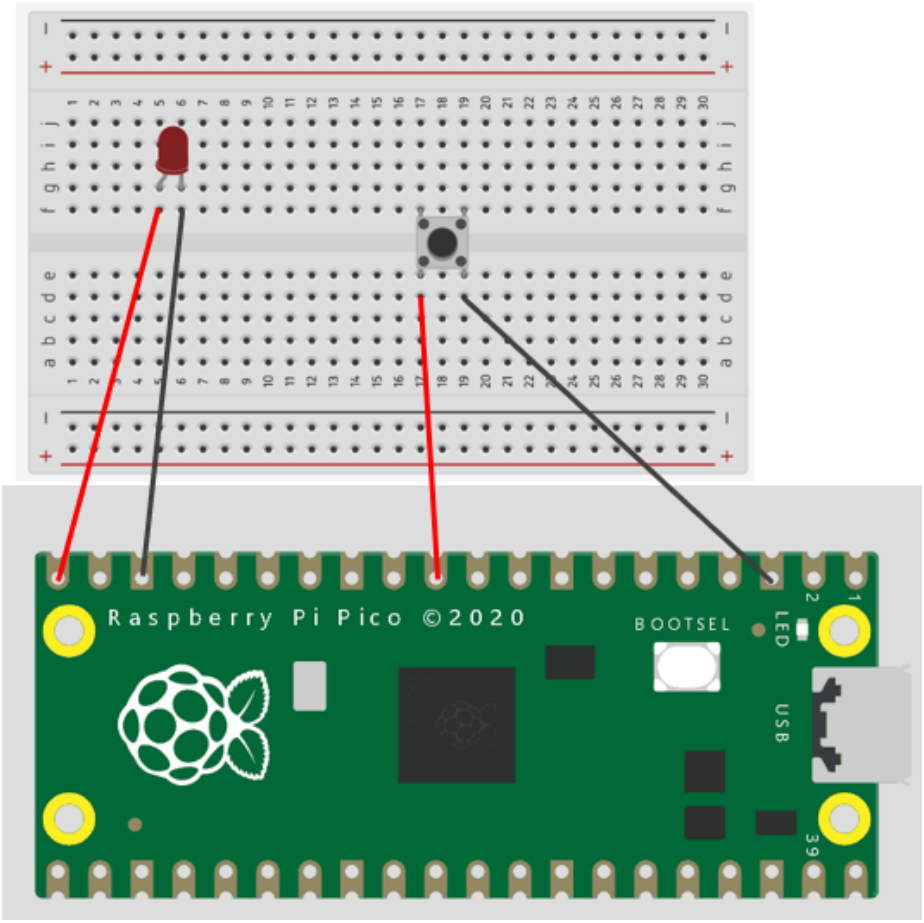
primeiro precisamos declarar o pino do LED como um pino PWM, e então estabelecendo sua frequência para 1000, ou seja, o LED vai usar uma frequência de 1000 sinais digitais por segundo para o envio do sinal analógico.

```
from machine import *
from time import time, sleep, sleep_ms

pwm = PWM(Pin(15))
pwm.freq(1000)

while True:
    for x in range(0, 65025, 1000):
        pwm.duty_u16(x)
        sleep_ms(30)
        print(x)

    for x in range(65025, 0, -1000):
        pwm.duty_u16(x)
        sleep_ms(30)
        print(x)
```

A função duty_u16(sinalAnalogico) recebe um valor de 0 a 65025 que é justamente a intensidade do brilho do LED, que aumenta no laço for para aumentar o sinal analógico em 1000 a cada repetição, amentando assim o brilho do LED a cada 30 milissegundos.

BOTÕES

O botão funciona como uma chave, quando ninguém o aperta essa chave está aberta e nenhuma corrente passa sobre ele.

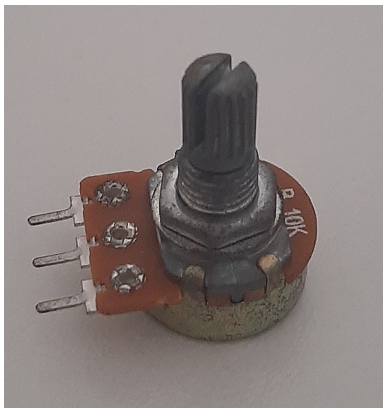


Como vamos **ler** um dado digital do botão, vamos declará-lo como um pino de entrada (INPUT), e também como um pino INPUT PULLUP para evitar pontos flutuantes e falsas leituras, mas declará-lo como INPUT_PULLUP irá inverter a leitura, fazendo com que o 0 seja **botão pressionado** e 1 seja **botão não pressionado**.

```
from machine import *
from time import time, sleep, sleep_ms

LED = Pin(15, Pin.OUT)
BOTAO = Pin(9, Pin.IN, Pin.PULL_UP)

while True:
    estado = BOTAO.value()

    if estado == 0:
        print('BOTÃO PRESSIONADO!')
        LED.value(1)
    else:
        LED.value(0)
```

USANDO POTENCIÔMETROS

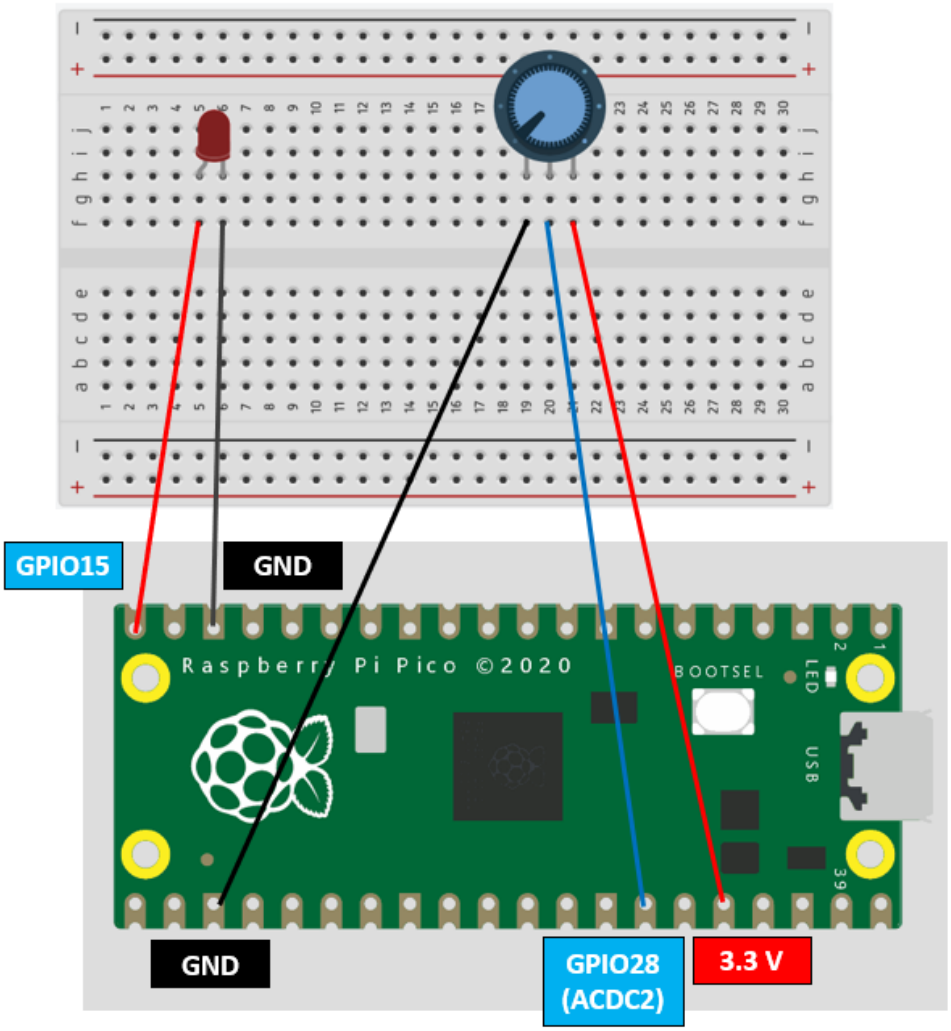
O potenciômetro é um componente onde é possível ajustar a sua resistência elétrica, então podemos lerseus valores e usá-los em nossos projetos.



Potenciômetro

Agora usaremos um potenciômetro com o LED, o potenciômetro é conectado em um pino analógico do Raspberry PICO, sendo os pinos GPIO com a função ADC que está escrito em vermelho na filmagem. Nele é possível ajustar manualmente a voltagem que eu mando para o circuito girando a sua rodinha.

O potenciômetro tem 3 pinos, sendo o do meio o pino ADC/analógico que lerá o valor ajustado manualmente, os pinos dos cantos são o GND e o VCC (3.3 Volts), onde podemos conectá-los em qualquer ordem, mas isso irá inverter o sentido de crescimento da rodinha.



O potenciômetro gera um valor entre 0 e 65025 que é lido pelo comando read_u16(), e o comando duty_u16 (intensidade) vai enviar um pulso entre 0 (0 Volts) até 65025 (3.3 Volts). Perceba que para ajustar o brilho do LED devemos colocá-lo em um pino PWM. O potenciômetro deve receber o sinal do pino analógico, mais os 3.3V e GND.

```
from machine import *
from time import time, sleep, sleep_ms

LED = PWM(Pin(15))    # PINO GPIO 15 DO LED
LED.freq(1000)        # FREQUÊNCIA PWM

POT = ADC(Pin(28))    # PINO ADC2 (GPIO 28) DO POTENCIÔMETRO

while True:
    intensidade = POT.read_u16()
    print('INTENSIDADE: ', intensidade)

    LED.duty_u16(intensidade)
```

Mapear

Percebemos que a leitura do potenciômetro está gerando valores entre 500 e 65025. Podemos também usar uma função para mapear os valores lidos para serem ajustados para um intervalo desejado (0 a 65000) usando a equação da reta.

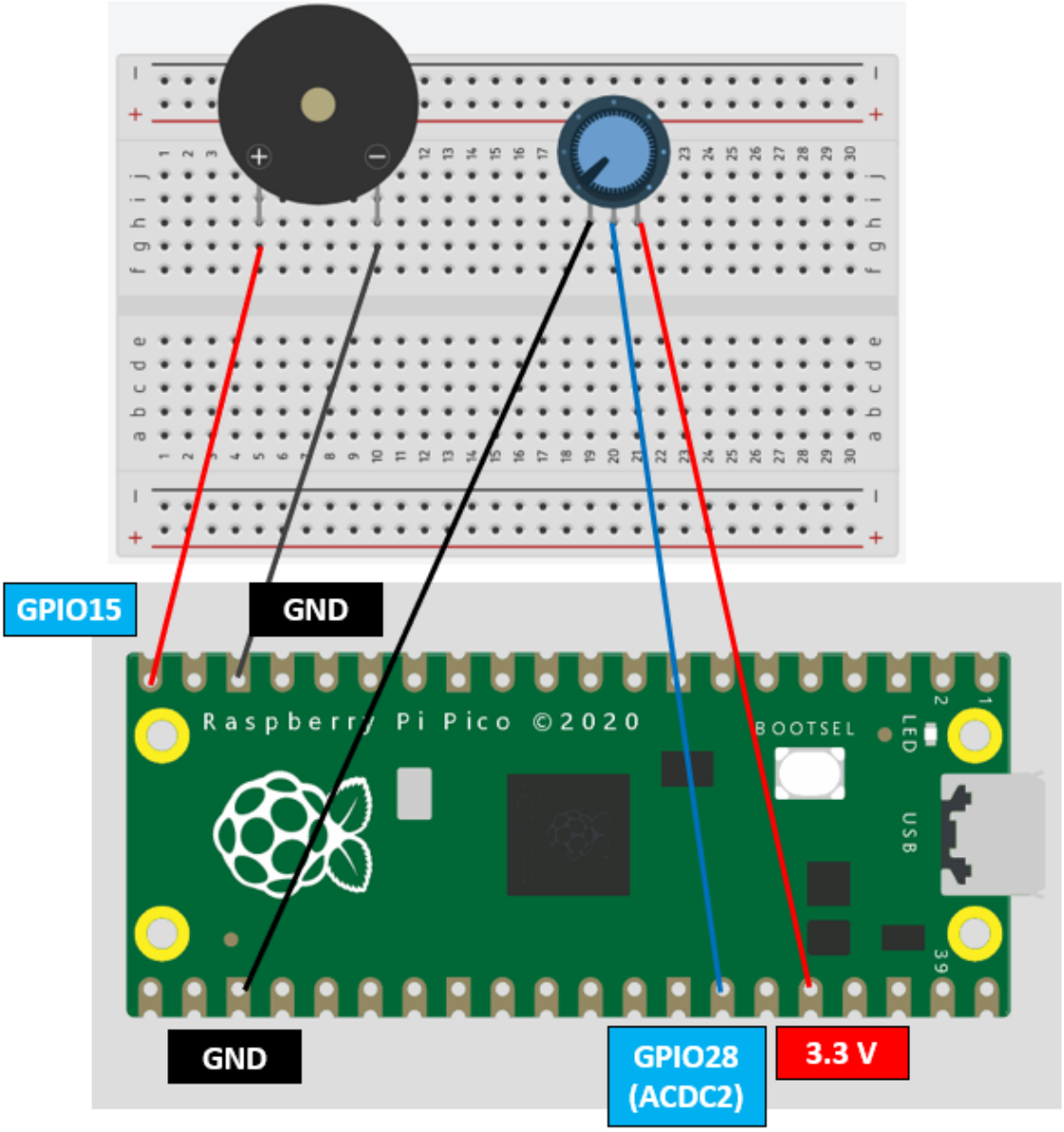
```
from machine import *
from time import time, sleep, sleep_ms

LED = PWM(Pin(15))    # PINO GPIO 15 DO LED
LED.freq(1000)        # FREQUÊNCIA PWM

POT = ADC(Pin(28))    # PINO ADC2 (GPIO 28) DO POTENCIÔMETRO

def mapear(sinal, Xo, X, Yo, Y):
   if sinal < Xo:
       return Yo
   if sinal > X:
       return Y
   return int((Y - Yo)/(X - Xo)*(sinal - Xo) + Yo)

while True:
    intensidade = POT.read_u16()
    intensidade = mapear(intensidade, 500, 65025, 0, 65000)

    print('INTENSIDADE: ', intensidade)
    LED.duty_u16(intensidade)
```

BUZZER

Os sonorizadores são dispositivos que têm a capacidade de produzir sons com quando são submetidos a uma certa voltagem, e faremos isso usando o Arduino.

Buzzer

Para usar o sonorizador no Raspberry PICO, lembre-se de que alguns possuem polaridade, ou seja, o lado positivo deve ser conetado aos 3.3 V e o negativo ao GND, se ficar muito barulhento, você pode adicionar resistores ao circuito a fim de diminuir o volume que o sonorizador faz. Vamos agora substituir o LED do projeto anterior por um buzzer, e assim, controlar o volume do sonorizador com o valor ajustado.



```
from machine import *
from time import time, sleep, sleep_ms

BUZZER = PWM(Pin(15))    # PINO GPIO 15 DO BUZZER
BUZZER.freq(1000)        # FREQUÊNCIA PWM

POT = ADC(Pin(28))    # PINO ADC2 (GPIO 28) DO POTENCIÔMETRO

def mapear(sinal, Xo, X, Yo, Y):
   if sinal < Xo:
       return Yo
   if sinal > X:
       return Y
   return int((Y - Yo)/(X - Xo)*(sinal - Xo) + Yo)

while True:
    intensidade = POT.read_u16()
    intensidade = mapear(intensidade, 500, 65025, 0, 65000)

    print('INTENSIDADE: ', intensidade)
    BUZZER.duty_u16(intensidade)
```