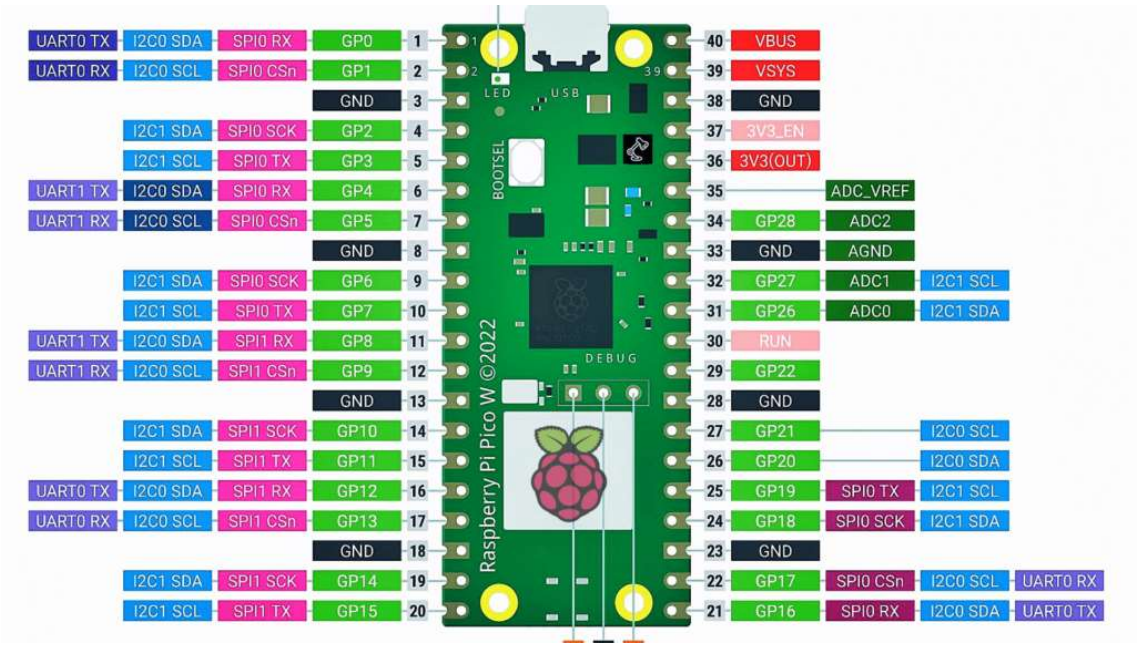
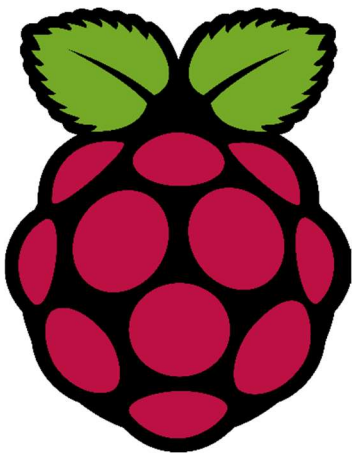
Raspberry Pi





Sumário

1. Introdução 4
1.1 O que é Raspberry Pi? 4
1.2 Sobre o Kit Maker Raspberry Pi Pico W 5
1.3 Instruções 7
2. Raspberry Pi Pico W 7
2.2 Pinos 10
3. Teoria básica de circuito e boas práticas 12
3.1 Circuito Eletrônico 12
3.2 Protoboard 14
3.3 Cuidados 17
4. O que é MicroPhyton? 19
4.1 Introdução ao MicroPhyton 19
4.2 Instalando a IDE Thonny 19
4.3 Instalando a MicroPhyton na sua Pico W 20
4.4 Upload das Bibliotecas na Pico W 21
4.6 Piscar o LED da Raspberry Pi Pico W 22
5. LED 24
5.1 O que é um LED? 25
5.2 O que é um resistor? 26
5.3 Pisca Pisca 28
5.4 LED oscilante 33
5.5 O que é um LED RGB Endereçável? 37
5.6 Acionando um Anel LED RGB 38
6. Botão 41
6.1 O que é o Módulo Sensor de Chave TTP224? 41
6.2 Ligando uma luminária com botão 42
6.3 Teclado de cores 46
7. Sensor de Movimento 51
7.1 O que é o Mini Sensor de Movimento Presença PIR? 51
7.2 Detector de movimento 52
8. Sensor de Luminosidade 56

8.2 Sensor de luz ambiente 57
9. Sensor de Umidade e Temperatura 60
9.1 O que é o Sensor de Umidade e Temperatura DHT11? 60
9.2 Termômetro luminoso 61
9.3 Higrômetro luminoso 66
10. Buzzer 71
10.1 O que é o Buzzer? 71
10.2 Acionando um Buzzer 73
10.3 Fazendo música 75
10.4 Theremim 79
11. Display 81
11.1 O que é o Display LCD? 81
11.2 Sua mensagem no display 83
11.3 Montando uma TV 85
12. Leitor Rfid 91
12.1 O que é o Módulo Leitor Rfid? 91
12.2 Acionando o Leitor Rfid 92
13. Microfone 95
13.1 O que é o Módulo Amplificador de Microfone? 95
13.2 Acendendo uma luminária com sons 96
13.3 Reagindo aos sons 99
14. Projetos para consolidar conhecimentos 102
14.2 Sirene 105
14.3 Contagem de passagem 115
14.4 Jukebox 118
14.5 Controle de acesso 121
15. Projetos IoT 131
15.1 O que é IoT? 131
15.2 Acessando a rede 132
15.3 Conectando à luz do @CheerLigths 138
15.11 Sistema de segurança 141

1. Introdução

Bem-vindo ao guia do **Kit Maker Raspberry Pi Pico W**! Aqui vamos te ajudar a dar os primeiros passos com esta placa extremamente compacta e versátil. Ao longo do guia daremos uma breve introdução à programação em MicroPhyton e conceitos básicos em eletrônica para montar seus primeiros circuitos, interagir com o mundo e ir além das linhas de código. Após concluir o curso você estará apto a aplicar de diferentes formas as habilidades de programação e eletrônica que foram aprendidas por aqui.

1.1 O que é Raspberry Pi?

A Fundação Raspberry Pi é uma instituição de caridade sediada no Reino Unido, fundada no ano de 2009 e criadora das placas com o mesmo nome. Seu grande propósito é empoderar pessoas em todo o mundo para que possam moldar um futuro cada vez mais digital, onde todos sejam capazes de resolver problemas que realmente importam, além de prepará-las para os empregos do futuro.

Para que isso aconteça, a Fundação Raspberry Pi fornece computadores de baixo custo e bom desempenho, que as pessoas usam para aprender, resolver problemas e se divertir. Também fornecem conteúdos educativos para que todas as pessoas possam aprender sobre computação e também como usar computadores em diferentes aplicações, além de treinar educadores que possam orientar outras pessoas a aprenderem.

Ao longo desses anos, a Fundação Raspberry Pi lançou mais de 10 modelos de placas, além de diversos acessórios oficiais como cases, teclado, mouse, câmeras e display.

Agora que você já sabe o que é a Raspberry Pi, você pode estar se perguntando “mas o que é essa história de Maker?”. O movimento maker representa a cultura conhecida como “faça você mesmo” (DIY – do it yourself, em inglês). A proposta é promover o estímulo de pessoas comuns para que explorem sua criatividade e possam desenvolver suas próprias soluções. A cultura maker se nutre da iniciativa de construir, consertar, modificar ou fabricar suas próprias coisas. Os adeptos deste movimento são conhecidos como makers ou fazedores.

Neste curso serão apresentados os primeiros passos com a Raspberry Pi, noções básicas de programação em MicroPython e uma introdução sobre eletrônica com projetos práticos. O conteúdo do curso é dividido em 16 **módulos com 59 aulas** .

Em algumas delas serão trazidos conteúdos de embasamento teórico, já em outras iremos aplicar esses conteúdos ensinando a criar projetos completos utilizando a placa.

É interessante prestar muita atenção aos procedimentos apresentados, tentar alterar alguns parâmetros e ver o que acontece em cada pequena mudança. Aos poucos você terá mais domínio sobre a placa, o que vai facilitar na hora de criar os seus próprios projetos.

1.2 Sobre o Kit Maker Raspberry Pi Pico W

O Kit Raspberry Pi Pico W oferece uma experiência perfeita para quem está começando e conhecendo a Raspberry Pi. Dentro do kit, além da placa microcontroladora Raspberry Pi Pico W, você encontrará diversos componentes eletrônicos para que você possa fazer os projetos deste guia e outros projetos que você desejar.

(01) A **Raspberry Pi Pico W** é a principal protagonista do nosso kit e será usada em todas as aulas deste guia. Perceba que a Pico W que acompanha o kit já está com a **barra pinos soldados**, então você não vai precisar se preocupar com isso.

(02) Com **Módulo Amplificador de Microfone MAX4466** você conta com um microfone de eletreto e um amplificador com ganho ajustável em uma pegada muito compacta.

(03) O **Cabo Micro USB** servirá tanto para conectar a placa ao computador e permitir a gravação dos códigos, quanto fará a alimentação de energia do computador a placa.

(04) Ao longo das aulas vamos aprender como funcionam os **LEDs** e como ligá-los na protoboard. Vários projetos desse guia utilizam este componente: você vai aprender a acender LEDs de diversas maneiras!

(05) Além de LEDs convencionais teremos o **Anel de LED RGB Endereçável** composto por 12 LEDS RGB cada um com microcontrolador próprio, perfeito para fazer diversas animações e deixar os projetos incríveis!

(06) A **Protoboard 400 Pontos** será utilizada em algumas aulas do guia que necessitam de montagem de circuitos eletrônicos. Mais à frente no guia iremos explicar como são feitas as ligações na protoboard.

(07) No kit temos dois valores de **resistores**: 220Ω e 10kΩ. O valor é indicado pelas faixas de cores impressas nele. Você vai entender melhor a diferença entre esses dois valores, assim como a função deles nos circuitos nas aulas seguintes.

(08) Os **jumpers** serão usados para fazer as conexões do circuito montado na protoboard com o Raspberry Pi Pico W e os componentes.

(09) O **Sensor DHT11** é o sensor de umidade e temperatura mais utilizado em projetos com Arduino e Raspberry Pi. Este sensor permite fazer leituras de temperaturas entre 0 ºC e 50 ºC e umidade entre 20% e 90%.

(10) O **Sensor de Luminosidade LDR** é um componente cuja resistência varia de acordo com a intensidade da luz. Quanto mais luz incidir sobre o componente, menor a resistência. E pode ser usado para projetos de automação residencial.

(11) O **Buzzer Ativo 5V** é um componente indicado para você que precisa adicionar efeitos sonoros em projetos eletrônicos como alarmes, sistemas de sinalização, jogos, brinquedos etc.

(12) **Módulo Sensor de Chave TTP224 para 4 Canais** é formado por quatro botões touch. Seu funcionamento é bem simples: quando você toca em uma das teclas indicadas, a saída do sensor referente à tecla vai para nível lógico alto.

(13) O **Mini sensor de movimento presença PIR** usa sensores infravermelhos para detectar a movimentação de pessoas, e seu tamanho reduzido é ideal para embutir em caixas, tomadas e projetos DIY.

(14) O **Anel de LED RGB x12 WS2812 5050 Endereçável** é composto por 12 leds RGB interligados e endereçáveis individualmente, permitindo fazer animações individuais com cada um dos 12 LEDs

(15) O **kit módulo leitor RFID** é baseado no chip MFRC522 da empresa NXP, altamente utilizado em comunicação sem contato a uma frequência de 13,56MHz. Este chip, de baixo consumo e pequeno tamanho, permite sem contato ler e escrever em cartões que seguem o padrão Mifare, muito usado no mercado.

(16) Com o Display **IPS LCD TFT** você vai adquirir cores mais vivas que um display LCD comum!

(17) A **Caixa Plástica** acomoda todos os componentes do Kit e após o fim do curso pode fazer parte da organização da sua bancada de trabalho.

(18) A peça **Luminária** foi produzida em impressão 3D, foi desenvolvida especialmente para o encaixe do Anel de LED para fazer a difusão da luz do anel e permitir que se enxergue melhor as animações e troca de cores.

(19) As peças que montam a **Televisão** foram produzidas em 3D para servirem de base para o Display LCD e deixar os seus projetos mais legais.

1.3 Instruções

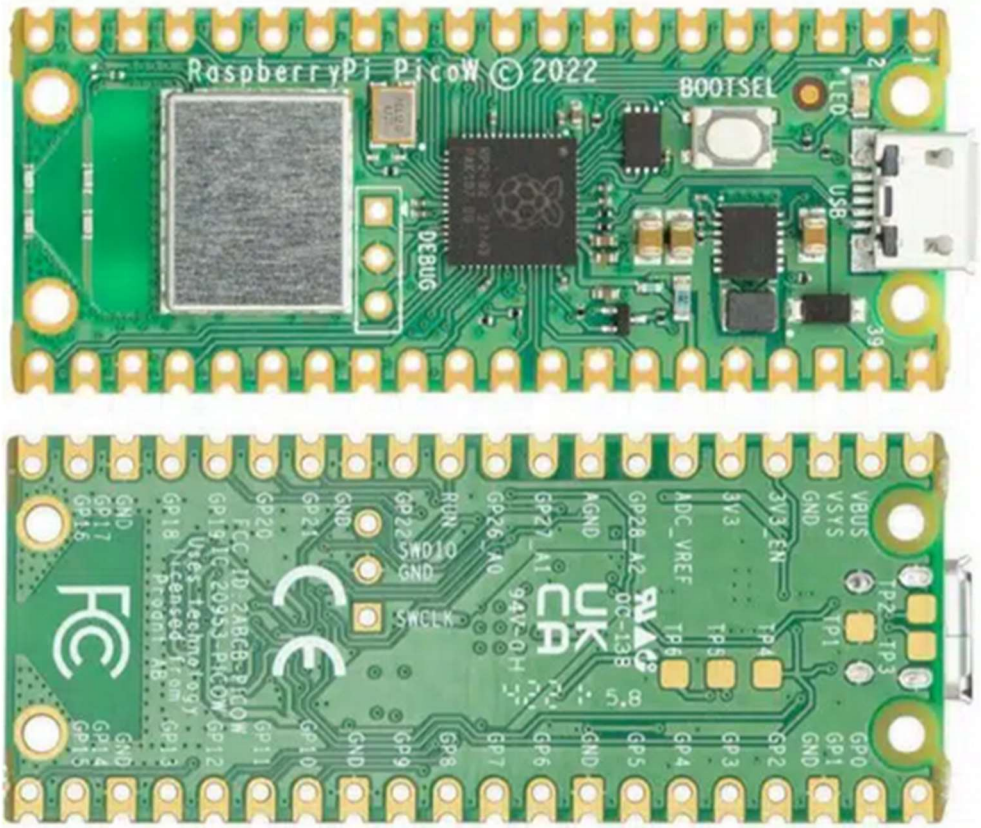
O curso completo com as aulas teóricas e todas as instruções para completar o curso estão dentro de uma plataforma de ensino, na qual você poderá acompanhar a sua evolução dentro do curso. Após a conclusão de cada aula você pode clicar no botão “Marcar como concluído” e seu progresso será registrado. Assim ao retomar as aulas você saberá onde parou.

As aulas serão compostas por textos ensinando a teoria, com imagens, esquemáticos e passo-a-passos de instruções como material de apoio. Deixando a trilha de ensino clara e objetiva para você aprender com a gente.

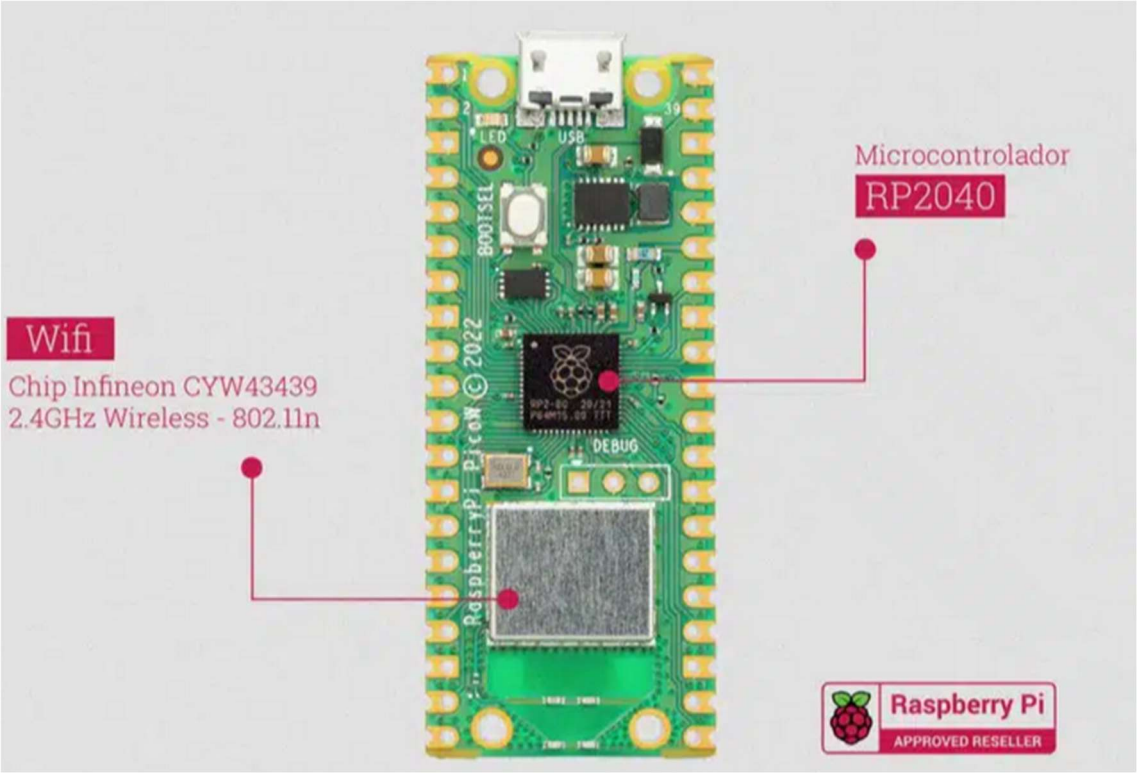
Fique atento que dentro das aulas de projetos teremos blocos de texto com a indicação de códigos de programação, os quais você será orientado dentro da aula a copiar e colar as linhas de código em sua IDE Thonny, que é a plataforma para programar a sua placa. Não se preocupe que nas próximas aulas você será guiado dentro de todo o percurso de ensino.

2. Raspberry Pi Pico W

A **Raspberry Pi Pico W** possui as mesmas dimensões e funções que sua antecessora, a Raspberry Pi Pico, mas surpreende por trazer a tecnologia wireless e Bluetooth embutida na placa, possibilitando a utilização em projetos IoT (Internet das Coisas).



No coração da placa Pico W está o chip **RP2040**, que conta com dois núcleos ARM Cortex-M0+ e clock de até 133 MHz. Também conta com o **Chip Infineon CYW43439** que possibilita a conexão da placa em redes wireless com banda de 2.4GHz e Bluetooth 5.2. Além disso, 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 Pico W possui uma **GPIO de 40 pinos**, sendo que 26 deles são multifuncionais. Com esta placa, você pode criar projetos IoT em Python e C/C++, de forma simples e bastante prática.

Além disso, a Pico W conta com **certificação da Anatel** (Agência Nacional de Telecomunicações) que assegura que o equipamento está dentro das especificações definidas pela agência, e em conformidade com a regulamentação brasileira de telecomunicações. Isso garante que você possa desenvolver e comercializar produtos utilizando esta placa.

Características:

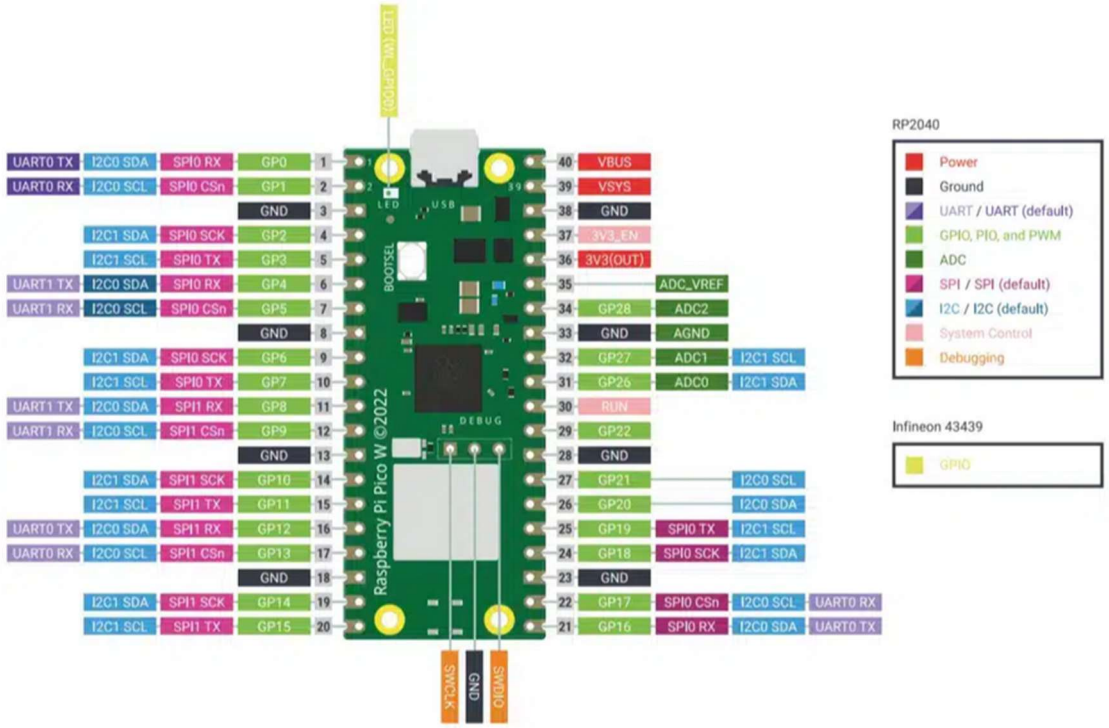
- Microcontrolador RP2040 ARM Cortex-M0+ Dual Core
- Chip Infineon CYW43439 Single Band 2,4 GHz e Bluetooth 5.2
- Clock 133 MHz
- Memória RAM: 256 KB
- Memória FLASH: 2 MB
- Desenvolvimento em Python ou C/C++
- Comunicação: 2×SPI, 2×I2C, 2×UART, 3×12-bit ADC, 16×canais PWM controláveis

- GPIO de 40 pinos (26 pinos multifuncionais)
- RTC (Real Time Counter) de alta precisão
- Sensor de temperatura on-board
- PIO: 8 pinos de máquina de estados programáveis
- Certificado de homologação Anatel: 13408-22-10629
- Dimensões: 21 mm x 51,3 mm x 3,9 mm

2.2 Pinos

A Raspberry Pi Pico W possui uma **GPIO de 40 pinos**, sendo que 26 deles são multifuncionais. Há pinos para comunicação, 3 entradas ADC e o nível lógico dos pinos é de 3,3 V.

Segue abaixo o esquemático dos pinos e sua nomenclatura:



Nome	Descrição	Função
GP0-GP28	Pinos de entrada/saída de uso geral	Atuam como entrada ou saída e não têm propósito próprio fixo.
GND	0 volts terra	Existem vários pinos GND ao redor da Pico W para facilitar a fiação do circuito.
RUN	Ativa ou desativa sua Pico	Inicie e pare a Pico W a partir de outro microcontrolador.
GPxx_ADCx	Entrada/saída de uso geral ou entrada analógica	Usado como entrada analógica e também como entrada ou saída digital – mas não ambas ao mesmo tempo.
ADC_VREF	Referência de tensão do conversor analógico-digital (ADC)	Um pino de entrada especial que define uma tensão de referência para qualquer entrada analógica.
AGND	Conversor analógico-digital (ADC) 0 volts terra	Uma conexão de aterramento especial para uso com o pino ADC_VREF.
3V3(O)	Tensão de 3,3 volts	Uma fonte de alimentação de 3,3 V, a mesma tensão em que a Pico W funciona internamente, gerada a partir da entrada VSYS.
3v3(E)	Ativa ou desativa a alimentação	Ligue ou desligue a alimentação 3V3(O), também pode desligar a Pico W.
VSYS	Tensão de 2-5 volts	Um pino conectado diretamente à fonte de alimentação interna da Pico, que não pode ser desligado sem desligar também o Pico W.
VBUS	Tensão de 5 volts	Uma fonte de energia de 5V retirada da porta micro USB da Pico é usada para alimentar hardware que precisa de mais de 3,3 V.

potência e tensão são coisas diferentes. Potência (expressa com unidade em Watts) é a relação de tensão e corrente.

3. Teoria básica de circuito e boas práticas

3.1 Circuito Eletrônico

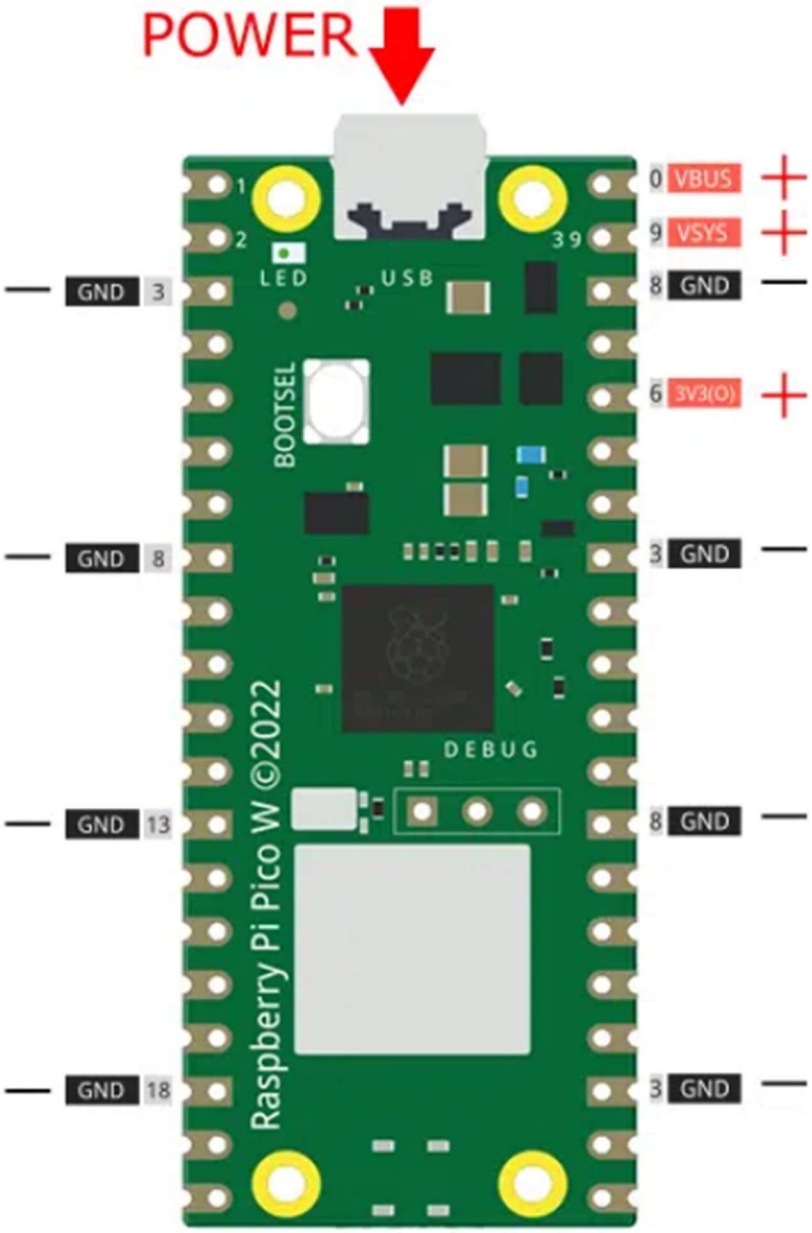
Há muitas coisas que você usa todos os dias que são alimentadas por eletricidade, como as luzes da sua casa e o computador no qual você está lendo isto.

Para usar eletricidade, você deve criar um circuito elétrico. Um circuito elétrico consiste em fios metálicos e componentes elétricos e eletrônicos.

Os circuitos requerem energia de algum lugar. Em sua casa, a maioria dos eletrodomésticos, como TVs e luminárias, são alimentadas por tomadas de parede. Entretanto muitos circuitos portáteis menores, como controles remotos e celulares são alimentados por baterias. A bateria possui dois terminais, um positivo (+) e um negativo (-).

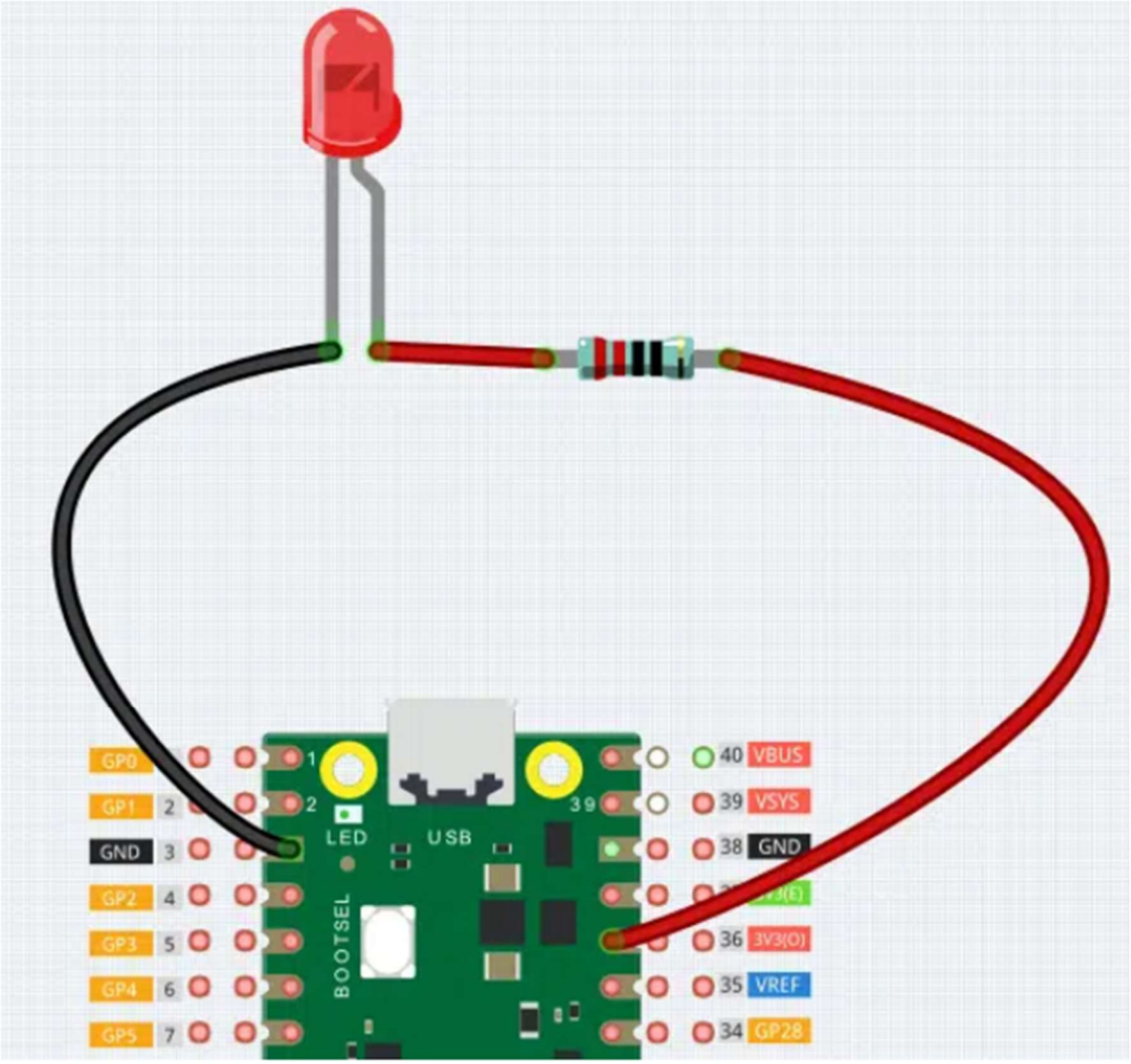
Para que a corrente flua, um caminho condutor (fio) deve conectar o terminal positivo da bateria ao terminal negativo, que é conhecido como circuito fechado (se estiver desconectado, é chamado de circuito aberto). A corrente elétrica fluirá através de aparelhos como lâmpadas para fazê-los funcionar, no caso, acender.

A Pico W funciona a partir do mesmo princípio e, portanto, possui alguns pinos de saída de energia (positivo) como os pinos 3V3, VSYS e VBUS e também alguns pinos de aterramento (negativo) que são os pinos GND. A partir do momento que você conectar a Pico W na fonte de alimentação, conectando-a ao computador pelo cabo USB. Você pode usar esses pinos como os lados positivo e negativo da fonte de alimentação dos componentes eletrônicos que você irá conectar ao circuito.



Com eletricidade, você pode criar obras com luz, som e movimento.

Abaixo temos um exemplo de circuito que acende um LED sem necessidade de programação. Conectamos o pino longo do LED ao terminal positivo e o pino curto ao terminal negativo. Nesse caso também temos que acrescentar um resistor para manter a estabilidade do circuito e prevenir danos ao LED. Com essas conexões completamos o circuito.



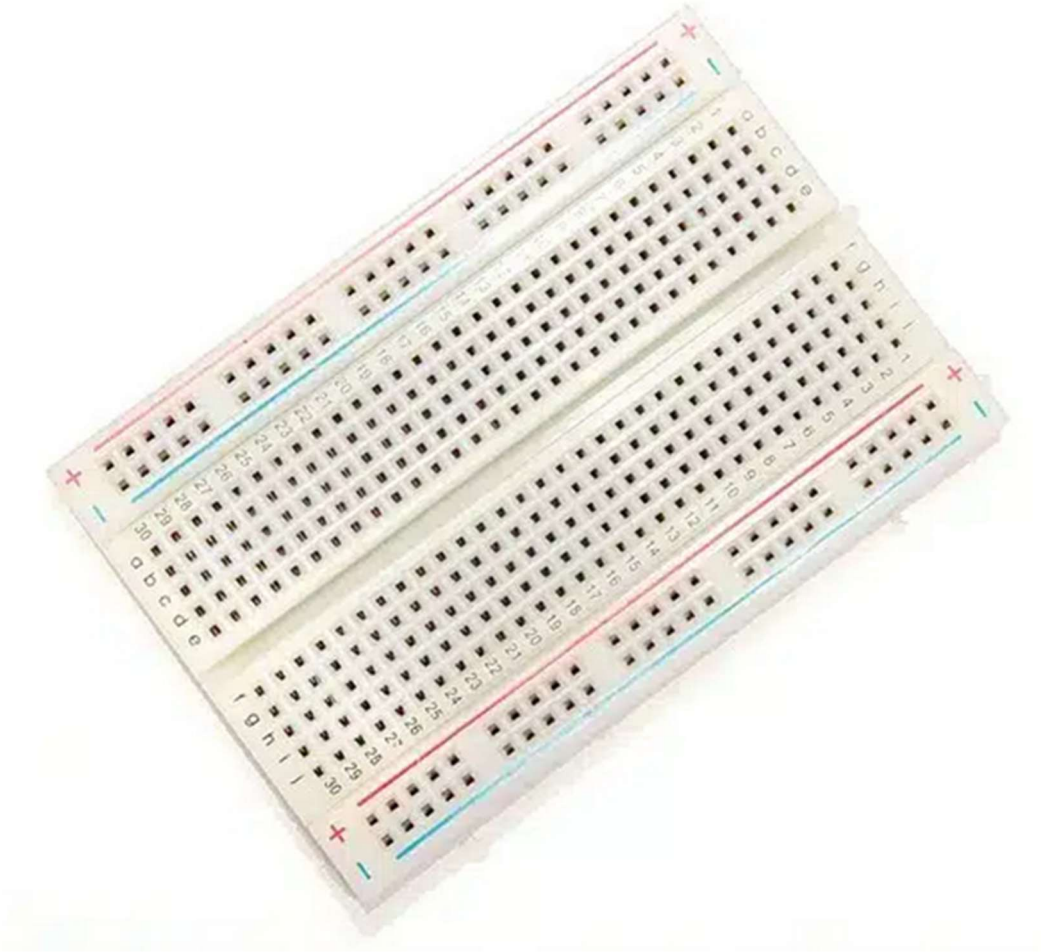
Para montar o circuito fisicamente podemos utilizar os jumpers, que são fios especialmente feitos para prototipar projetos de eletrônica e são usados para fazer as conexões entre placa e componente. Mas para auxiliar e deixar seus projetos mais organizados utilizamos a protoboard como base de montagem dos circuitos eletrônicos.

3.2 Protoboard

A protoboard é um componente utilizado em todos os projetos do nosso kit e em quase todos os projetos de eletrônica. É nela que se faz a montagem dos circuitos eletrônicos, ação também conhecida como **prototipagem**.

Uma protoboard é cheia de furos, onde você pode encaixar os pinos e terminais dos componentes eletrônicos, ou até mesmo um fio (jumper) diretamente, e tem

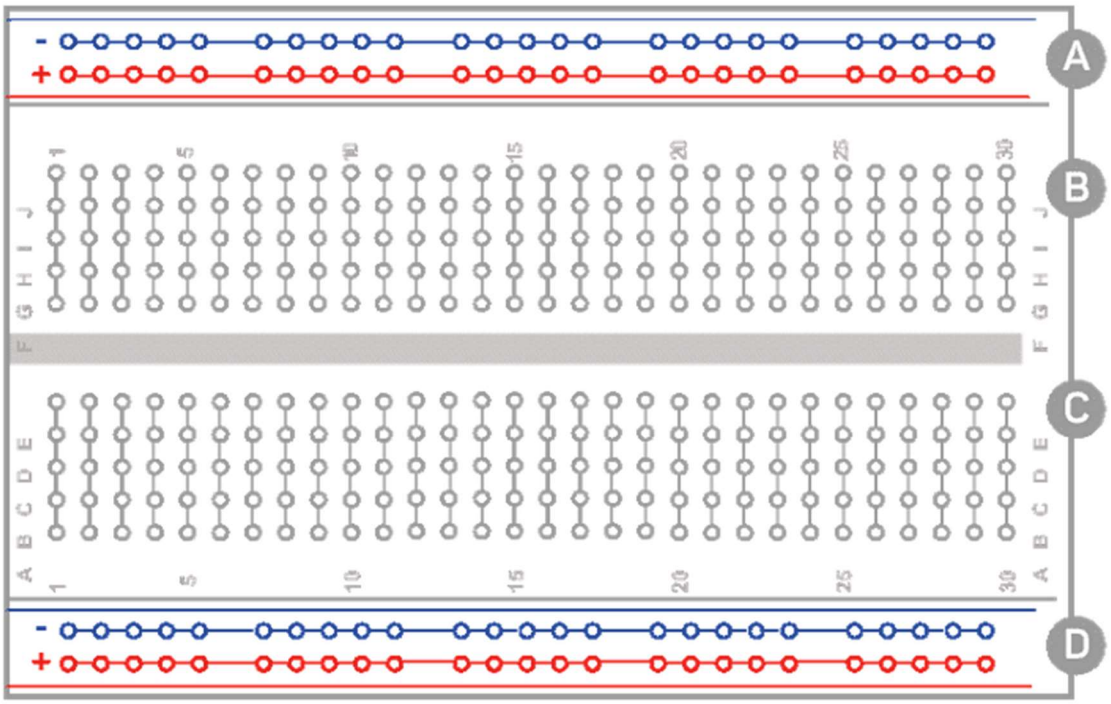
uma lógica de conexão entre os furos que, depois que você entende, a montagem dos seus projetos fica bem intuitiva.



Com a protoboard podemos experimentar a montagem de diversos tipos de combinações de componentes eletrônicos e circuitos sem a necessidade de conectá-los permanentemente. Caso haja a necessidade de trocar um componente de posição ou mesmo substituí-lo, isso pode ser feito de maneira muito rápida e fácil.

Como você pode ver na figura abaixo, na protoboard existem dois blocos de colunas B e C, que são uma série de 30 colunas (1 a 30) lado a lado nomeada de “a” a “e” e “f” a “j”. Cada coluna possui 5 furos e esses estão interligados entre si como mostrado nas linhas cinzas. Uma coluna não possui conexão interna com a coluna ao lado.

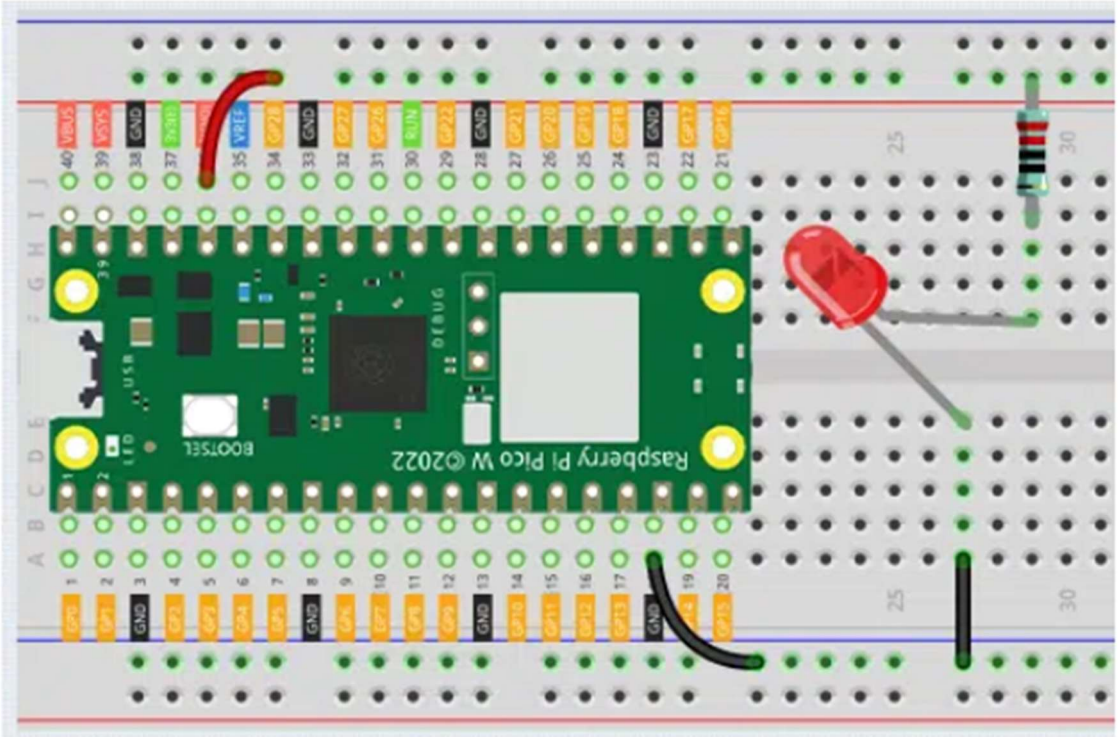
Nos blocos B e C são montados a maior parte dos circuitos. Mas os dois blocos não são interligados entre si, sendo separados por uma cavidade central.



Já nos blocos A e D, nas extremidades superior e inferior da protoboard, temos as linhas em vermelho e azul. Todos os furos de uma mesma linha estão interligados entre si, mas os furos da linha vermelha não estão conectados aos furos da linha azul. Na linha vermelha existe um sinal de positivo “+” e na azul um sinal de negativo ”-”.

É nessas linhas que ligamos a energia do circuito que vamos montar. Nos projetos, geralmente se coloca o a tensão de alimentação (3.3V no caso da Pico W) no vermelho e o GND no azul, apenas uma questão de organização.

Veja o exemplo do circuito para ligar um LED utilizando a protoboard:



Nos nossos projetos sugerimos uma montagem através de um desenho auxiliar. Mas se você mantiver as mesmas conexões e respeitar a lógica da protoboard, pode montar o projeto em qualquer lugar dela.

3.3 Cuidados

Direção do circuito

Existe uma orientação para os circuitos, e a orientação desempenha um papel significativo em certos componentes eletrônicos. Existem alguns dispositivos com polaridade, o que significa que devem ser conectados corretamente com base nos seus pólos positivo e negativo. Circuitos construídos com orientação errada não funcionarão corretamente.

No kit temos alguns componentes que possuem polaridade e precisam de atenção, são eles:

- **LEDs:** têm um lado positivo (+) e outro negativo (-). Se você inverter, o LED não acende.
- **Buzzer:** têm um lado positivo (+) e outro negativo (-).

- **Resistores:** não têm pólos, então você pode colocá-los de qualquer maneira.

Uma dica é procurar por os seguintes pontos:

- Procure por símbolos “+”, “-”, “GND”, “VCC” nos componentes.
- Veja se os pinos têm comprimentos diferentes: o maior geralmente é positivo (+).
- Consulte o manual do componente se tiver dúvidas.

Lembre-se:

- A direção dos componentes é importante para o funcionamento do circuito.
- Tenha cuidado ao conectar os componentes para evitar problemas.
- Se você não tiver certeza, consulte o manual ou um especialista.

Curtos-Circuitos

Imagine um caminho errado na cidade: o trânsito fica parado e tudo pode dar errado! Em eletrônica, os curtos-circuitos são como esse caminho: quando dois componentes que não deveriam se tocar se encostam, a corrente elétrica toma um atalho inesperado.

No seu kit, você tem resistores, LEDs e outros componentes com pinos metálicos. Se esses pinos se tocarem acidentalmente pode ocorrer o curto-circuito.

Alguns circuitos simplesmente travam e não funcionam mais. Mas o pior pode acontecer: um curto-circuito pode **queimar** seus componentes.

Como se proteger?

Para evitar esses perigos, siga estas dicas:

- **Cuidado com os Pinos:** Evite tocar os pinos dos componentes com as mãos ou ferramentas metálicas.
- **Organização é tudo:** Mantenha os componentes organizados na protoboard, evitando que se encostem.
- **Revise antes de ligar:** Antes de ligar o circuito, faça uma revisão visual para garantir que não há nenhum componente tocando outro.

Com atenção, seus projetos de eletrônica funcionarão perfeitamente!

4. O que é MicroPhyton?

Neste módulo você aprenderá sobre o tipo de programação que iremos usar nos nossos projetos, que será a linguagem MicroPython e como utilizar os softwares que gravam os códigos na placa.

4.1 Introdução ao MicroPhyton

O MicroPython é uma implementação enxuta e eficiente da linguagem de programação Python 3. Ele inclui um pequeno subconjunto da biblioteca padrão do Python e é otimizado para funcionar em microcontroladores e em ambientes com pouca memória e processamento.

Essa linguagem de programação é ideal para prototipagem, sendo assim, você pode testar e desenvolver rapidamente programas para seus projetos eletrônicos. Pois possui bibliotecas para interagir com sensores, LEDs, displays e outros componentes eletrônicos.

MicroPython é uma ótima opção para quem quer começar a programar microcontroladores sem precisar aprender uma linguagem totalmente nova. É uma linguagem amigável e poderosa, tornando o desenvolvimento de projetos eletrônicos mais acessível.

Além de implementar uma seleção de bibliotecas básicas do Python, o MicroPython inclui módulos como “máquina” para acessar hardware de baixo nível.

MicroPython se esforça para ser o mais compatível possível com Python normal (conhecido como CPython), para permitir a transferência fácil de código do desktop para um microcontrolador ou sistema embarcado, então quanto mais você aprende sobre MicroPython, mais fácil você entrará no universo em Python.

4.2 Instalando a IDE Thonny

Antes de começar a programar o Pico com MicroPython, você precisa de um ambiente de desenvolvimento integrado (IDE), aqui recomendamos Thonny.

Thonny vem com Python 3.7 integrado, apenas um instalador simples é necessário e você está pronto para aprender programação.

Observação: Como o interpretador Raspberry Pi Pico só funciona com Thonny versão 3.3.3 ou posterior, você pode pular este capítulo se o tiver; caso contrário, atualize ou instale-o.

Passo 1: Você pode baixá-lo visitando o site do Thonny . Depois de abrir a página, você verá uma caixa cinza claro no canto superior direito, clique no link que se aplica ao seu sistema operacional.

4.3 Instalando a MicroPhyton na sua Pico W

Agora vamos instalar o MicroPython na Raspberry Pi Pico W.

Você pode programar sua Pico W conectando-o a um computador via USB e, em seguida, arrastando e soltando um arquivo nele. Então reunimos um arquivo UF2 para download para permitir que você instale o MicroPython com mais facilidade.

Passo 1: Pressione e segure o botão BOOTSEL e utilizando o cabo USB, conecte a Pico W à porta USB do computador. Solte o botão BOOTSEL só depois que sua Pico W estiver conectada ao computador.

Passo 2: Ela será identificada pelo computador como um dispositivo de armazenamento em massa chamado RPI-RP2.

Passo 3: Arraste o arquivo MicroPython UF2, que você fez o download anteriormente, e solte no volume RPI-RP2. Sua Pico W será reiniciada.

Pronto! Agora você está executando o MicroPython.

Caso você queira mais informações, o livro Raspberry Pi Pico Python SDK contém instruções passo a passo para conectar a Pico W e programá-la em MicroPython usando a linha de comando e o IDE Thonny.

4.4 Upload das Bibliotecas na Pico W

Em alguns projetos, vamos precisar de bibliotecas adicionais para rodar o código. Bibliotecas são pacotes de códigos já desenvolvidos especificamente para programação e funcionamento de algum componente.

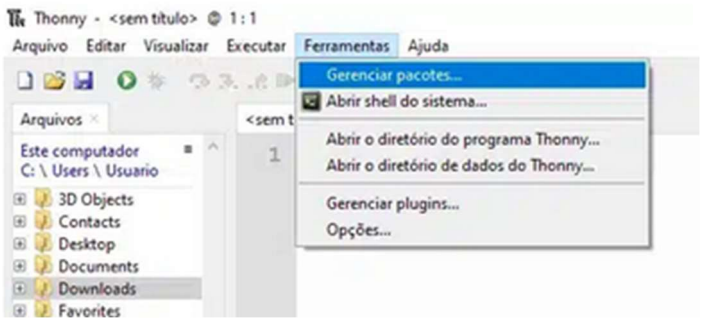
Então, aqui, vamos explicar o processo de instalação de bibliotecas pelo IDE Thonny. Assim, quando você precisar instalar alguma biblioteca nas aulas futuras, você pode voltar e consultar o passo a passo.

Há dois jeitos de instalar uma biblioteca na IDE Thonny:

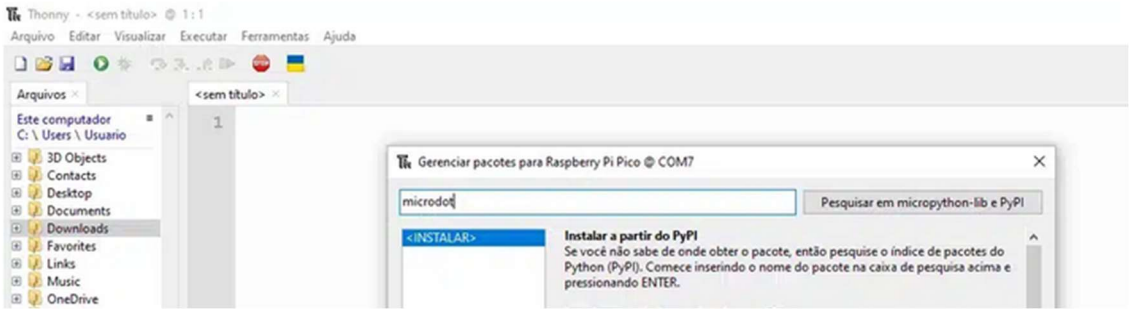
- Diretório da IDE: onde buscamos o pacote de código dentro do próprio diretório da IDE Thonny.
- Link Externo: o pacote da Biblioteca vem de um link externo e fazemos o upload do pacote para a IDE Thonny.

Instalação de biblioteca pelo diretório da IDE

Passo 1: Para isso vá em **Ferramentas > Gerenciar pacotes;**



Passo 2: Busque pelo nome da Biblioteca sugerida na aula na barra e selecione a versão escolhida.

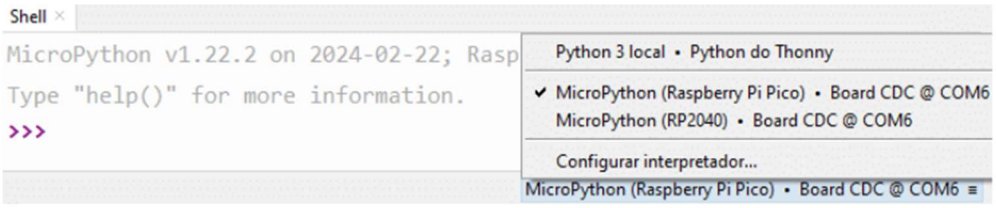


Instalação de biblioteca por link externo

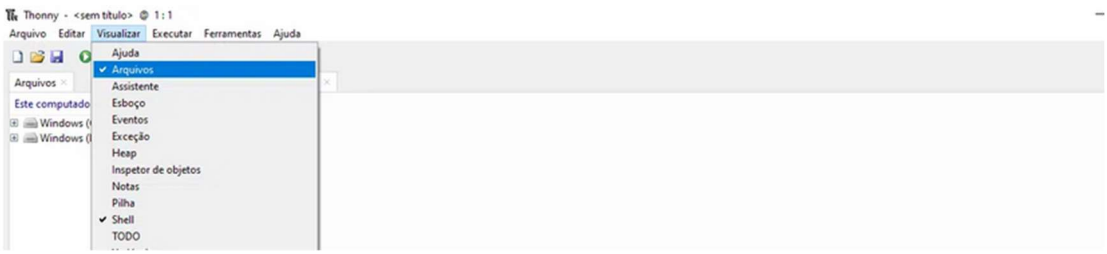
Passo 1: Baixe o código relevante no link exposto na aula do projeto em questão.

- Link biblioteca

Passo 2: Abra o Thonny IDE e conecte a Pico W ao seu computador usando o cabo micro USB e clique no interpretador “MicroPython (Raspberry Pi Pico).COMXX” no canto inferior direito.



Passo 3: Na barra de navegação superior, clique em **Visualizar** -> **Arquivos** .



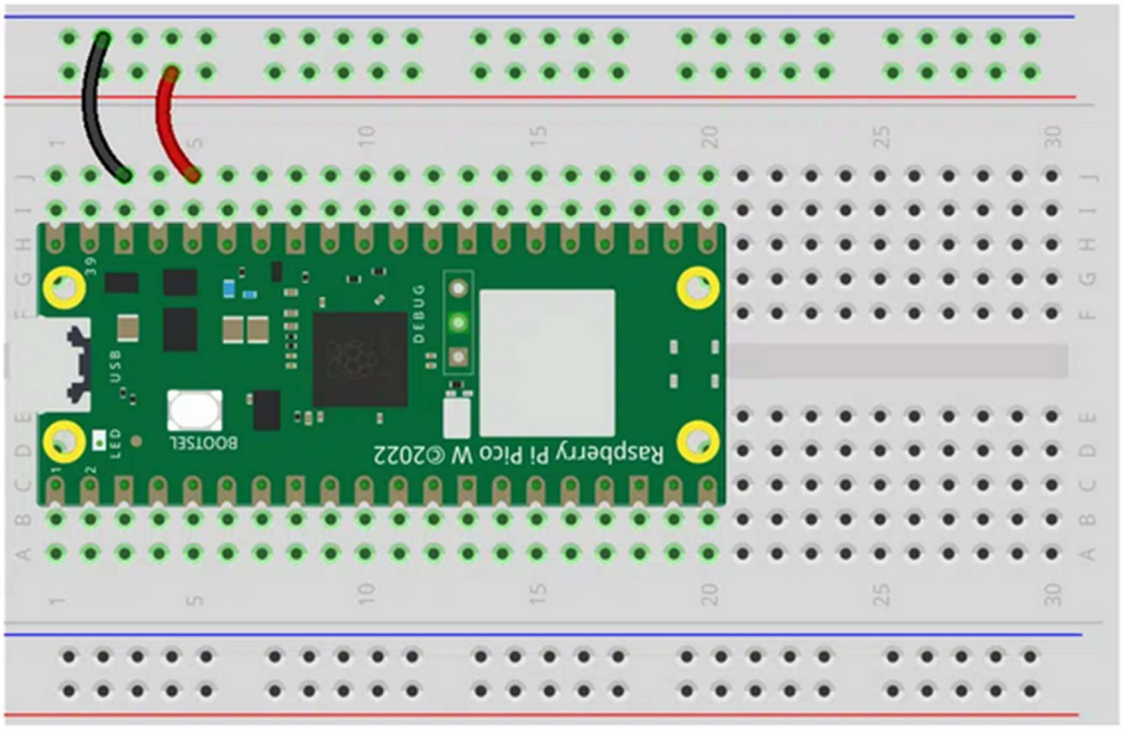
4.6 Piscar o LED da Raspberry Pi Pico W

A IDE Thonny pode salvar e executar programas MicroPython diretamente em seu Raspberry Pi Pico W.

Nesta etapa, você criará um programa MicroPython para acender e apagar o LED integrado à própria placa em um loop.

Passo 1: Abra a IDE Thonny.

Passo 2: Conecte a Pico W ao seu computador com um cabo micro USB e selecione o interpretador “MicroPython (Raspberry Pi Pico)” no canto inferior direito.



Essas conexões apresentadas acima, são as conexões padrão entre placa e protoboard.

Passo 3: Clique no painel principal do editor do Thonny. Digite o seguinte código para alternar o LED.

```
from machine import Pin

led = Pin("LED", Pin.OUT)

led.toggle()
```

Passo 4: Clique no botão Executar para executar seu código.

Caso a opção “Permitir executar programas sem nome” (que fica na aba Executar e depurar no menu localizado em Ferramentas -> Opções) esteja ativada, o Thonny perguntará se você deseja salvar o arquivo neste computador ou no dispositivo MicroPython . Escolha o dispositivo MicroPython.

Digite blink.py como o nome do arquivo.

Dica: Você precisa inserir a .py extensão do arquivo para que Thonny reconheça o arquivo como um arquivo Python.

Thonny pode salvar seu programa na Raspberry Pi Pico e executá-lo.

Você deverá ver o LED integrado alternar entre ligado e desligado sempre que clicar no botão **Executar**.

Passo 5: Você pode usar o módulo **Timer** para definir um temporizador que execute uma função em intervalos regulares.

Atualize seu código para que fique assim:

```
from machine import Pin, Timer
led = Pin("LED", Pin.OUT)
timer = Timer()
def blink(timer):
  led.toggle()
timer.init(freq=2.5,mode=Timer.PERIODIC, callback=blink)
```

Passo 6: Clique em **Executar** e seu programa acenderá e apagará o LED até você clicar no botão **Parar**.

Passo 7: Salve seu projeto.

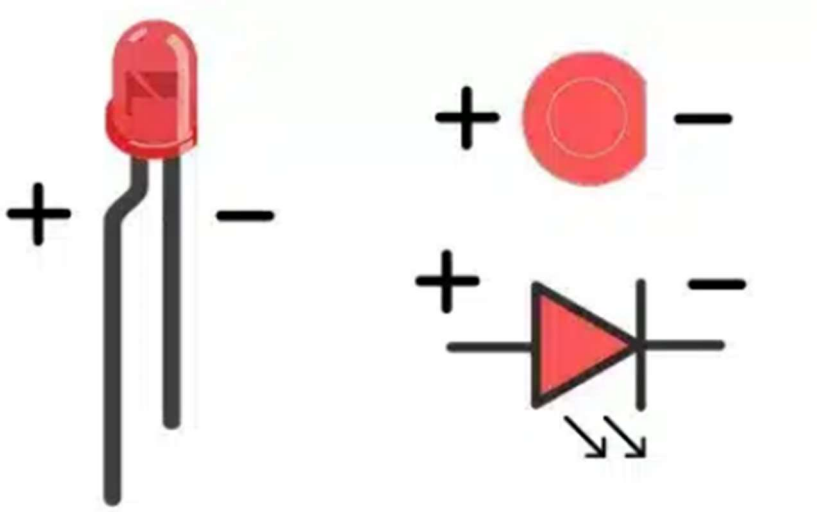
5. LED

Neste módulo vamos apresentar o LED e seu conceito de funcionamento. E ainda apresentar um modelo mais complexo chamado de LED endereçável.

5.1 O que é um LED?

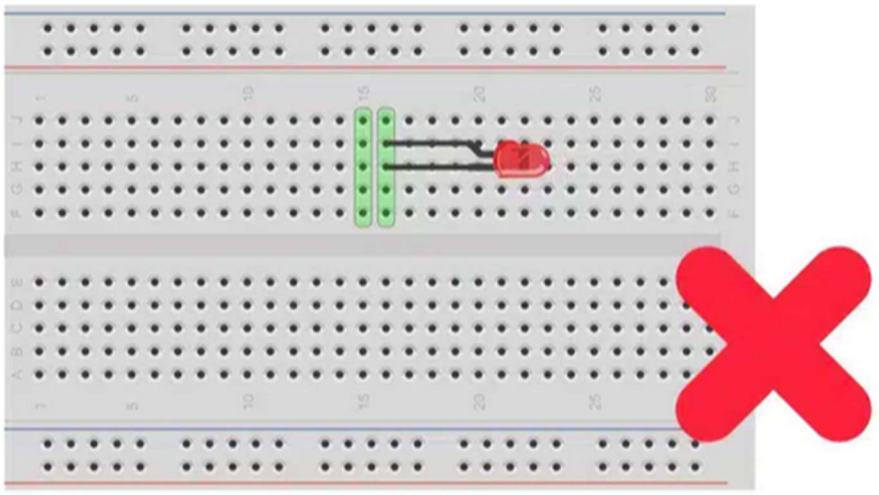
LED (do inglês, Light Emitting Diode) é um diodo emissor de luz, basicamente é uma lâmpada que consome pouca energia. Por seu baixo consumo, está se tornando cada vez mais comum o seu uso em casas e iluminação pública.

O LED, assim como alguns componentes, possui um lado positivo “+” e um lado negativo “-“, essa característica também é conhecida como polaridade. Para esses componentes, se ligar de modo invertido, ele não irá funcionar e pode até acabar danificando o componente. O LED possui duas “perninhas” que chamamos de terminais. A haste maior do LED é o lado positivo e a menor é o lado negativo. Você pode ver também uma indicação pelo próprio formato da parte de translúcida do LED que possui um lado mais achatado, que é o lado negativo enquanto o lado arredondado é o positivo.

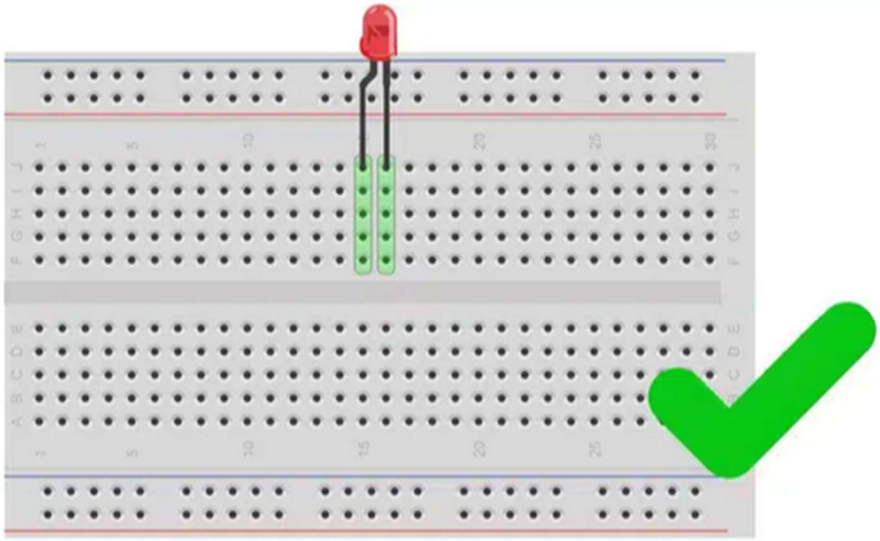


Pinagem LED

A montagem correta de um LED em uma protoboard é feita como mostrado na figura abaixo. Os dois terminais do LED não podem ficar na mesma coluna da protoboard, lembrando que os furos das colunas são conectados entre si na vertical. Esse mesmo princípio de montagem se aplica também a outros componentes do kit.



Ligação incorreta do LED na protoboardLigação correta do LED na protoboard.



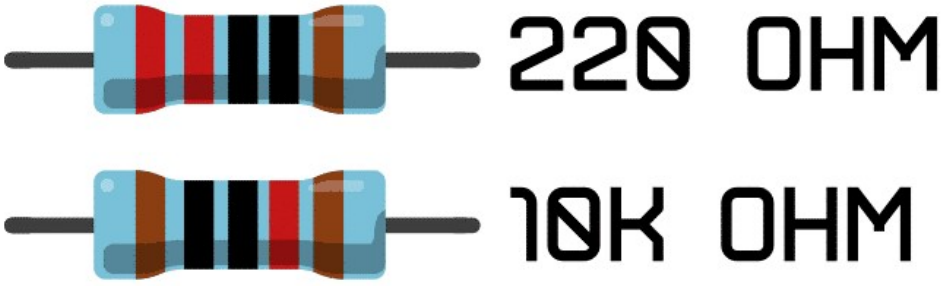
Ligação correta do LED na protoboard

5.2 O que é um resistor?

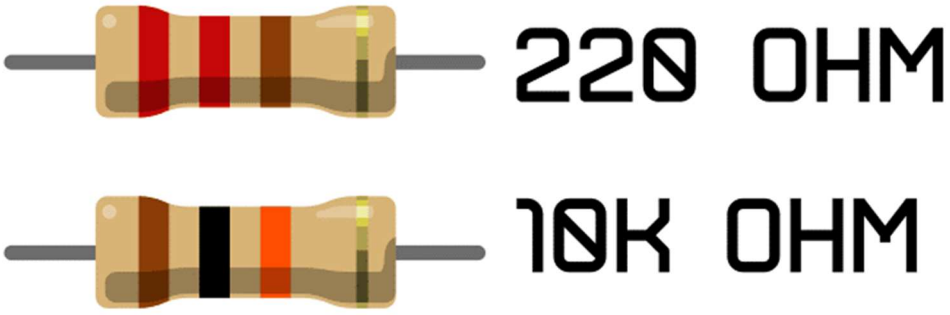
Resistores são componentes eletrônicos passivos que têm a função principal de limitar ou controlar o fluxo de corrente elétrica em um circuito. Eles são projetados para ter uma resistência elétrica específica, medida em ohms (Ω), e essa resistência determina o quão difícil é para a corrente elétrica fluir através deles.

Os resistores têm diferentes valores de resistência, quanto mais alta a resistência, mais ele irá limitar a corrente que passa por ele. O valor do resistor é fixo e é indicado pelas faixas de cores pintadas nele.

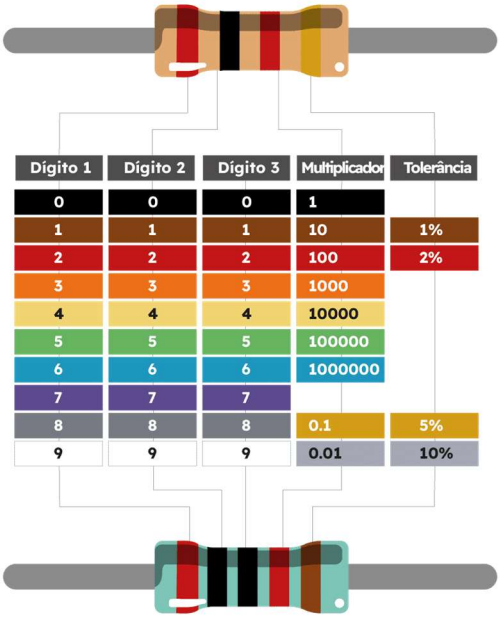
No kit temos dois tipos: 220 ohm e 10K ohm.



Atenção! O kit pode conter resistores de coloração ou faixa de cores diferentes da ilustração acima, mas não se preocupe, basta olhar o valor escrito na etiqueta do pacote de resistores. Caso venham na cor amarela e quatro faixas eles são resistores equivalentes aos azuis, só utilizam uma nomenclatura diferente, portanto identifique-os de acordo com a ilustração abaixo:



Cada valor de resistor tem o seu código de cores, que você pode ver na tabela abaixo como descobrir o valor do seu resistor pelas barrinhas coloridas dele:



Como o resistor não possui polaridade como os LEDs, portanto você pode ligar um resistor em qualquer sentido, tanto faz o lado que é conectado seus terminais. Pois ele não tem lado positivo ou negativo.

5.3 Pisca Pisca

Assim como imprimir “Olá, mundo!” é o primeiro passo para aprender a programar com uma IDE. Usar um programa para acionar um LED é a introdução tradicional ao aprendizado de programação e eletrônica.

Agora que conhecemos o funcionamento do resistor e do LED, em nosso primeiro exemplo prático iremos fazer uma luz piscar. Parece um exercício simples demais, mas isso exemplifica a utilização da placa Raspberry Pi Pico W para controle de dispositivos externos. Os conceitos aprendidos neste exemplo servem para acionamento de outros dispositivos como ventilador, lâmpadas, motores etc.

Materiais Necessários

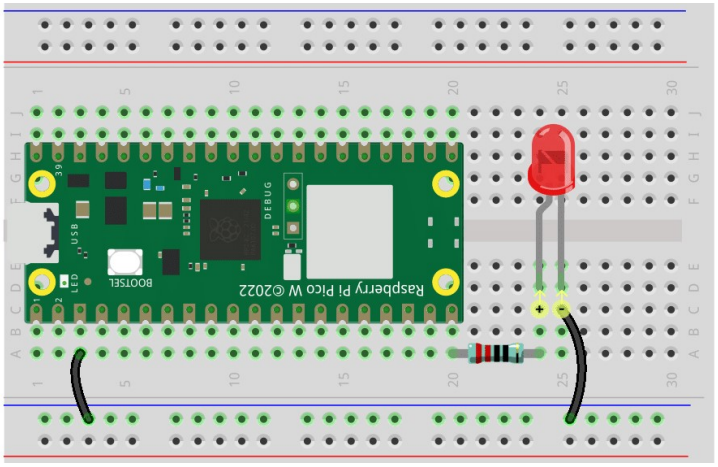
Em todos os exercícios e projetos teremos uma seção que mostra os componentes que iremos utilizar. Se você tiver alguma dúvida sobre qual é o componente, você pode voltar na lista de materiais:

- Placa Raspberry Pi Pico W
- Cabo USB

- Protoboard 400 pontos
- Jumper macho-macho
- LED Vermelho 5mm
- Resistor 220 ohm

Circuito

No esquema da montagem do circuito, as linhas coloridas são a representação gráfica dos jumpers. Utilize-os para ligar os componentes entre si conforme a ilustração abaixo:



Nesse exemplo, após a programação da placa, o LED acende quando a fonte de energia (que nesse caso é o pino de saída da Pico W) é ligada e ao desligar a fonte de energia, o LED apagará. O resistor é colocado para reduzir a corrente que passa pelo circuito inteiro.

Para construir o circuito, vamos seguir o sentido da corrente!

- O LED é alimentado pelo pino GP15 da placa Pico W, e o circuito começa aqui.
- Para proteger o LED, a corrente deve passar por um resistor de 220 ohms. Um terminal do resistor deve ser inserido na mesma linha do pino Pico W GP15, e o outro terminal deve ser inserido na linha livre da protoboard.

Observação: Os anéis coloridos do resistor de 220 ohms são vermelho, vermelho, preto, preto e marrom.

- Se você olhar o LED, verá que um de seus terminais é mais longo que o outro. Conecte o terminal mais longo à mesma fileira do resistor e o fio mais curto à mesma fileira no espaço intermediário da protoboard.

Observação: O terminal do LED mais longo é o ânodo, que representa o lado positivo do circuito; já o terminal curto é o cátodo, que representa o lado negativo.

O ânodo precisa ser conectado ao pino GPIO através de um resistor; o cátodo precisa ser conectado ao pino GND.

- Usando um fio jumper macho-macho, conecte o pino curto do LED ao barramento de alimentação negativo da protoboard.
- Conecte o pino GND da Pico W ao barramento de alimentação negativo usando um jumper.

Não é necessário conectar os componentes exatamente nos mesmos furos como indicado acima, basta apenas que os terminais de cada componente não estejam na mesma coluna.

Programa

Primeiramente vamos explicar o código em partes e logo mais abaixo você verá o programa completo. Ao fim da explicação você pode copiar o código completo e colar na sua IDE Thonny.

A biblioteca que será usada neste projeto é a machine, que será necessária para usar as portas GPIO. Essa biblioteca já faz parte do pacote de bibliotecas básicas do MicroPhyton que instalamos na IDE Thonny, portanto ao escrever o nome dela diretamente no código, ela já estará funcionando.

```
1import machine
```

A biblioteca contém todas as instruções necessárias para a comunicação entre MicroPython e a Pico W. Na ausência desta linha de código, não poderemos controlar nenhum GPIO.

A próxima que temos que prestar atenção é esta linha:

```
1led = machine.Pin(15, machine.Pin.OUT)
```

O objeto *led* é definido aqui. Tecnicamente, pode ser qualquer nome, como x, y, banana, ou qualquer personagem. Para garantir que o programa seja fácil de ler, é melhor usar um nome que descreva a finalidade.

Na segunda parte desta linha (a parte após o sinal de igual), chamamos a função Pin, encontrada na biblioteca *machine*. É usada para informar aos pinos GPIO da Pico o que fazer.

A função Pin possui dois parâmetros:

- O primeiro **(15)** representa o pino a ser configurado;
- O segundo parâmetro **(machine.Pin.OUT)** especifica que o pino deve ser de saída em vez de entrada.

O código acima "configurou" o pino, mas não acenderá o LED. Para fazer isso, também precisamos "usar" o pino, ou seja, dizer qual função ele fará.

```
1led.value(1)
```

O pino GP15 foi configurado anteriormente e denominado led. A função desta instrução é definir o valor de led como 1 para acender o LED.

Resumindo, para usar o GPIO, estas etapas são necessárias:

- **Importar biblioteca da máquina:** isso é necessário e só é executado uma vez.
- **Definir GPIO:** Antes de usar, cada pino deve ser definido.
- **Uso:** Altere o estado de funcionamento do pino atribuindo um valor a ele.

Se seguirmos as etapas acima para escrever um exemplo, você obterá um código como este:

```
import machine

led = machine.Pin(15, machine.Pin.OUT)

led.value(1)
```

Execute-o e você poderá acender o LED.

A seguir, tentamos adicionar a declaração oposta, ou seja, LED desligado:

```
import machine

led = machine.Pin(15, machine.Pin.OUT)

led.value(1)

led.value(0)
```

Com base na linha de código, este programa acenderá primeiro o LED e depois o apagará. Mas ao usá-lo, você descobrirá que não é esse o caso, pois não vemos o LED aceso. Isto se deve à velocidade de execução muito rápida entre as duas linhas, muito mais rápida do que o olho humano pode reagir. Quando o LED acende, não percebemos a luz instantaneamente. Isso pode ser corrigido desacelerando o programa.

A segunda linha do programa deve conter a seguinte instrução:

```
import utime
```

Da mesma forma que machine, a biblioteca utime é importada aqui, que trata de todas as coisas relacionadas ao tempo. Os atrasos que precisamos usar estão incluídos nisso. Adicione uma instrução de atraso entre led.value(1)e led.value(0) deixe-os separados por 2 segundos.

```
1utime.sleep(2)
```

É assim que o código deve ficar agora. Veremos que o LED acende primeiro e depois apaga quando o executamos:

```
import machine
import utime
led = machine.Pin(15, machine.Pin.OUT)
led.value(1)
utime.sleep(2)
led.value(0)
```

Finalmente, devemos fazer o LED piscar. Vamos criar um laço para repetir infinitamente a parte que queremos do programa com o laço while. Coloque as linhas que você quer que repitam após adicionar a linha while True: Todas as linhas que serão repetidas devem ser identadas (espaçadas com tab). Esse espaçamento determina todas as linhas de código que serão englobadas pelo laço de repetição. Abaixo você terá o código completo para copiar e colar na IDE:

```
import machine
import utime

led = machine.Pin(15, machine.Pin.OUT)
while True:
    led.value(1)
    utime.sleep(2)
    led.value(0)
    utime.sleep(2)
```

5.4 LED oscilante

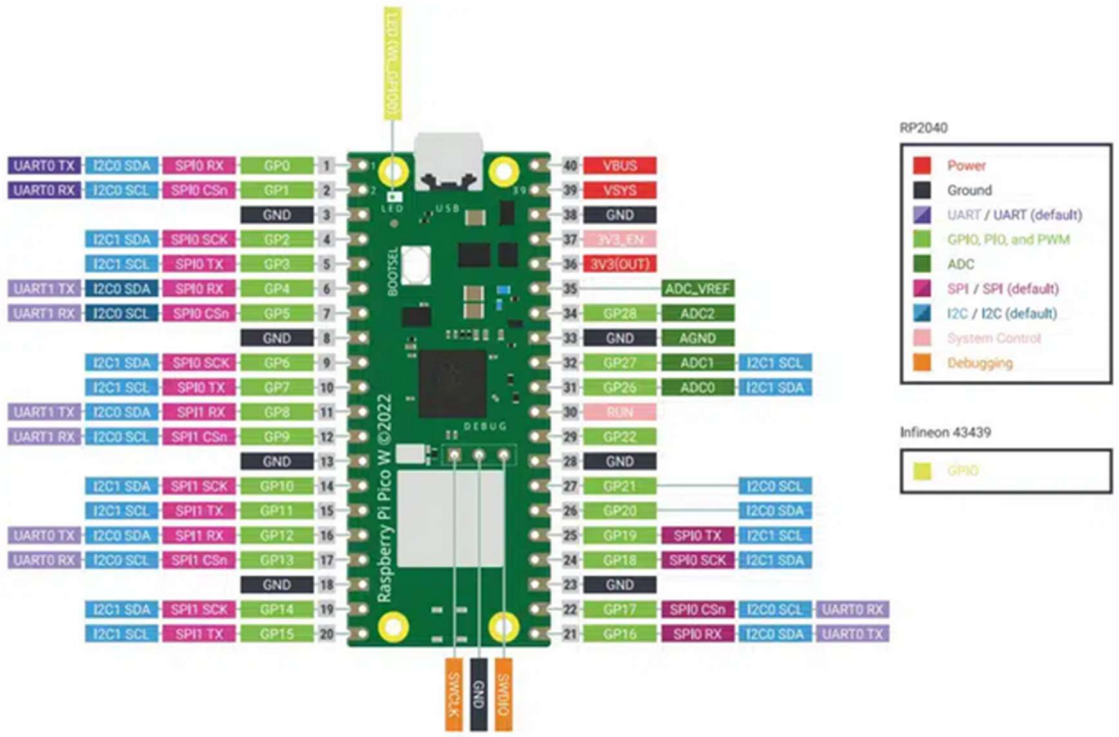
Até agora, usamos apenas dois sinais de saída: nível alto e nível baixo (também chamados de ON e OFF), que é chamado de saída digital. No entanto, na utilização real, muitos dispositivos não se limitam a LIGAR/DESLIGAR para funcionar, por exemplo, ajustando a velocidade do motor, ajustando o brilho da

lâmpada de mesa, e assim por diante. Para atingir esse objetivo a modulação por largura de pulso (PWM) surgiu como uma solução viável para problemas tão complexos.

Um pulso é uma saída digital que contém um nível alto e um nível baixo. A largura de pulso desses pinos pode ser ajustada alterando a velocidade ON/OFF.

Quando estivermos em um curto período de tempo (como 20ms, que é o período de retenção visual da maioria das pessoas), deixar o LED acender, desligar e ligar novamente, não veremos que ele foi desligado, mas sim o brilho da luz será ligeiramente mais fraco. Durante este período, quanto mais tempo o LED estiver aceso, mais brilhante ele se tornará. Em outras palavras, no ciclo, quanto mais amplo o pulso, maior será a “intensidade do sinal elétrico” emitida pelo microcontrolador. É assim que o PWM controla o brilho do LED (ou a velocidade do motor).

Há alguns pontos a serem observados quando o Pico W usa PWM. Vamos dar uma olhada nesta imagem.



A Pico W suporta PWM em cada pino GPIO, mas na verdade existem 16 saídas PWM independentes (em vez de 30), distribuídas entre GP0 a GP15 à esquerda, e a saída PWM do GPIO direito é idêntica à esquerda.

É importante evitar configurar o mesmo canal PWM para finalidades diferentes durante a programação. Por exemplo, GP0 e GP16 são ambos PWM_0A.

Vamos tentar obter o efeito de LED desbotado depois de compreender esse conhecimento.

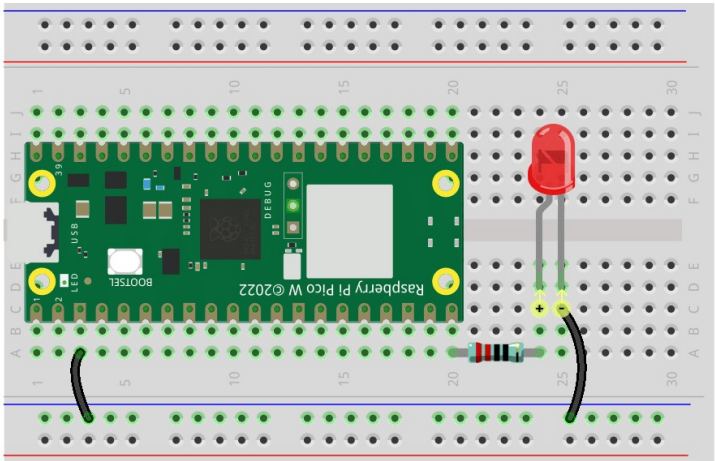
Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- LED Vermelho 5mm
- Resistor 220 ohm

Circuito

Este projeto possui o mesmo circuito do primeiro projeto 5.3 Pisca-Pisca, mas o tipo de sinal é diferente. O primeiro projeto é emitir níveis altos e baixos digitais (0 e 1) diretamente do GP15 para fazer os LEDs acenderem ou desligarem. Este projeto é para emitir o sinal PWM do GP15 para controlar o brilho do LED.



Programa

Vamos utilizar as mesmas bibliotecas do projeto anterior do 5.3 Pisca Pisca, que faz o controle do led. Aqui, alteramos o brilho do LED alterando o ciclo de trabalho da saída PWM do GP15.

O brilho do LED em PWM é controlado pelo controle da largura do pulso, que é a quantidade de tempo que o LED fica aceso a cada ciclo. Com uma frequência de temporizador de 100 Hz, cada ciclo leva 0,01 segundo, ou 10 ms.

Para obter o efeito de desbotamento mostrado no início deste tutorial, queremos definir a extensão do pulso para um valor pequeno, depois aumentar lentamente a extensão do pulso para iluminar o LED e recomeçar quando atingirmos o brilho máximo.

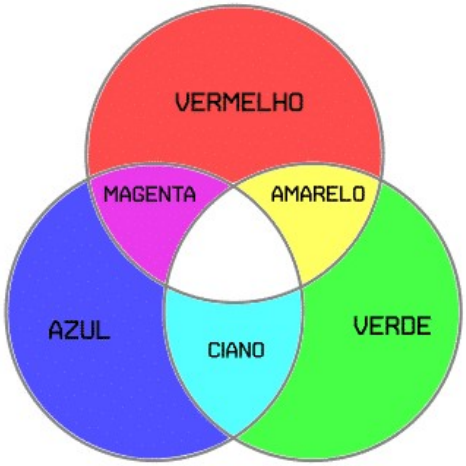
Vamos dar uma olhada no código completo abaixo:

```
import machine

import utime

led = machine.PWM(machine.Pin(15))

led.freq(1000)

for brilho in range(0,65535,50):

  led.duty_u16(brilho)

  utime.sleep_ms(10)

led.duty_u16(0)
```

- A linha led = machine.PWM(machine.Pin(15)) define o pino GP15 como saída PWM.
- A linha led.freq(1000) é usada para definir a frequência PWM, aqui ela é configurada para 1000Hz, o que significa que 1ms (1/1000) é um ciclo.
- a linha for brightness in range(0,65535,50): itera a variável brightness
- A linha led.duty_u16() é usada para definir o ciclo de trabalho, que é um número inteiro de 16 bits (2 ^ 16 = 65536). Um 0 indica ciclo de trabalho de 0%, o que significa que cada ciclo tem 0% de tempo para gerar um nível alto, ou seja, o sinal fica desligado durante todo o ciclo. O valor 65535 indica um ciclo de trabalho de 100%, o que significa que o sinal está ativado durante todo o ciclo, e o resultado é '1'. Quando for 32768 (metade do valor) , ele ficará em nível alto durante metade do ciclo, então o LED terá metade do brilho quando estiver totalmente ligado.

5.5 O que é um LED RGB Endereçável?

Um alguns LEDs são capazes de mostrar várias cores, chamamos eles de LED RGB. Ele tem o mesmo funcionamento básico de um LED comum, porém possui três cores no mesmo componente, como se fossem três LEDs juntos: Red(R), Green(G) e Blue(B). Através da mistura das 3 cores podendo atingir o espectro completo de cores luz. Conforme podemos ver no esquema abaixo:



Com base neste método, podemos usar as três cores primárias para misturar a luz visível de qualquer cor de acordo com diferentes valores de adição de cada cor. Por exemplo, o laranja pode ser produzido com mais vermelho e menos verde.

Dentro do nosso kit de componentes você vai encontrar o Anel de LED RGB, composto por 12 leds RGB interligados e endereçáveis individualmente, já que cada led possui um driver embutido.

Mas já entendemos o que é ser RGB, mas o que significa ser um LED endereçável?

Os LEDs endereçáveis são uma nova geração de LEDs, pois incluem um chip controlador, no caso dos LEDs do anel possuem o controlador **WS2812**, o que permite acessar vários LEDs com um único pino digital, atribuindo um endereço a cada LED e fornecendo uma comunicação por fio. Mas, diferentemente dos LEDs simples, esses tipos de LEDs não acendem apenas pela aplicação de tensão, eles também exigem um microcontrolador, como a placa Raspberry Pi Pico W, por exemplo.

O controle endereçável permite uma lógica interna para variar a cor de cada LED RGB (vermelho / verde / azul) individualmente (ao contrário das fitas de LEDs RGB convencionais nas quais todos os LEDs mudam de cor simultaneamente).

5.6 Acionando um Anel LED RGB

Nesta aula vamos aprender a programar o Anel de LED RGB composto por 12 LEDs. Você aprenderá o conceito de destinar uma informação específica para cada LED individualmente.

O projeto desta aula será uma animação de mudança de cores entre os LEDs do anel.

Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- Anel de LED RGB

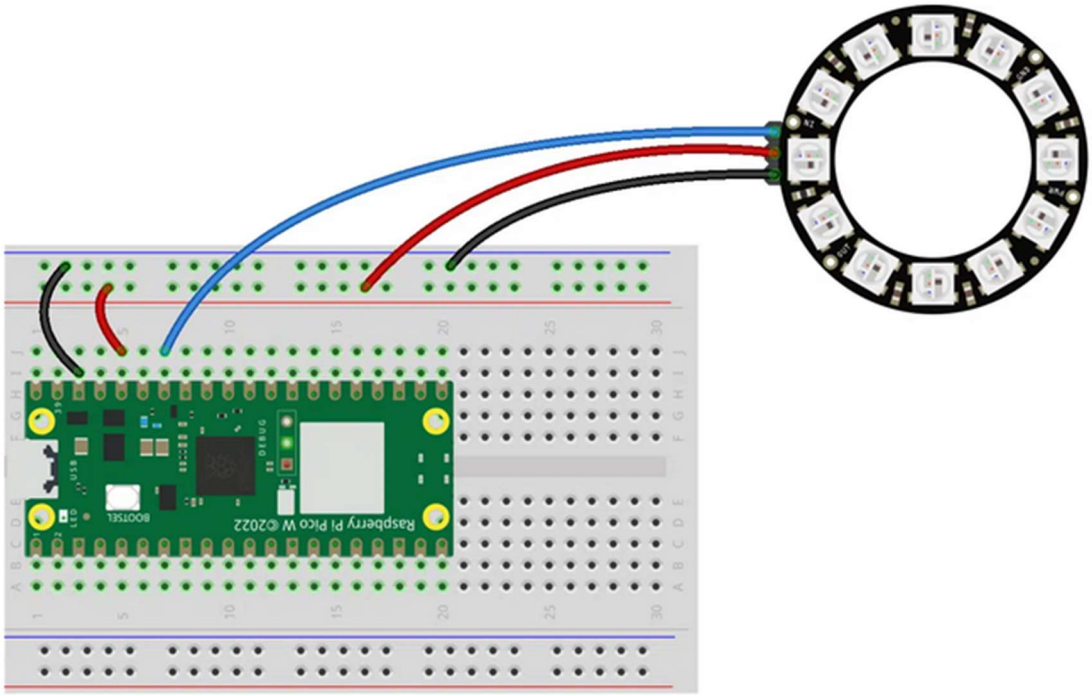
Circuito

Como padrão para a placa Raspberry Pi Pico W a energia que alimentará o circuito virá do pino 3V3(OUT).

O Anel de LED é alimentado por seu terminal 5V, que tem o fio vermelho soldado, e que deve ser ligado à linha positiva da protoboard.

Já o terminal GND do Anel de LED, que tem o fio preto, deve ser ligado a linha negativa da protoboard que estará conectada ao pino GND da placa Pi Pico.

Os dados serão transmitidos pela terminal DI do Anel e deve ser ligado a porta GP28 da placa. Este terminal do Anel de LED pode ter diferentes cores de fios soldados, mas nunca será nas cores vermelho ou preto.



Programa

Utilizaremos duas bibliotecas neste projeto, a machine, que utilizamos nos projetos anteriores e a ws2812. A biblioteca ws2812 é responsável por se comunicar com o controlador WS2812 que está embutido em cada LED. Essa biblioteca não faz parte do pacote do MicroPhyton que instalamos na IDE Thonny, portanto precisamos fazer a instalação da mesma no nosso ambiente de projeto.

Para isso vamos seguir os passos da aula 4.4 Como fazer upload de bibliotecas na Pico W utilizando o **arquivo** ws2812.py

Após a instalação da biblioteca partimos para a formação do código. Iniciamos o código mencionando as duas bibliotecas que usaremos.

```
# Codigo 5.6 - Acionando um Anel LED RGB

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao


import machine

import utime

import urandom


# Importacao das bibliotecas customizadas

from ws2812 import WS2812


# Declaracao do Anel de LEDs

ws = WS2812(machine.Pin(28),12)


# Laco de execucao

while True:

  # Itera no Anel de LEDs passando o valor(cor) do LED i para o LED i-1

  for i in range(11,0,-1):

    ws[i] = ws[i-1]

  # Define um valor de cor aleatorio para o LED 0

  ws[0] = int(urandom.uniform(0, 0xFFFFFF))



  # Envia a informacao para o Anel de LEDs

  ws.send()
```

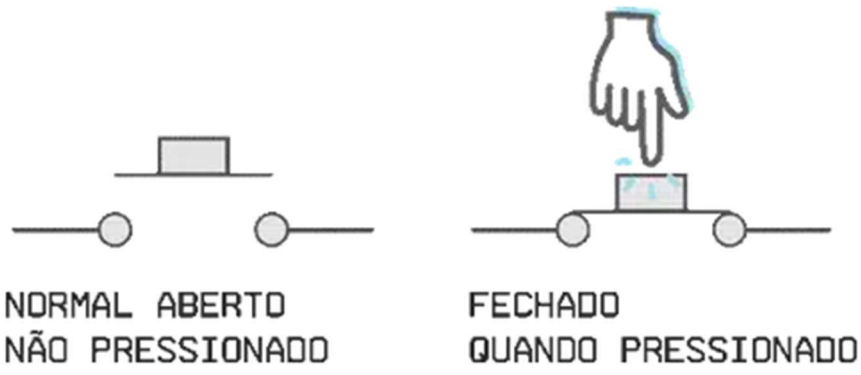
```
utime.sleep_ms(80)
```

6. Botão

Neste módulo vamos apresentar o conceito de botão e seu funcionamento. E ainda apresentar o modelo de componente que você tem no kit chamado Módulo Sensor de Chave que usaremos no lugar do push-button comum.

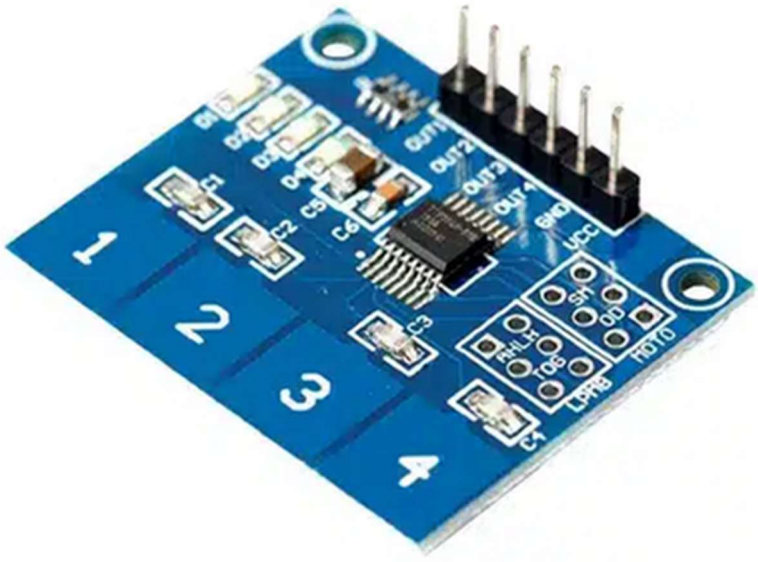
6.1 O que é o Módulo Sensor de Chave TTP224?

Praticamente todos os equipamentos eletrônicos terão pelo menos um botão para ligar. O botão é usado para permitir interação do usuário com a máquina, seja para a operação direta ou para realizar ajustes.
Os botões ou chaves como são chamados em eletrônica, funcionam como um contato que abre e fecha, sendo assim, uma chave possui dois valores, 0 ou 1, aberto ou fechado. Conectando uma chave a uma porta da placa Raspberry Pi Pico W podemos ler o valor 0 ou 1 da chave e assim tomar uma ação, como por exemplo acionar um LED.



O botão, quando pressionado, faz contato entre um lado e outro dele. Quando esse contato é fechado, essa corrente elétrica “entra” na placa e ela percebe que o botão foi pressionado. Ao escrevermos o programa, decidimos o que fazer com essa informação.

No Kit Maker Raspberry Pi Pico W o componente que usaremos como botões é o **Módulo Sensor de Chave TTP224**. Esse módulo é composto por quatro chaves, mas ao invés de botões mecânicos (botão que faz “click”), possui uma área de sensor capacitivo que detecta toques, ou seja, 4 botões touch.



Seu funcionamento é bem simples: quando você toca na região indicada, a saída do sensor vai para nível lógico alto. Sem tocar o sensor, não há atividade na saída. Assim, o componente funciona como um sensor touch com princípio semelhante aos presentes em telas de smartphones e tablets.

6.2 Ligando uma luminária com botão

Neste projeto iremos acender o Anel de LED utilizando botões. Como temos o Módulo Sensor de Chave de quatro canais, não iremos apenas usar a função liga e desliga. Mas determinar uma intensidade de luz para três dos botões e o quarto botão irá desligar nossa luminária.

Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB

- Protoboard 400 pontos
- Jumper macho-macho
- Jumper macho-fêmea
- Anel de LED RGB
- Sensor de Toque Capacitivo TTP224

Circuito

Repetiremos uma parte do circuito do projeto da aula 5.5 Acionando um Anel LED RGB e acrescentaremos o Sensor de Toque Capacitivo TTP224.

A energia que alimentará o circuito virá do pino 3V3 da placa Raspberry Pi Pico W.

O Anel de LED é alimentado por seu terminal 5V, que deve ser ligado à linha positiva da protoboard. Já o terminal GND deve ser ligado a linha negativa da protoboard que estará conectada ao pino GND da placa Pi Pico. Os dados serão transmitidos pelo terminal DI do Anel e deve ser ligado a porta GP6 da placa.

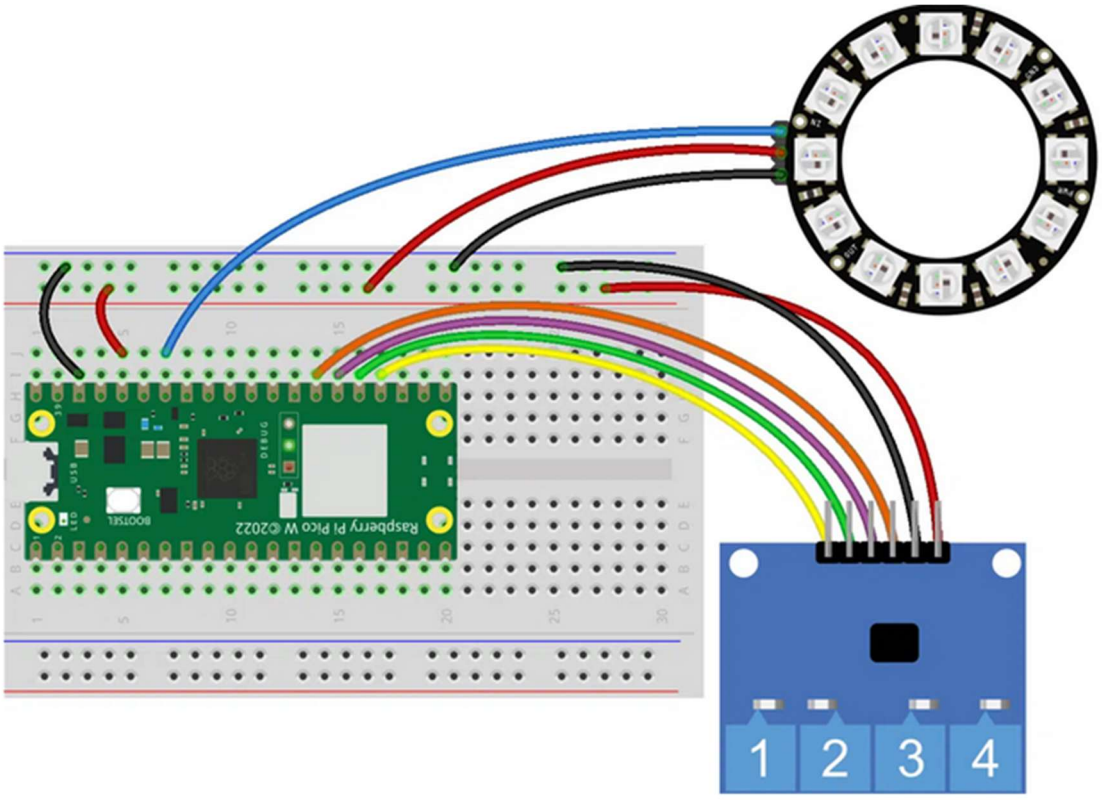
Já o Sensor de Toque Capacitivo TTP224 terá o terminal VCC ligado à linha positiva da protoboard. Já o terminal GND deve ser ligado a linha negativa da protoboard. E os pinos correspondentes das quatro chaves devem ser conectadas aos seguintes pinos:

OUT 1 > GP18

OUT 2 > GP19

OUT 3 > GP20

OUT 4 > GP21



Programa

Veja o código completo abaixo:

```
# Codigo 6.2 - Ligando luminaria com botao

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrão


import machine

import utime

# Importacao das bibliotecas customizadas

from ws2812 import WS2812

# Declaracao do Anel de LEDs
```

```
ws = WS2812(machine.Pin(28), 12)
# Declaracao dos botoes
botao_1 = machine.Pin(18, machine.Pin.IN)
botao_2 = machine.Pin(19, machine.Pin.IN)
botao_3 = machine.Pin(20, machine.Pin.IN)
botao_4 = machine.Pin(21, machine.Pin.IN)

# Declaracao das variaveis do codigo

estado_botao_1 = 0
estado_botao_2 = 0
estado_botao_3 = 0
estado_botao_4 = 0

# Laco de execucao
while True:
  estado_botao_1 = botao_1.value()
  estado_botao_2 = botao_2.value()
  estado_botao_3 = botao_3.value()
  estado_botao_4 = botao_4.value()
  if estado_botao_1 == 1:
    ws.write_all(0x444444)
  elif estado_botao_2 == 1:
    ws.write_all(0x888888)
  elif estado_botao_3 == 1:
    ws.write_all(0xFFFFFF)
  elif estado_botao_4 == 1:
```

```
        ws.write_all(0x000000)
    utime.sleep(0.01)
```

6.3 Teclado de cores

Nesta aula seguiremos aprendendo como utilizar o Módulo Sensor de Toque Capacitivo TTP224 e o anel de LED. No projeto desta aula iremos destinar uma cor de luz para cada botão do módulo.

No primeiro clique do botão ele irá acender todo o anel de LED na cor escolhida, no segundo toque do mesmo botão ele desligará os LEDs. Já ao pressionar os botões em qualquer sequência você poderá trocar a cor dos LEDs para as cores destinadas a cada botão.

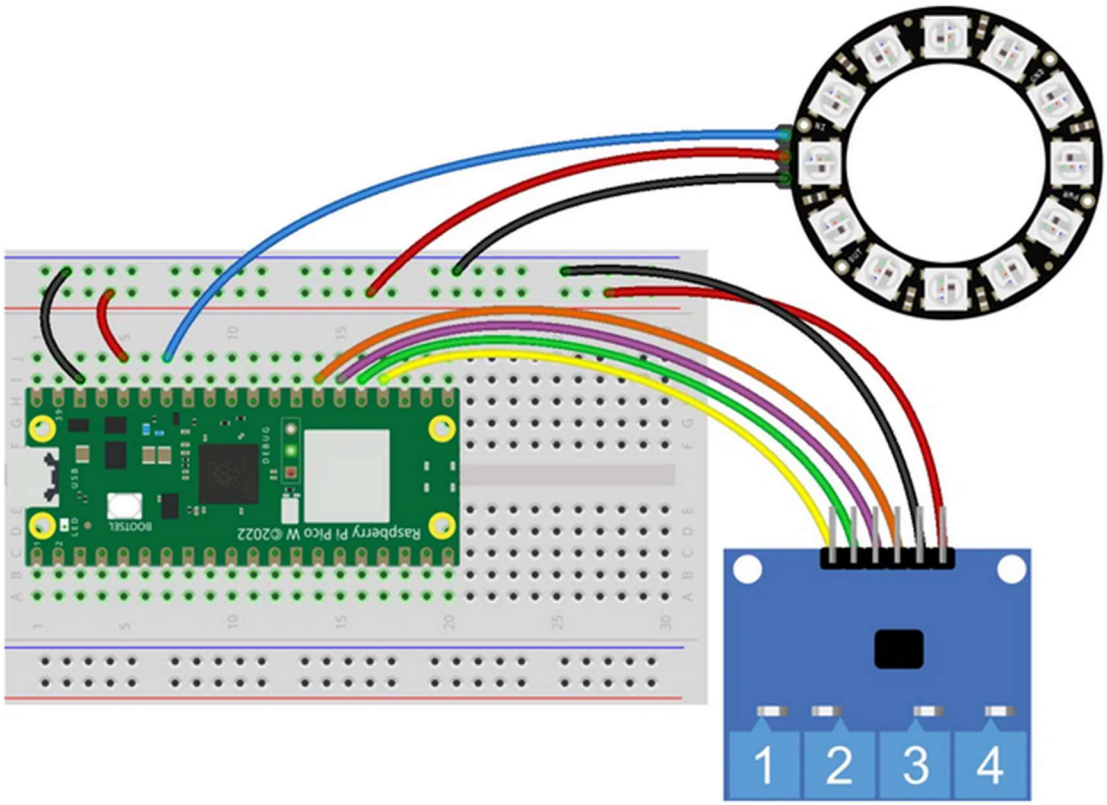
Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- Jumper macho-fêmea
- Anel de LED RGB
- Sensor de Toque Capacitivo TTP224

Circuito

O circuito deste projeto fica igual ao projeto da aula anterior, com a ligação do anel de LED e do módulo de toque capacitivo, a mudança de funções dos botões e cores dos LEDs será feita diretamente na programação. Segue abaixo o esquemático para referência:



Programa

Veja o código completo abaixo

```
# Codigo 6.3 - Teclado de Cores

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import machine

import utime

# Importacao das bibliotecas customizadas

from ws2812 import WS2812

# Declaracao do Anel de LEDs

# Declaracao dos botoes
```

```
botao_1 = machine.Pin(18, machine.Pin.IN)
botao_2 = machine.Pin(19, machine.Pin.IN)
botao_3 = machine.Pin(20, machine.Pin.IN)
botao_4 = machine.Pin(21, machine.Pin.IN)

# Declaracao das variaveis
estado_botao_1 = 0
estado_botao_2 = 0
estado_botao_3 = 0
estado_botao_4 = 0
azul_ligado = False
branco_ligado = False
verde_ligado = False
vermelho_ligado = False

# Laco de execucao
while True:
  estado_botao_1 = botao_1.value()
  estado_botao_2 = botao_2.value()
  estado_botao_3 = botao_3.value()
  estado_botao_4 = botao_4.value()
  if estado_botao_1 == 1:
    if not vermelho_ligado:
      ws.write_all(0xFF0000)
      azul_ligado = False
      branco_ligado = False
      verde_ligado = False
```

```
            vermelho_ligado = True

        else:
            ws.write_all(0x000000)
            azul_ligado = False
            branco_ligado = False
            verde_ligado = False
            vermelho_ligado = False
    elif estado_botao_2 == 1:
        if not verde_ligado:
            ws.write_all(0x00FF00)
            azul_ligado = False
            branco_ligado = False
            verde_ligado = True
            vermelho_ligado = False

        else:

            ws.write_all(0x000000)
            azul_ligado = False
            branco_ligado = False
            verde_ligado = False
            vermelho_ligado = False
    elif estado_botao_3 == 1:
        if not azul_ligado:
            ws.write_all(0x0000FF)
            azul_ligado = True
            branco_ligado = False
```

```
        verde_ligado = False
        vermelho_ligado = False

    else:
        ws.write_all(0x000000)
        azul_ligado = False
        branco_ligado = False
        verde_ligado = False
        vermelho_ligado = False

elif estado_botao_4 == 1:
    if not branco_ligado:
        ws.write_all(0xFFFFFF)
        azul_ligado = False
        branco_ligado = True
        verde_ligado = False

        vermelho_ligado = False

    else:

        ws.write_all(0x000000)
        azul_ligado = False
        branco_ligado = False
        verde_ligado = False
        vermelho_ligado = False
utime.sleep(0.2)
```

7. Sensor de Movimento

Neste módulo aprenderemos sobre o Mini Sensor de Movimento e Presença PIR e as possibilidades de aplicações deste componente.

7.1 O que é o Mini Sensor de Movimento Presença PIR?

O Mini Sensor PIR é um dispositivo altamente sensível projetado para detectar movimento por meio de mudanças no calor emitido pelos objetos em seu campo de visão. Com uma ampla faixa de alimentação de 2,7 a 12 V e um consumo de energia estático extremamente baixos, é adequado para uma variedade de aplicações de automação residencial e segurança.

Com um tempo de atraso e bloqueio de 2 segundos, este sensor de presença oferece resposta rápida e eficiente. Seu alcance de até 5 metros e temperatura de operação de -20 a 60°C garantem sua eficácia em uma variedade de ambientes. Com dimensões compactas, é fácil de integrar em diversos projetos.

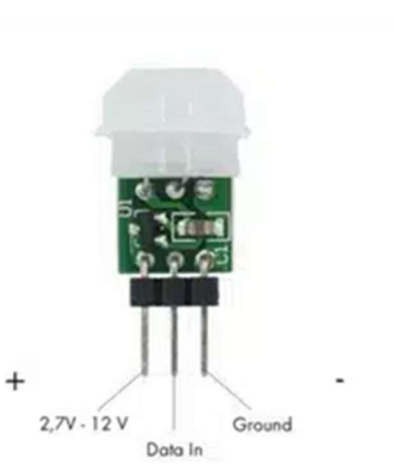
Como funciona o Mini Sensor PIR?

O Mini Sensor PIR (Passive Infrared Sensor) detecta movimento através da detecção de mudanças no ambiente causadas por calor emitido por corpos em movimento. Ele contém um sensor piroelétrico que converte variações de calor em sinais elétricos.

Quando um objeto dentro do campo de visão do sensor emite calor, ocorre uma alteração na distribuição térmica detectada pelo sensor de movimento. Isso resulta em um sinal de saída indicando a presença de movimento. O sensor é passivo, não emitindo luz ou energia.

Pinagem do Mini Sensor PIR

O Mini Sensor PIR geralmente possui três pinos, que são:



VCC: Este pino é conectado à fonte de alimentação (geralmente 5V).

GND: Este pino é conectado ao terra (GND) do circuito.

OUT: Este pino é a saída digital do sensor e é conectado a um pino digital do microcontrolador, onde envia um sinal de alto ou baixo para indicar a detecção de movimento ou presença.

7.2 Detector de movimento

Neste projeto simulamos um detector de movimento como os que temos nos sistemas usados em casa. Quando o movimento é detectado um LED acende marcando a detecção.

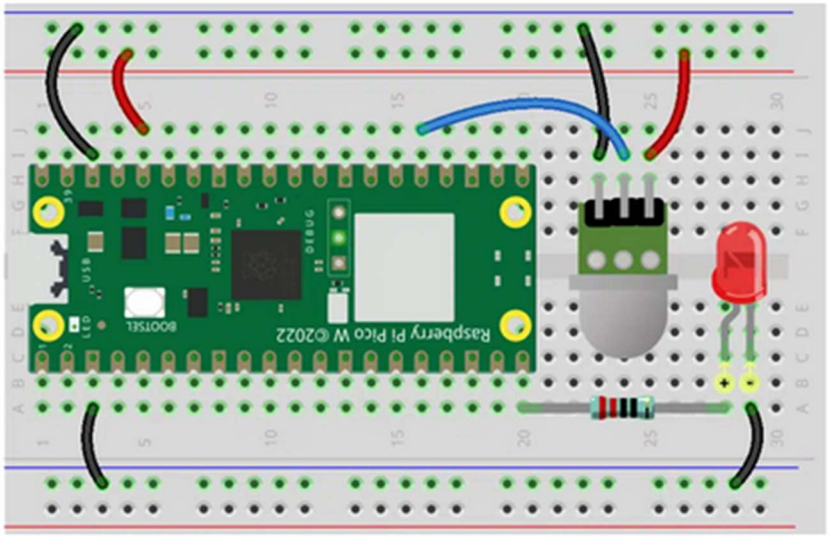
Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- LED Vermelho 5mm
- Resistor 220 ohm
- Mini Sensor PIR

Circuito

Para o circuito repetiremos as ligações do LED como vimos em projetos anteriores. Prestando atenção na polaridade dos terminais do LED e sempre lembrando de adicionar o resistor 220 ohm ao terminal positivo



E adicionaremos o Mini Senor PIR, com as seguintes conexões:

- O primeiro terminal, identificado pelo +, será conectado à linha positiva da protoboard.
- O terminal central, o de Dados, deve ser conectado ao pino GP19.
- O terceiro terminal, o GND, deve ser ligado à linha negativa da protoboard que estará conectada ao pino GND da placa Pico W.

Programa

Nesse programa vamos utilizar duas novidades, vamos criar nossa própria função para deixar o código um pouco mais legível e iremos utilizar interrupções que substituem a leitura do estado do botão.

A função é um segmento de código que realiza uma tarefa específica ao ser chamada. Já utilizamos funções antes, mas precisamos entender um pouco mais cada elemento para construir a nossa própria. A função pode receber um ou mais parâmetros para desempenhar seu papel, que devem ser descritos na definição da função entre parênteses.

Se for mais de um, eles devem ser separados por vírgulas.

Então colocamos dentro da função todas as variáveis que ela porventura precisar criar e todas as ações. Uma função pode ou não retornar um valor ao seu término.

Por padrão no micropython, todas as variáveis que forem colocadas como argumento na definição da função ou que são criadas dentro da função são conhecidas apenas pela função, não são vistas pelo programa todo.

Caso precisemos mexer com uma variável que o resto do programa precisa conhecer, precisamos primeiro declarar a variável dentro da função com o modificador global. Depois disso o código irá reconhecer que deve utilizar a variável do resto do código e não uma nova conhecida apenas pela função.

Veja o código completo abaixo

```
# Codigo 7.2 - Detector de movimento
# Bibliotecas necessarias para o codigo
# Importacao das bibliotecas padrao

import machine
import utime
# Declaracao do sensor PIR

sensor_pir = machine.Pin(19, machine.Pin.IN)
# Declaracao do LED
led = machine.Pin(15,machine.Pin.OUT)
# Declaracao das variaveis do codigo
movimento = False
# Funcao a ser chamada quando houver uma borda positiva no pino do sensor PIR
def movimento_detectado(_):
  global movimento
  movimento = True

# Declaracao da interrupcao no pino do sensor PIR
sensor_pir.irq(trigger=machine.Pin.IRQ_RISING, handler=movimento_detectado)
 # Laco de execucao
while True:
  if movimento:
    led.value(1)
    print("Alguem passou aqui!")
    utime.sleep(3)
```

```
    led.value(0)

    movimento = False
```

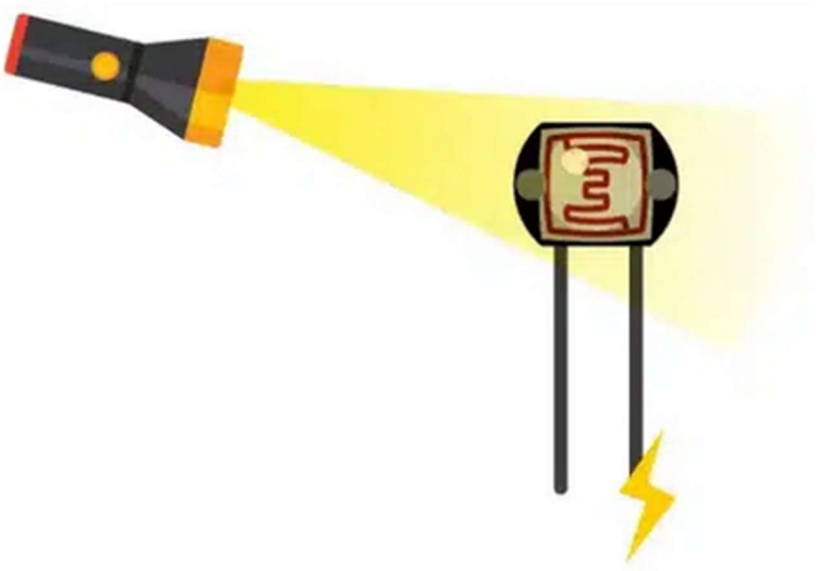
8. Sensor de Luminosidade

Neste módulo apresentaremos o Sensor de Luminosidade LDR, que tem como princípio de funcionamento captando a incidência de luz no ambiente.

O LDR é bastante usado nos postes de luz na cidade, fazendo que quando anoitece as luzes da cidade acendam. Ele também é usado em lâmpadas de jardim que acendem ao anoitecer. Veja neste módulo como utilizar o sensor de luz LDR com MicroPhyton.

O LDR é um fotoresistor ou fotocélula, que é um resistor variável controlado por luz. A resistência de um fotoresistor diminui com o aumento da intensidade da luz incidente; em outras palavras, exibe fotocondutividade.

Um fotoresistor pode ser aplicado em circuitos detectores sensíveis à luz e circuitos de comutação ativados por luz e escuro, atuando como um semicondutor de resistência. No escuro, um fotoresistor pode ter uma resistência tão alta quanto vários megaohms (MΩ), enquanto na luz, um fotoresistor pode ter uma resistência tão baixa quanto algumas centenas de ohms.



Assim como um potenciômetro varia sua resistência conforme a rotação, o LDR é um resistor que varia sua resistência conforme a intensidade de luz no ambiente. Com isso conseguimos medir a quantidade de luz presente em um ambiente.

8.2 Sensor de luz ambiente

O fotoresistor é um dispositivo típico para entradas analógicas e é utilizado de forma muito semelhante a um potenciômetro. Seu valor de resistência depende da intensidade da luz, quanto mais forte for a luz irradiada, menor será seu valor de resistência; inversamente, aumenta.

Neste projeto vamos colocar em prática o uso do sensor de luminosidade. Portanto vamos simular o funcionamento de um poste de luz que vemos na rua. Quando o dia acaba e a noite chega o poste acaba acendendo automaticamente. Assim neste projeto usaremos o LDR e um LED.

Assim, quando o sensor está captando que tem luz no ambiente o LED fica apagado, ao cobrir o sensor de luz, simulando a noite, o LED acende. Você pode ajustar os parâmetros de luminosidade que o programa considerará “noite” ou “dia”.

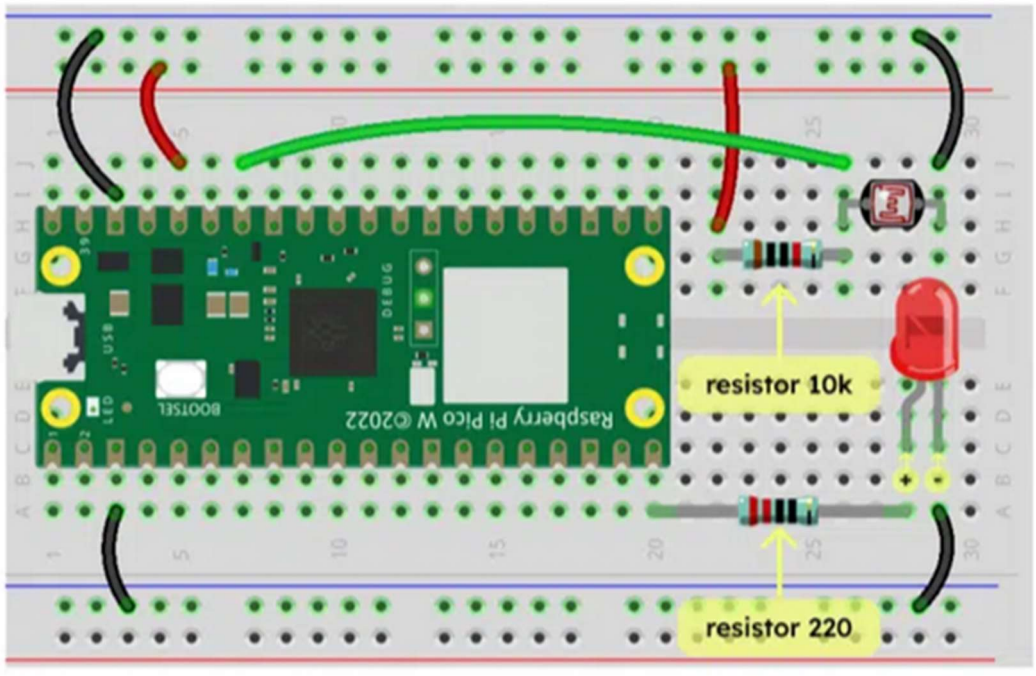
Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- LED Vermelho 5mm
- Sensor de luz LDR
- Resistor 220 ohm
- Resistor 10K ohm

Circuito

Neste circuito repetiremos as ligações do LED como vimos em projetos anteriores. Prestando atenção na polaridade dos terminais do LED e sempre lembrando de adicionar o resistor 220 ohm ao terminal positivo.



Neste circuito, o resistor de 10K e o LDR estão conectados em série, e a corrente que passa por eles é a mesma. O resistor de 10K atua como proteção, e o pino GP28 da placa lê o valor após a conversão de tensão do fotoresistor.

Observação: Note que o LDR deve ser ligado somente nos pinos GP26, GP27 e GP28, pois são os únicos pinos da Pico W que possuem conversor analógico digital.

O conversor analógico-digital (frequentemente abreviado por conversor A/D ou ADC) é um dispositivo eletrônico capaz de gerar uma representação digital a partir de uma grandeza analógica, normalmente um sinal representado por um nível de tensão ou intensidade de corrente elétrica. No caso do LDR a entrada converte o sinal de tensão em dados para o programa ler, interpretar e gerar uma resposta que determinamos, no caso acender ou apagar o LED.

Programa

Para utilizar os conversores analógicos-digitais da placa, precisamos fazer a declaração dos pinos de acordo. Para isso utilizaremos a função ADC da biblioteca machine

```
# Codigo 8.2 - Sensor de luz ambiente


# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao


import machine

import utime

# Declaracao do LDR

ldr = machine.ADC(28)

# Declaracao do LED

led = machine.Pin(15, machine.Pin.OUT)

# Laco de execucao

while True:
```

```
valor_de_luminosidade  = ldr.read_u16()
if valor_de_luminosidade > 20000:
    led.value(1)
else:
    led.value(0)
utime.sleep_ms(10)
```

9. Sensor de Umidade e Temperatura

Neste módulo aprenderemos os conceitos de leitura de umidade e temperatura através do sensor DHT11 que é um componente versátil amplamente utilizado em projetos de automação residencial, monitoramento ambiental e controle climático. Com suas especificações notáveis, como faixa de umidade de 20% a 90%, precisão de ±5% e faixa de temperatura de 0°C a 50°C com precisão de ±2°C, o DHT11 oferece medições confiáveis.

9.1 O que é o Sensor de Umidade e Temperatura DHT11?

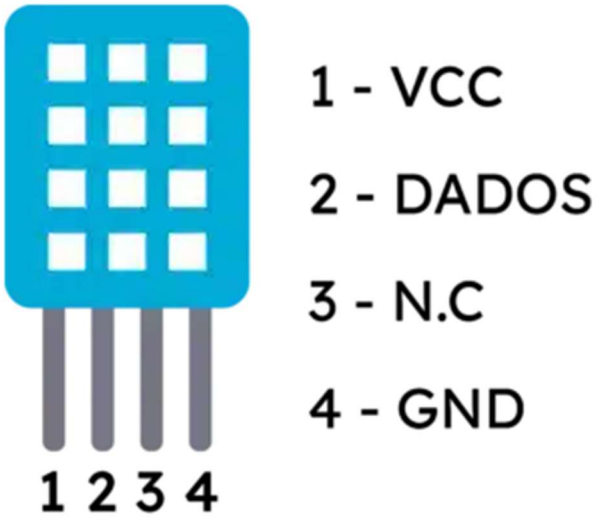
O Sensor de Umidade e Temperatura DHT11 opera com base em um sensor capacitivo para medir a umidade relativa do ar e um termistor para medir a temperatura ambiente. Ele possui uma membrana porosa que permite que a umidade do ar entre em contato com o sensor capacitivo, alterando sua capacitância de acordo com a umidade. O termistor registra a variação de temperatura e ambos os valores são convertidos em sinais digitais pelo sensor.

Esses sinais digitais são então lidos pelo microcontrolador (como a Pico W) por meio de um protocolo de comunicação serial. Assim, o microcontrolador pode interpretar os dados e fornecer leituras de umidade e temperatura em tempo real. Este processo permite que o DHT11 forneça medições precisas e confiáveis da umidade e temperatura ambiente.

Pinagem do Sensor de Umidade e Temperatura DHT11

O sensor de umidade e temperatura DHT11 possui quatro pinos: VCC, DADOS, N.C e GND. O pino VCC é conectado à alimentação, o pino DADOS é usado para

enviar os dados do sensor para o microcontrolador (no caso deste kit a Pico W) e o pino GND é conectado à terra. O pino N.C não deve ser conectado.



9.2 Termômetro luminoso

Neste projeto aprenderemos a fazer a leitura dos dados de temperatura do DHT11 e iremos fazer uma representação visual através do Anel de LEDs. Assim definiremos uma faixa de temperatura para cada cor, ao ir atingindo a temperatura da faixa o círculo do Anel de LEDs irá ascendendo gradativamente e se completará. Mudando de cor para a próxima faixa de temperaturas.

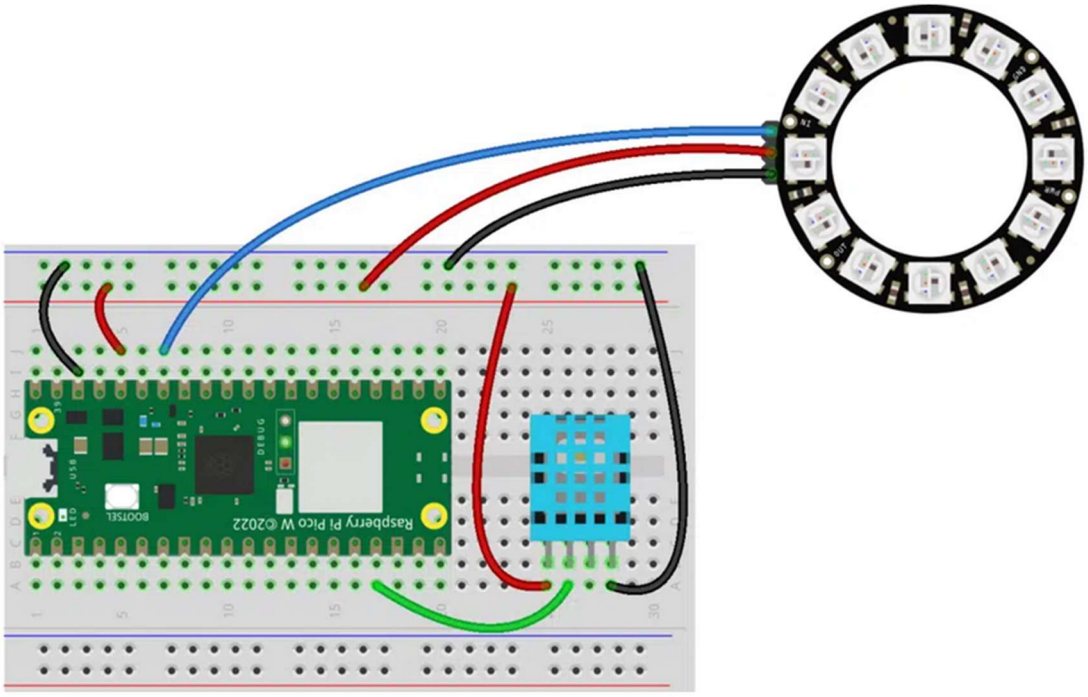
Material Necessário

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- Anel de LED RGB
- Sensor de Temperatura e Umidade DHT11

Montagem do Circuito

Como padrão para a placa Raspberry Pi Pico W a energia que alimentará o circuito virá do pino 3V3(OUT).



O Anel de LED é alimentado por seu terminal 5V, que tem o fio vermelho soldado, e que deve ser ligado à linha positiva da protoboard. Já o terminal GND do Anel de LED, que tem o fio preto, deve ser ligado a linha negativa da protoboard que estará conectada ao pino GND da placa Pi Pico. Os dados serão transmitidos pelo terceiro terminal do Anel, o terminal DI, e deve ser ligado à porta GP28 da placa.

Já as conexões do Sensor DHT11 serão as seguintes, primeiro terminal VCC será à linha positiva da protoboard. O segundo terminal, o de Dados, deve ser conectado ao pino GP13. E o quarto terminal, o GND, deve ser ligado à linha negativa da protoboard que estará conectada ao pino GND da placa Pico W.

Programa

Para este projeto vamos utilizar a biblioteca **dht**. Caso não se lembre como realizar a instalação, veja novamente os passos da aula 4.4 Como fazer upload de bibliotecas na Pico W.

Como o sensor DHT11 é mais sensível e nem sempre consegue concluir as leituras em tempo, existe a possibilidade de os dados não estarem prontos e isso causa um erro na biblioteca.

Sem uma previsão de erro, o código simplesmente pararia e seria encerrado. Para que isso não aconteça existe a estrutura try except. Nessa estrutura em try tentamos executar um comando ou uma parte do código, e caso retorne um erro, executa a ou as instruções presentes na estrutura except. No exemplo abaixo tentamos realizar a leitura do sensor e atribuir o valor de da leitura a variável temperatura. Caso aconteça um erro então atribuimos a string 'err' a variavel temperatura na extrutura except. Como as demais estruturas do micropython, a identação separa os blocos de código.

```
try:
    sensor.measure()
    temperatura = sensor.temperature()
except:
    temperatura = 'err'
```

Veja o código completo abaixo

```
# Codigo 9.2 - Termometro Luminoso
# Bibliotecas necessarias para o codigo
# Importacao das bibliotecas padrao
from machine import Pin, I2C
import utime
# Importacao das bibliotecas customizadas
from dht import DHT11
from ws2812 import WS2812
# Declaracao do Anel de LEDs
ws = WS2812(machine.Pin(28), 12)
# Declaracao do Sensor de temperatura e umidade
pin = Pin(13, Pin.IN, Pin.PULL_UP)
sensor = DHT11(pin)
# Atraso inicial para inicializacao do sensor
utime.sleep(5)
```

```
# Saida que nao sofrera alteracao no console
print('Dados do sensor:')

# Laco de execucao
while True:
  try:
    sensor.measure()
    temperatura = sensor.temperature()
  except:
    temperatura = 'err'
  string = "Temperatura: {}°C".format(temperatura)
  print("\r",string,end='')
  if temperatura <= 16:
    ws.write(0,0x0000FF)
    for i in range(1,12):
      ws.write(i,0x000000)
  elif temperatura <= 18:
    for i in range(2):
      ws.write(i, 0x0000FF)
    for i in range(2,12):
      ws.write(i,0x000000)
  elif temperatura <= 20:
    for i in range(3):
      ws.write(i, 0x0033FF)
    for i in range(3,12):
      ws.write(i,0x000000)
  elif temperatura <= 22:
    for i in range(4):
```

```
        ws.write(i, 0x00AAFF)
    for i in range(4,12):


        ws.write(i,0x000000)
elif temperatura <= 24:
    for i in range(5):
        ws.write(i, 0x00FF00)
    for i in range(5,12):
        ws.write(i,0x000000)
elif temperatura <= 26:
    for i in range(6):
        ws.write(i, 0x00FF00)
    for i in range(6,12):
        ws.write(i,0x000000)
elif temperatura <= 28:
    for i in range(7):
        ws.write(i, 0xFF5500)
    for i in range(7,12):
        ws.write(i,0x000000)
elif temperatura <= 30:
    for i in range(8):
        ws.write(i, 0xFFAA00)
    for i in range(8,12):
        ws.write(i,0x000000)
elif temperatura <= 32:
    for i in range(9):
        ws.write(i, 0xFF0000)
    for i in range(9,12):
```

```
        ws.write(i,0x000000)
    elif temperatura <= 34:


      for i in range(10):
        ws.write(i, 0xFF0000)
      for i in range(10,12):
        ws.write(i,0x000000)
    elif temperatura <= 36:
      for i in range(11):
        ws.write(i, 0xFF0000)
      for i in range(11,12):
        ws.write(i, 0x000000)
    elif temperatura <= 38:
        ws.write_all(0xFF0000)
    utime.sleep(4)
```

9.3 Higrômetro luminoso

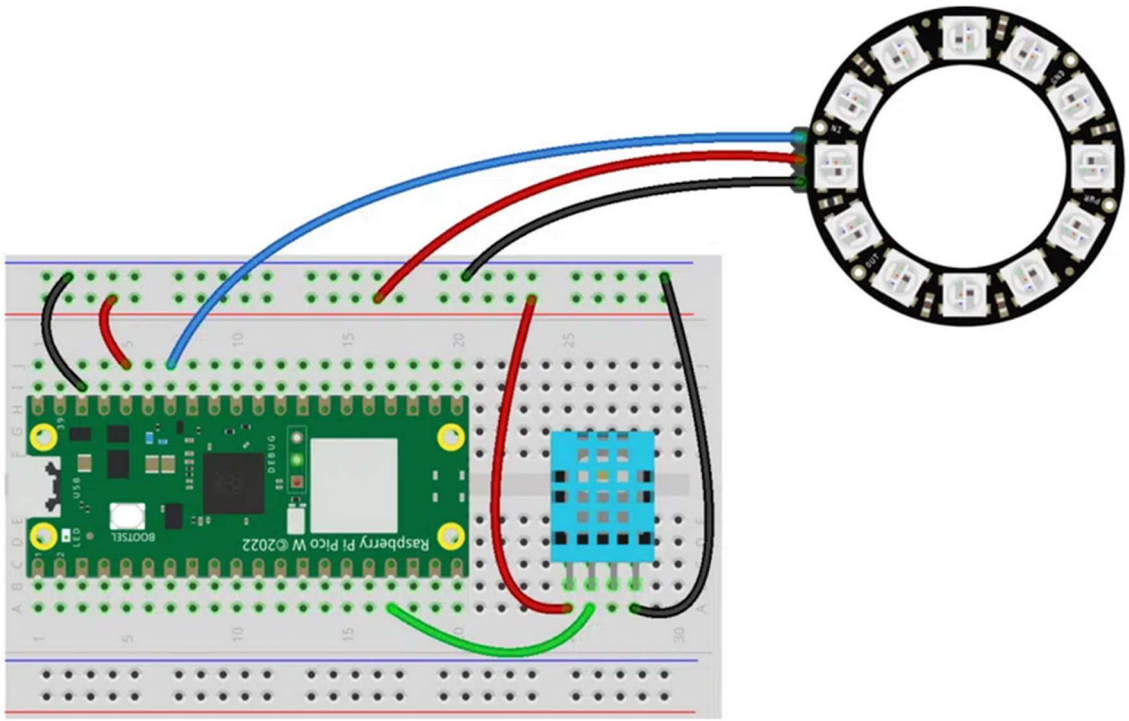
Já sabemos monitorar a temperatura, agora vamos monitorar a umidade do ambiente, a outra grandeza que o sensor DHT11 permite que façamos medição.

Material Necessário

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- Anel de LED RGB
- Sensor de Temperatura e Umidade DHT11

Montagem do circuito

Nesse projeto iremos utilizar a mesma montagem do projeto anterior, veja o esquemático completo abaixo



Programa

O programa é muito parecido com o do projeto anterior, mas ao invés de lermos a temperatura, iremos ler a umidade retornada pelo sensor

Codigo 9.2 - Higrometro Luminoso

Bibliotecas necessarias para o codigo

Importacao das bibliotecas padrao

from machine import Pin, I2C

import utime

Importacao das bibliotecas customizadas

from dht import DHT11

from ws2812 import WS2812

Declaracao do Anel de LEDs

```
ws = WS2812(machine.Pin(28), 12)
# Declaracao do Sensor de temperatura e umidade

pin = Pin(13, Pin.IN, Pin.PULL_UP)

sensor = DHT11(pin)
# Atraso inicial para inicializacao do sensor
utime.sleep(5)
# Saida que nao sofrera alteracao no console
print('Dados do sensor:')
# Laco de execucao
while True:
  try:
    sensor.measure()
    temperatura = sensor.temperature()
    umidade = sensor.humidity()
  except:

    temperatura = 'err'
    umidade = 'err'
  string = "Temperatura:{}°C Umidade: {}%".format(temperatura, umidade)
  print("\r",string,end='')
  if umidade <= 10:
    ws.write(0,0xFF0000)
    for i in range(1,12):
      ws.write(i,0x000000)
  elif umidade <= 18:
    for i in range(2):
```

```
        ws.write(i, 0xFF0000)


    for i in range(2,12):


        ws.write(i,0x000000)
elif umidade <= 26:
    for i in range(3):
        ws.write(i, 0xFF0000)
    for i in range(3,12):
        ws.write(i,0x000000)
elif umidade <= 34:
    for i in range(4):
        ws.write(i, 0xFF3300)
    for i in range(4,12):
        ws.write(i,0x000000)
elif umidade <= 42:
    for i in range(5):
        ws.write(i, 0xFFAA00)
    for i in range(5,12):
        ws.write(i,0x000000)
elif umidade <= 50:
    for i in range(6):
        ws.write(i, 0xBBBB00)
    for i in range(6,12):
        ws.write(i,0x000000)
elif umidade <= 58:
    for i in range(7):
        ws.write(i, 0x00FF00)
```

```
    for i in range(7,12):
        ws.write(i,0x000000)

elif umidade <= 66:

    for i in range(8):
        ws.write(i, 0x00FF00)
    for i in range(8,12):
        ws.write(i,0x000000)
elif umidade <= 72:
    for i in range(9):
        ws.write(i, 0x00FFFF)
    for i in range(9,12):
        ws.write(i,0x000000)
elif umidade <= 80:
    for i in range(10):
        ws.write(i, 0x0055FF)
    for i in range(10,12):
        ws.write(i,0x000000)
elif umidade <= 88:
    for i in range(11):
        ws.write(i, 0x0000FF)
    for i in range(11,12):
        ws.write(i, 0x000000)
elif umidade <= 90:
        ws.write_all(0x0000FF)
utime.sleep(4)
```

10. Buzzer

Neste módulo você será apresentado ao buzzer, uma forma de alertar e sinalizar com som o usuário dos seus projetos.

10.1 O que é o Buzzer?

Um “buzzer” é um dispositivo eletrônico que produz um som contínuo ou intermitente, muitas vezes semelhante a um zumbido ou bipe. Esses dispositivos são comumente usados em uma variedade de aplicações, como alarmes, sinalizações, jogos eletrônicos, circuitos eletrônicos e instrumentos musicais.

No contexto de eletrônica e engenharia, um buzzer pode ser um buzzer converte sinais elétricos em vibrações mecânicas, resultando em um som audível. Eles são frequentemente utilizados para fornecer feedback sonoro ou para indicar a ocorrência de eventos específicos em diferentes dispositivos e sistemas.

O Buzzer possui dois pinos com pólos positivo e negativo. Com um + na superfície e o terminal mais comprido representa o ânodo e o outro é o cátodo.

Tipos de buzzers

Existem principalmente dois tipos de buzzers: os ativos (ativos eletronicamente) e os passivos (mecânicos). Abaixo, descrevemos brevemente cada tipo:

Buzzer Ativo

Eletrônico: Os buzzers ativos contêm um circuito eletrônico interno que gera o som quando alimentado com uma corrente elétrica.

Menos Controle de Frequência: Tendem a oferecer menos controle sobre a frequência do som, resultando em tons mais fixos e limitados.

Comum em Aplicações Simples: São frequentemente encontrados em aplicações mais simples, como brinquedos, sinalizações básicas e projetos DIY.

Buzzer Passivo

Mecânico: Os buzzers passivos dependem de um componente mecânico, geralmente um cristal piezoelétrico, para criar o som quando uma corrente é aplicada.

Frequência Controlada: Eles oferecem maior controle sobre a frequência do som produzido, permitindo ajustes precisos conforme necessário.

Simplicidade e Durabilidade: São mais simples em design e, em alguns casos, mais duráveis devido à natureza mecânica do componente.

Ampla Aplicação: São comumente utilizados em eletrônicos modernos, como dispositivos de consumo, sistemas de alarme, e outros onde é necessário um controle mais refinado sobre o som.

Ambos os tipos de buzzers têm suas vantagens e desvantagens, e a escolha entre eles depende das necessidades específicas da aplicação. O buzzer ativo é mais versátil em termos de facilidade de uso, enquanto o buzzer passivo é muitas vezes escolhido por sua simplicidade e durabilidade em aplicações mais básicas.

10.2 Acionando um Buzzer

Mas e se precisarmos chamar ainda mais a atenção para o nosso projeto? Uma luz você precisa estar olhando para ela para que ela sirva de alerta. Agora um som será ouvido de qualquer forma. Então vamos aprender como acionar um buzzer para ampliar as possibilidades de nossos projetos.

Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- Buzzer Ativo 5V
- Resistor 220 ohm

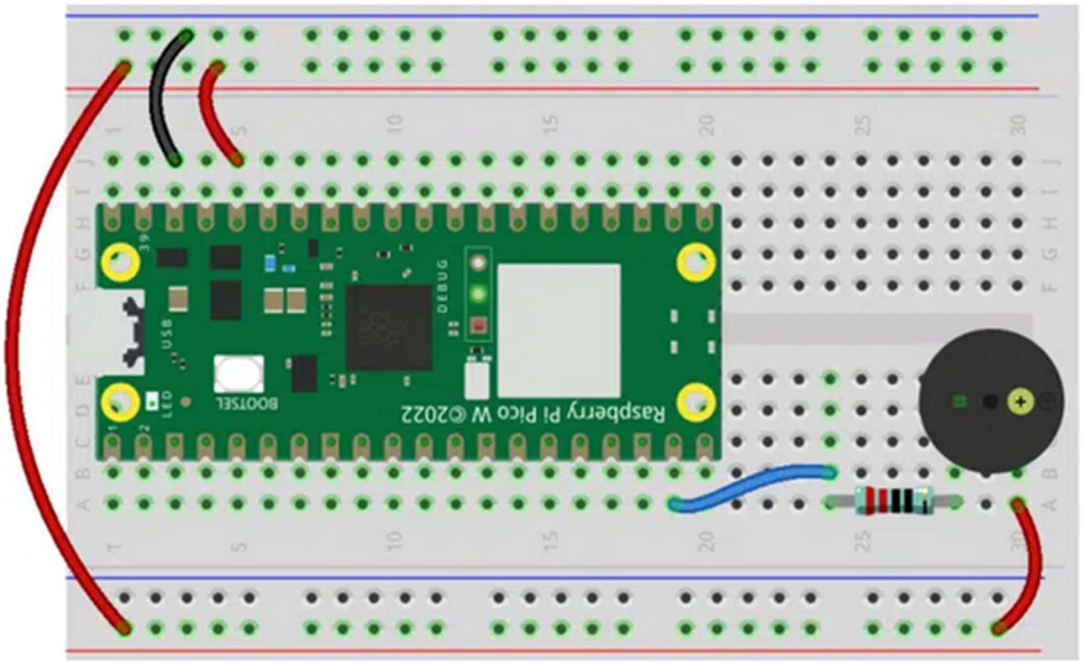
Circuito

Para este projeto teremos o seguinte esquemático:

Como padrão para a placa Raspberry Pi Pico W a energia que alimentará o circuito virá do pino 3V3(OUT).

Já o terminal positivo do buzzer, terminal maior, será conectado a linha positiva da protoboard.

O terminal negativo do buzzer será ligado em série ao resistor 220 ohm e ao pino GP14 da placa.



Programa

Este código não trará novidades em termos de funções e estruturas. Veja o código completo abaixo

```
# Codigo 10.2 - Acionando um buzzer

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import machine

import utime

# Declaracao do buzzer

buzzer = machine.PWM(machine.Pin(14))

# Definicao da frequencia do PWM e periodo que aciona o buzzer

buzzer.freq(6000)

buzzer.duty_u16(30000)
```

10.3 Fazendo música

Neste projeto faremos a simulação do funcionamento de um piano. Utilizando o Sensor de Toque Capacitivo TTP224, teremos somente quatro teclas à disposição. Entretanto, já conseguimos testar o princípio de que cada botão emitirá um som através do buzzer.

Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

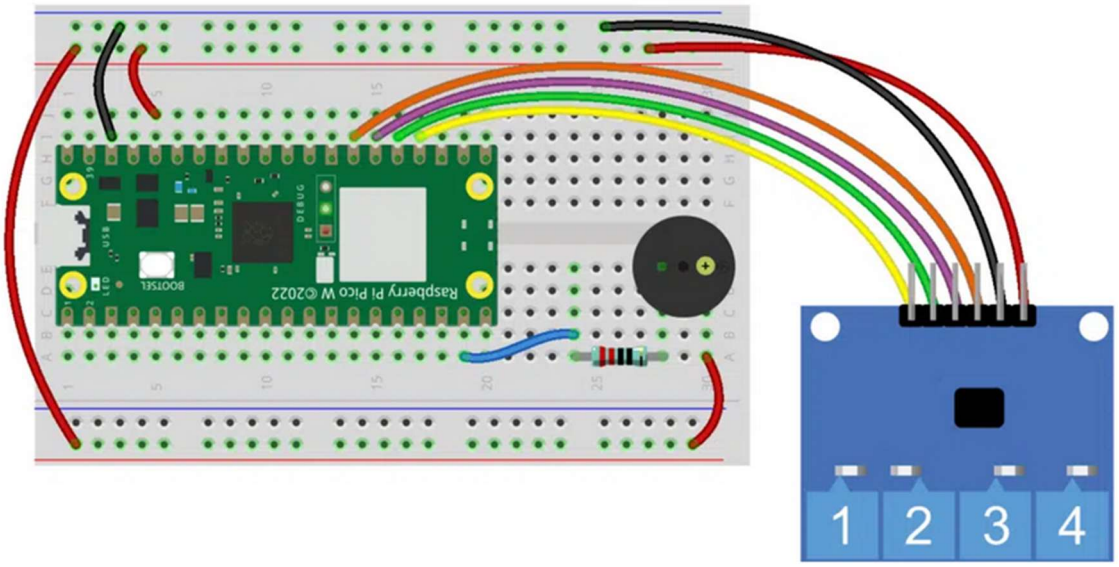
- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- Jumper macho-fêmea
- Buzzer Ativo 5V
- Resistor 220 ohm
- Sensor de Toque Capacitivo TTP224

Circuito

Para este projeto repetiremos o esquemático do projeto anterior e adicionaremos as conexões do Sensor de Toque Capacitivo TTP224. Para isso teremos o seguinte esquemático:

O Sensor de Toque Capacitivo TTP224 terá o terminal VCC ligado à linha positiva da protoboard. Já o terminal GND deve ser ligado a linha negativa da protoboard. E os pinos correspondentes das quatro chaves devem ser conectadas aos seguintes pinos:

- OUT 1 > GP18
- OUT 2 > GP19
- OUT 3 > GP20
- OUT 4 > GP21



Programa

Este código não trará novidades em termos de funções e estruturas. Nas variáveis temos as frequências de cada nota que iremos tocar.

1NOTA_C4 = 262

2NOTA_D4 = 294

3NOTA_E4 = 330

4NOTA_F4 = 349

Você pode alterar os valores, que são em Hz, para alterar as notas e fazer as suas próprias variações.

Veja o código completo abaixo

```
# Codigo 10.3 - Fazendo musica

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import machine

import utime

# Declaracao do buzzer


buzzer = machine.PWM(machine.Pin(14, machine.Pin.OUT))
```

```
# Declaracao dos botoes
botao_1 = machine.Pin(21, machine.Pin.IN)
botao_2 = machine.Pin(20, machine.Pin.IN)
botao_3 = machine.Pin(19, machine.Pin.IN)
botao_4 = machine.Pin(18, machine.Pin.IN)
# Definicao das variaveis do codigo

NOTA_C4 =  262
NOTA_D4 =  294
NOTA_E4 =  330
NOTA_F4 =  349

estado_botao_1 = 0
estado_botao_2 = 0
estado_botao_3 = 0
estado_botao_4 = 0

# Laco de execucao

while True:

  estado_botao_1 = botao_1.value()
  estado_botao_2 = botao_2.value()
  estado_botao_3 = botao_3.value()
  estado_botao_4 = botao_4.value()
```

```
  valor = (estado_botao_1 << 3) + (estado_botao_2 << 2) + (estado_botao_3 << 1) +
estado_botao_4


  if valor == 0:
    buzzer.freq(100)
    buzzer.duty_u16(65535)
  elif valor == 1:
    buzzer.freq(NOTA_C4)
    buzzer.duty_u16(32767)
    utime.sleep_ms(100)
  elif valor == 2:
    buzzer.freq(NOTA_D4)
    buzzer.duty_u16(32767)
    utime.sleep_ms(100)
  elif valor == 4:
    buzzer.freq(NOTA_E4)
    buzzer.duty_u16(32767)
    utime.sleep_ms(100)
  elif valor == 8:
    buzzer.freq(NOTA_F4)
    buzzer.duty_u16(32767)
    utime.sleep_ms(100)
  else:
    buzzer.duty_u16(65535)
  utime.sleep(0.01)
```

10.4 Theremim

Theremin é um instrumento musical eletrônico que não requer contato físico. Com base na posição da mão do músico, produz tons diferentes.

Sua seção de controle geralmente é composta por duas antenas metálicas que detectam a posição das mãos do terminista e controlam os osciladores com uma mão e o volume com a outra. Os sinais elétricos do theremin são amplificados e enviados para um alto-falante.

Não podemos reproduzir o mesmo instrumento através da Pico W, mas podemos usar fotoresistor e o buzzer para obter uma reação semelhante.

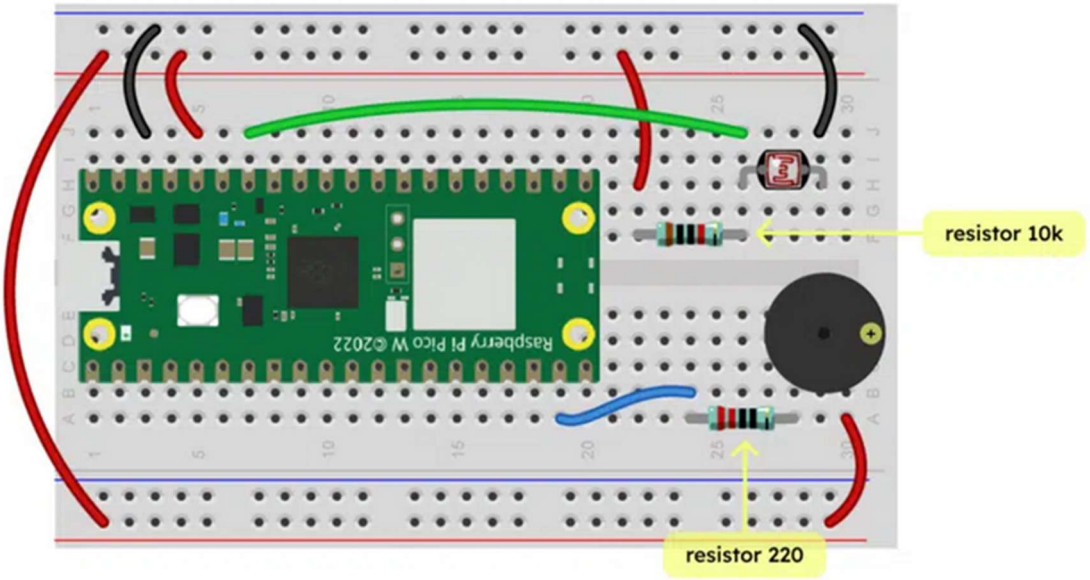
Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- Jumper macho-fêmea
- Buzzer Ativo 5V
- Sensor de luz LDR
- Resistor 220 ohm
- Resistor 10K ohm

Circuito

Para este projeto teremos o seguinte esquemático:



Programa

Neste código precisamos de uma função para mapear um intervalo para outro distinto. Nossa função precisará receber um valor, o intervalo a que ele pertence e o intervalo para o qual ele deve ser mapeado e retornar o valor mapeado no novo intervalo. Pense no número 100 num intervalo de 0 a 200, corresponde a 50% da escala. Se mapearmos para um intervalo de 0 a 100, o número que corresponde a 50% nesse intervalo é 50. Essa é a lógica da nossa função.

Para que uma função retorne um valor usamos a expressão return seguida do valor que queremos retornar, como o exemplo abaixo

```
def mapeamento_de_intervalo(x, in_min, in_max, out_min, out_max):


    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
```

Confira o código completo abaixo

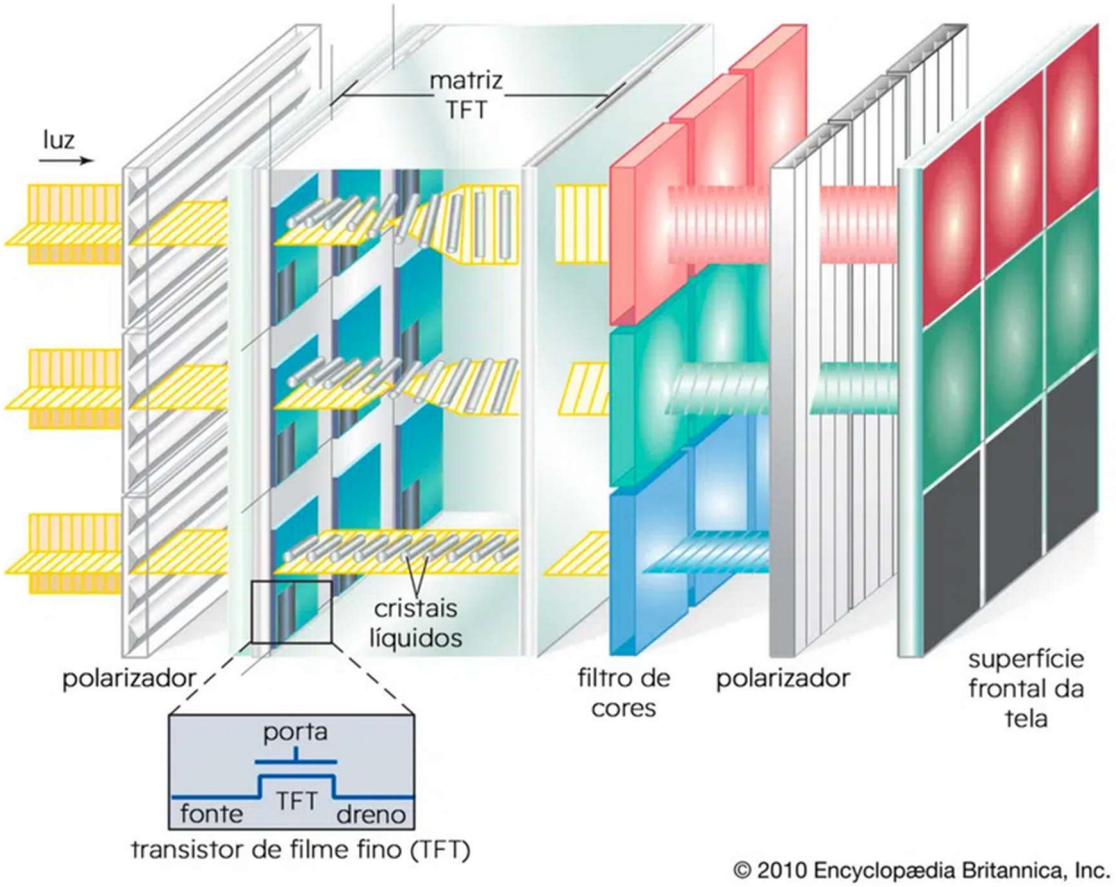
```
# Codigo 10.4 - Theremin

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao


import machine
```

```
import utime
# Declaracao do LED
led = machine.Pin(16, machine.Pin.OUT)
# Declaracao do LDR
LDR = machine.ADC(28)
# Declaracao do buzzer
buzzer = machine.PWM(machine.Pin(14))
# Declaracao das variaveis do codigo
pouca_luz=3000
muita_luz=55000
# Funcao que mapeia um intervalo de valores para outro
def mapeamento_de_intervalo(x, in_min, in_max, out_min, out_max):
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
# Definicao do ciclo de trabalho do PWM
buzzer.duty_u16(30000)
# Laco de execucao
while True:
  valor_de_luminosidade  = LDR.read_u16()
  frequencia =
int(mapeamento_de_intervalo(valor_de_luminosidade,pouca_luz,muita_luz,50,60
00))
  buzzer.freq(frequencia)
  utime.sleep_ms(10)
```

11. Display

11.1 O que é o Display LCD?

O Display LCD (do inglês Liquid Crystal Display) é um tipo de display que se vale das propriedades de materiais do tipo cristal líquido. O cristal líquido é um tipo de substância que apresenta algumas propriedades físicas de líquido, em especial a consistência, e outras propriedades de cristais, em especial quanto à dispersão de luz. Ele também reage ao campo elétrico, alterando a orientação e consequentemente as características de transmissão de luz. Graças a essas propriedades, o cristal líquido é utilizado na composição de várias camadas que guiam a luz e permitem ou não a sua passagem. É feita a montagem dessas estruturas de forma microscópica para formar os pontos de uma imagem no caso de displays de resolução elevada para o tamanho da tela.



A luz passa por um primeiro polarizador, que filtra a luz em um sentido apenas. Depois passa pelo campo com os cristais líquidos, que podem girar o sentido da luz ou deixá-la passar como entrou. Cada ponto individual é controlado por um transistor de filme fino, ou TFT em inglês. A luz então passa por um filtro colorido para gerar a cor de cada ponto e por fim por outro polarizador, que tem sentido defasado de 90° em relação ao primeiro. Caso não haja alimentação nos cristais e consequentemente não haja rotação da luz, ela será bloqueada.

11.2 Sua mensagem no display

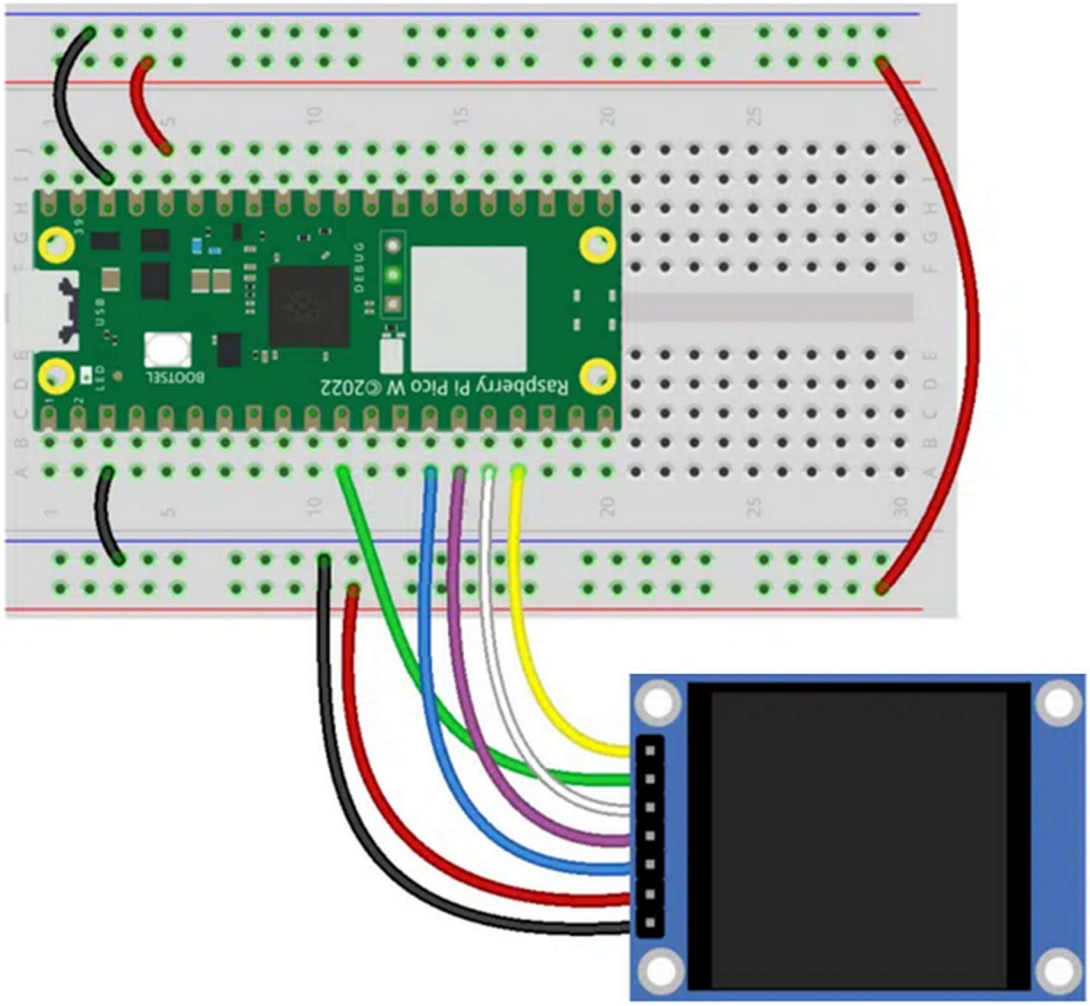
Agora que você já conhece um pouco sobre o princípio do display LCD, vamos usá-lo na prática. Vamos começar com algo simples que é escrever uma frase no display. Por ser algo corriqueiro de se fazer, em geral as funções facilitam bastante o processo. Iremos usar uma biblioteca que irá realizar a comunicação com o display de forma simples para o código final.

Material necessário

- Raspberry Pi Pico W
- Protoboard 400 pontos
- Display IPS LCD 1.3" 240×240
- Jumper Macho-Fêmea
- Cabo USB

Montagem do circuito

Para esta atividade vamos conectar o display à Pico W usando a protoboard conforme a imagem a seguir



O pino GND do display será ligado a um dos pinos GND da Pico W, o pino VCC do display ao pino 3V3 out da placa, o pino SCL do display ao pino GP10, Pino SDA ao pino GP11, Pino RES ao pino GP12, pino DC ao pino 8 e BLK ao pino GP13

Programa

Para o código precisamos carregar as bibliotecas **colour.py** e **LCD.py** para a pasta lib da placa. Caso não se lembre como realizar a instalação, veja novamente os passos da aula 4.4 Como fazer upload de bibliotecas na Pico W.

Agora vamos explicar um pouco das funções utilizadas no código.

A função **fill**() preenche a tela toda com a cor passada como parâmetro.

A biblioteca customizada colour gera o valor RGB565, padrão utilizado pela biblioteca do display, a partir de um trio de valores para R, G e B de 0 a 255.

A função **printstring**() imprime o texto passado, nas coordenadas informadas, com o tamanho informado, recebendo informação para atualizar automaticamente ou não o caracter ou a frase como um todo e a cor que deve mostrar o texto.

Veja o código completo abaixo.

```
# Codigo 11.2 - Sua mensagem no display

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import utime

# Importacao das bibliotecas customizadas

import LCD

from colour import colour

# Declaracao do LCD

display = LCD.LCD_1inch3()


# Preenche o fundo do display com preto e escreve 'Oi, mundo!' em verde partindo das coordenadas 20 px e 10 px


display.fill(colour(0,0,0))

display.printstring('Oi, mundo!',24,30,3,0,0,colour(0,255,0))

display.show()
```

11.3 Montando uma TV

Agora que sabemos o básico da interação com o display, vamos mexer um pouco as coisas. Você vai aprender a criar formas e animações para mostrar virtualmente o que quiser no seu display.

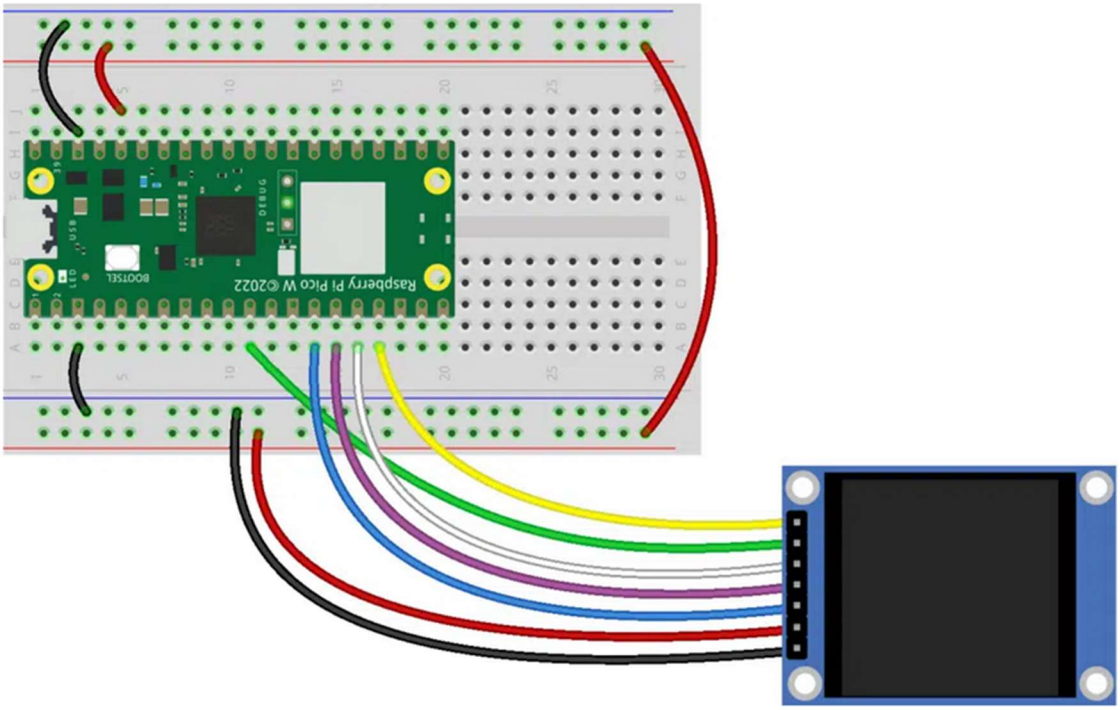
Material necessário

- Raspberry Pi Pico W

- Protoboard 400 pontos
- Display IPS LCD 1.3" 240×240
- Jumpers Macho-Fêmea
- Cabo USB

Montagem do circuito

Para esta atividade vamos conectar o display à Pico W usando a protoboard conforme a imagem a seguir



O pino GND do display será ligado a um dos pinos GND da Pico W, o pino VCC do display ao pino 3V3 out da placa, o pino SCL do display ao pino GP10, Pino SDA ao pino GP11, Pino RES ao pino GP12, pino DC ao pino 8 e BLK ao pino GP13

Programa

Para o código precisamos ter as bibliotecas **colour.py** e **LCD.py** carregadas na pasta lib da placa. Caso não estejam instaladas e não se lembre como realizar a instalação, veja novamente os passos da aula 4.4 Como fazer upload de bibliotecas na Pico W. Após o carregamento, vamos utilizar as funções nativas da biblioteca do LCD para desenhar retângulos (rect()), círculos (ellipse()) e polígonos.(poly()) para animar nossa TV.

A função **rect**() desenha um retângulo a partir das coordenadas informadas, com a altura e largura informadas, da cor informada e se ele deve ser preenchido ou ser apenas um contorno.

Já a função **ellipse**() desenha uma elipse com o centro nas coordenadas informadas, com os semi-eixos x e y informados, na cor informada, se deve ser preenchido ou apenas o contorno e, opcionalmente, quais quadrantes devem ser desenhados.

Por fim, a função **poly**() desenha um polígono com o deslocamento em x e y informado, com vértices nas coordenadas informadas, da cor informada e se ele deve ser preenchido ou ser apenas um contorno.

A função **poly**() recebe como um dos parâmetros um **array** com as coordenadas de cada vértice do polígono. Para isso precisamos importar a biblioteca array. Então declaramos um array da seguinte forma

Você pode conferir o código completo abaixo.

```
# Codigo 11.3 - Montando uma TV

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import utime

import array

import os

import sys

# Importacao das bibliotecas customizadas

import LCD

from colour import colour

# Declaracao do LCD

display = LCD.LCD_1inch3()

# Declaracao dos arrays de coordenadas para o logo da MakerHero


logo_parte_externa = array.array('h',
[40,53,55,40,185,40,200,53,200,187,185,200,55,200,40,187,40,53])
```

```
logo_parte_interna = array.array('h',
[105,75,115,75,115,90,125,90,125,75,135,75,135,90,150,90,150,105,165,
                105,165,115,150,115,150,125,165,125,165,135,150,135,150,1
50,135,150,
                135,165,125,165,125,150,115,150,115,165,105,165,105,150,9
0,150,90,135,
                75,135,75,125,90,125,90,115,75,115,75,105,90,105,90,90,105,
90,105,75])

# Preenche o fundo do display com preto
display.fill(colour(0,0,0))
display.show()
utime.sleep(3)
# Laco de execucao
while True:
  # Desenha as barras coloridas
  display.rect(0,0,35,240,colour(255,255,255),True)
  display.rect(35,0,35,240,colour(255,255,0),True)
  display.rect(70,0,35,240,colour(0,255,255),True)
  display.rect(105,0,35,240,colour(0,255,0),True)
  display.rect(140,0,35,240,colour(255,0,255),True)
  display.rect(175,0,35,240,colour(255,0,0),True)
  display.rect(210,0,35,240,colour(0,0,255),True)
  display.show()
  utime.sleep(3)

  # Desenha o circulo cinza de forma animada fazendo um setor de cada vez

  display.ellipse(120,120,120,120,colour(240,240,240),True,0b0001)
```

```
display.show()
utime.sleep(.1)
display.ellipse(120,120,120,120,colour(240,240,240),True,0b1000)
display.show()
utime.sleep(.1)
display.ellipse(120,120,120,120,colour(240,240,240),True,0b0100)
display.show()
utime.sleep(.1)
display.ellipse(120,120,120,120,colour(240,240,240),True,0b0010)
display.show()
utime.sleep(.1)

# Desenha o logo da MakerHero

display.poly(0,0,logo_parte_externa,colour(0,0,0),True)
display.poly(0,0,logo_parte_interna,colour(255,255,255),True)
display.show()
utime.sleep(5)

# Desenha o smile

display.fill(colour(255,255,255))
display.ellipse(120,120,74,74,colour(245,245,0),True)
display.ellipse(120,120,72,72,colour(250,250,0),True)
display.ellipse(120,120,70,70,colour(255,255,0),True)
display.rect(90,80,10,30,colour(0,0,0),True)
display.rect(140,80,10,30,colour(0,0,0),True)
display.ellipse(120,130,30,30,colour(0,0,0),True,0b1100)
```

```
display.show()
utime.sleep(.5)
# Desenha o smile piscando e mostrando a lingua
display.ellipse(120,120,74,74,colour(245,245,0),True)
display.ellipse(120,120,72,72,colour(250,250,0),True)
display.ellipse(120,120,70,70,colour(255,255,0),True)
display.rect(90,80,10,30,colour(0,0,0),True)
display.rect(140,90,10,10,colour(0,0,0),True)
display.ellipse(120,130,30,30,colour(0,0,0),True,0b1100)
display.rect(110,140,20,20,colour(255,120,120),True)
display.ellipse(120,160,10,10,colour(255,120,120),True,0b1100)
display.show()
utime.sleep(.5)

# Desenha o smile
display.fill(colour(255,255,255))
display.ellipse(120,120,74,74,colour(245,245,0),True)
display.ellipse(120,120,72,72,colour(250,250,0),True)
display.ellipse(120,120,70,70,colour(255,255,0),True)
display.rect(90,80,10,30,colour(0,0,0),True)
display.rect(140,80,10,30,colour(0,0,0),True)
display.ellipse(120,130,30,30,colour(0,0,0),True,0b1100)
display.show()
utime.sleep(3)

# Preenche o fundo do display com preto

display.fill(colour(0,0,0))
```

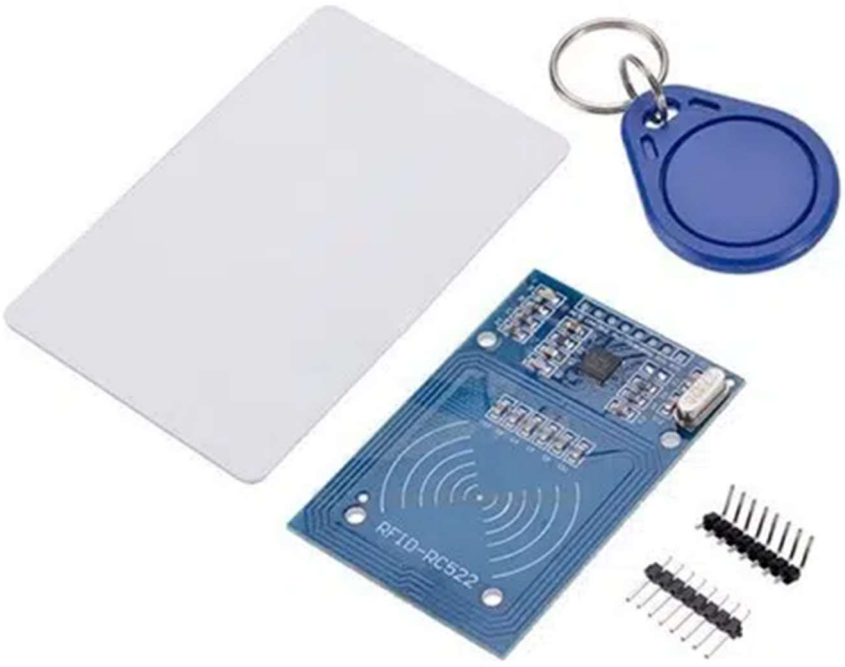
```
display.show()
utime.sleep(1)
```

12. Leitor Rfid

Neste módulo você irá conhecer o Módulo Leitor Rfid e as possibilidades de associar ele a outros componentes para formar sistemas mais completos.

12.1 O que é o Módulo Leitor Rfid?

O kit módulo leitor RFID é baseado no chip MFRC522 da empresa NXP, altamente utilizado em comunicação sem contato a uma frequência de 13,56MHz. Este chip, de baixo consumo e pequeno tamanho, permite por conexão sem contato ler e escrever em cartões que seguem o padrão Mifare, muito usado no mercado.



Este Kit módulo leitor RFID acompanha uma Tag RFID e um Cartão RFID, portanto possui as ferramentas que você precisa para um projeto de controle de acesso ou sistemas de segurança. As tags (ou etiquetas) RFID contém um número de identificação único, como se fosse o CPF do cartão, e também podem ser gravados nelas vários dados sobre o proprietário do cartão, como nome e endereço e, no caso de produtos, informações sobre procedência e data de validade, apenas para citar alguns exemplos.

12.2 Acionando o Leitor Rfid

Como o funcionamento principal do Módulo Leitor Rfid é ler as informações que definimos em cada cartão ou tag, vamos começar aprendendo como fazer essa gravação de dados.

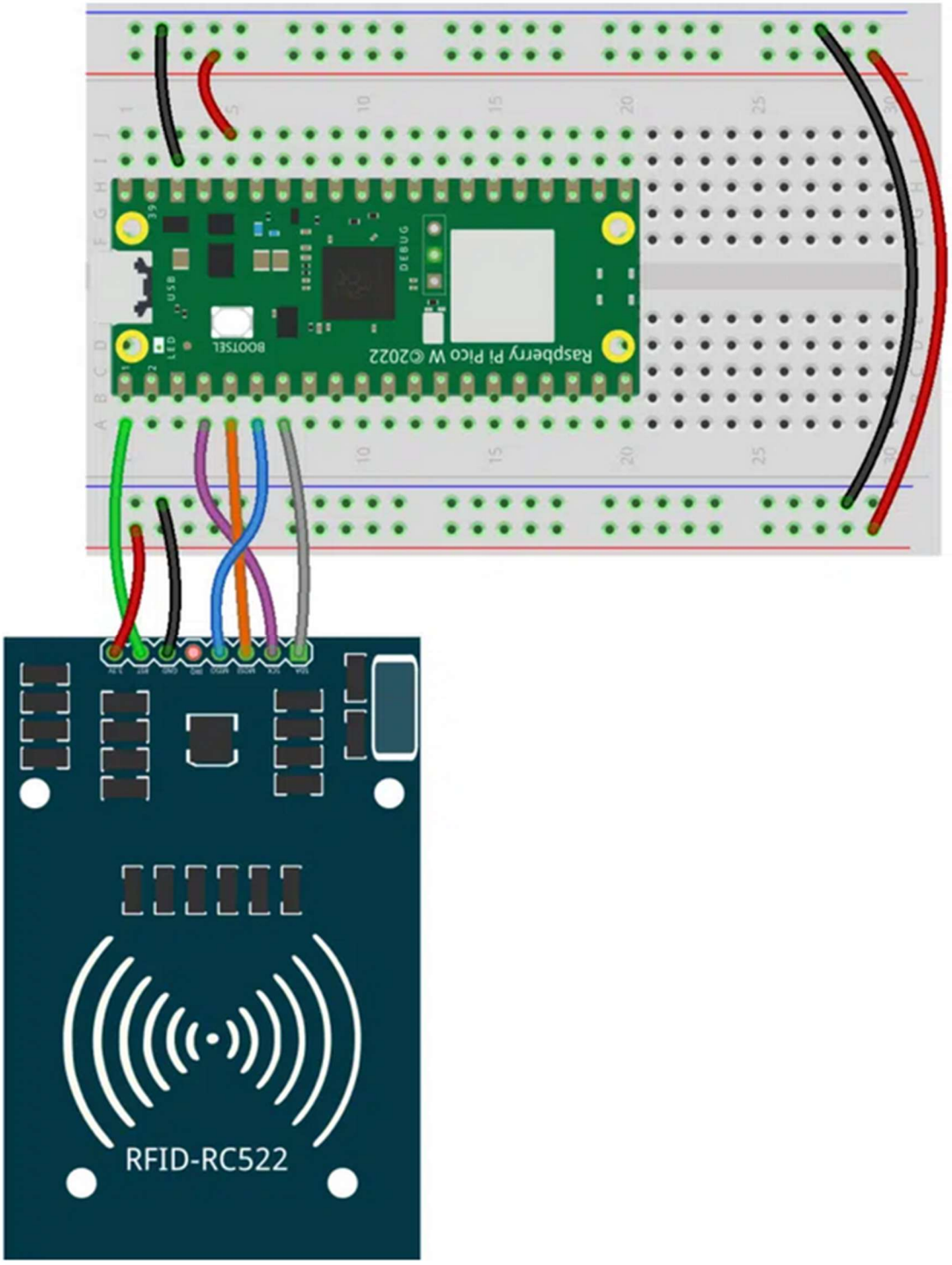
Materiais Necessários

Neste projeto, precisamos dos seguintes componentes:

- Placa Raspberry Pi Pico W
- Cabo USB
- Protoboard 400 pontos
- Jumper macho-macho
- Jumper macho-fêmea
- Módulo Leitor Rfid

Circuito

No circuito deste projeto teremos somente as conexões do Leitor Rfid.



Programa

Aqui você precisa usar a biblioteca **mfrc522**, verifique se ela foi carregada na pasta lib da Pico W, para um tutorial detalhado consulte a aula 4.4 Como fazer upload de bibliotecas na Pico W.

Vamos iniciar o programa como padrão com a chamada da biblioteca. E a identificação dos pinos da Pico W que estão sendo conectados do Módulo Rfid:

```
1from mfrc522 import SimpleMFRC522

2

3reader = SimpleMFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=5,rst=0)
```

Depois vamos criar e utilizar a função **realiza_leitura**() para ler o id do cartão e imprmir esse ID na saída de texto.

```
    def realiza_leitura():

        print("Lendo...Aproxime o cartao...")

        id, text = reader.read()

        print("ID do cartao: %s\nText: %s" % (id,text))
```

Veja o código completo abaixo

```
# Codigo 12.2 - Acionando o leitor RFID


# Bibliotecas necessarias para o codigo


# Importacao das bibliotecas padrao


import utime

# Importacao das bibliotecas customizadas

from mfrc522 import SimpleMFRC522

# Declaracao do leitor RFID

reader = SimpleMFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=5,rst=0)

# Funcao que realiza a leitura do cartao e printa o ID no terminal

def realiza_leitura():


    print("Lendo...Aproxime o cartao...")
```

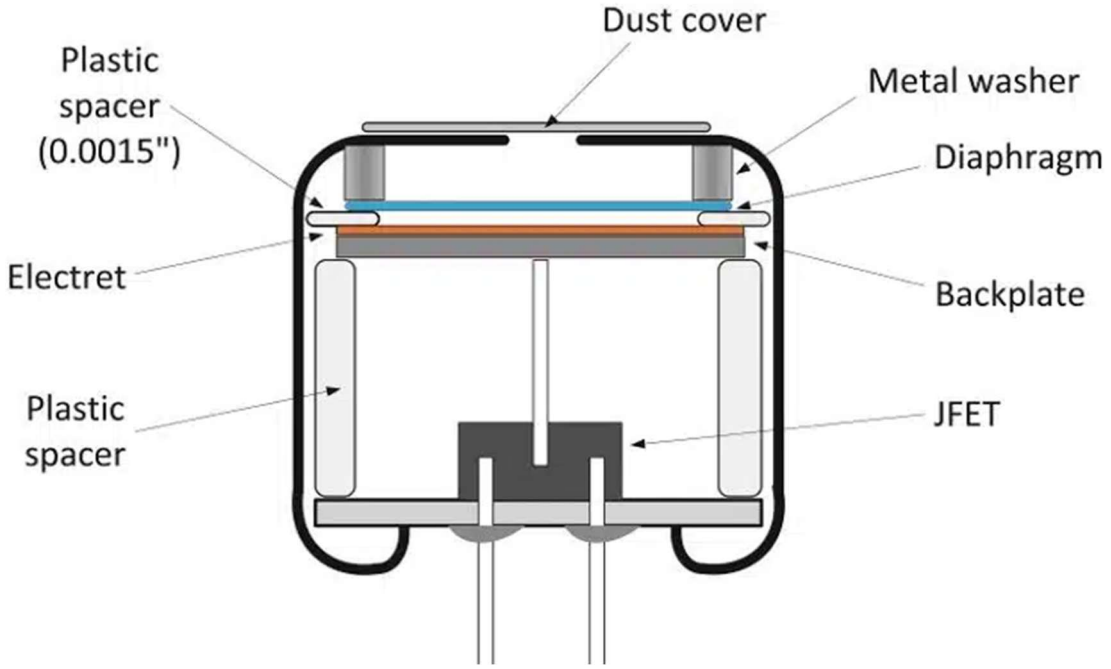
```
    id, text = reader.read()
    print("ID do cartao: %s\nText: %s" % (id,text))
# Laco de execucao
while True:
    realiza_leitura()
    utime.sleep(1)
```

13. Microfone

Neste módulo vamos apresentar o módulo amplificador de microfone e seu conceito de funcionamento.

13.1 O que é o Módulo Amplificador de Microfone?

O módulo amplificador de microfone combina um microfone de eletreto para a captação de sons com um amplificador de sinais MAX4466. O microfone de eletreto funciona criando um capacitor a partir de uma placa de metal fina que pode vibrar com ondas sonoras. Esse capacitor é então ligado a porta (gate) de um transistor mosfet. A variação da capacitância causa uma variação na porta (gate). Essa variação por sua vez varia a resistência entre dreno (drain) e fonte (source) do mosfet. Ao passar uma corrente pelo mosfet, é possível detectar a variação de resistência e consequentemente, detectar o sinal de som eletricamente.



Esse sinal possui uma amplitude bastante reduzida e o amplificador serve para elevar o sinal para um nível em que microcontroladores e outros sistemas possam realizar a leitura mais facilmente. Quanto mais próximo do microfone o amplificador estiver, menos sujeito a ruídos o sinal estará. Assim, o módulo amplificador de microfone faz a captação e amplificação em um único módulo compacto.

Pinagem do módulo

O módulo possui apenas 3 pinos, VCC, GND e Out. VCC e GND recebem as alimentações positiva e negativa, enquanto Out é a saída do sinal já amplificado. Devemos conectar o sinal de saída a uma porta analógica para a leitura correta.

13.2 Acendendo uma luminária com sons

Vamos aproveitar que conseguimos captar sons para fazer um projeto bastante comum mas muito interessante, que é comandar um dispositivo por som.

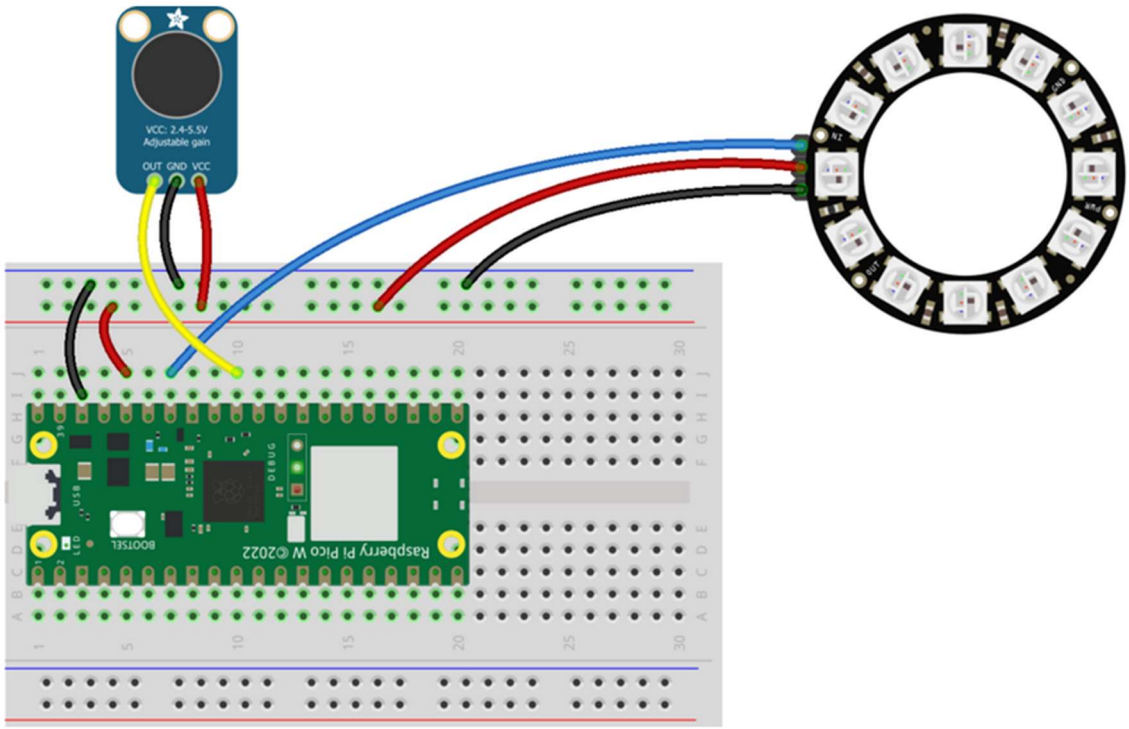
Vamos utilizar palmas, estalos ou outros sons de maior intensidade para acender ou apagar nossa luminária.

Material necessário

- Raspberry Pi Pico W
- Protoboard 400 pontos
- Jumper Macho-Fêmea
- Módulo amplificador de microfone
- Anel de LEDs
- Cabo USB

Montagem do circuito

Para esta atividade vamos conectar o Anel de LEDs à Pico W usando a protoboard conforme a imagem a seguir



Programa

Não teremos nenhuma novidade de funções ou estruturas para esse programa. Veja o código completo abaixo.

Codigo 13.2 - Acendendo uma luminaria com sons

```
# Bibliotecas necessarias para o codigo
# Importacao das bibliotecas padrao
import machine
import utime
# Importacao das bibliotecas customizadas
from ws2812 import WS2812
# Declaracao do Anel de LEDs
ws = WS2812(machine.Pin(28), 12)
# Declaracao do microfone
mic = machine.ADC(26)
# Declaracao das variaveis
ultima_palma = 0
uma_palma = False
duas_palmas = False

# Le uma media de 100 amostras do ruido ambiente e faz uma media
ambient_noise = sum(mic.read_u16() for _ in range(100)) / 100
# Laco de execucao
while True:
  if ultima_palma == 0:
    ultima_palma = utime.ticks_ms()
  if mic.read_u16() > ambient_noise * 1.8:
    print('uma palma')
    if (utime.ticks_ms() - ultima_palma < 400):
      duas_palmas = True
      print('duas palmas')

    else:
```

```
            uma_palma = True
        ultima_palma = utime.ticks_ms()


    if duas_palmas:
        ws.write_all(0xFFFFFF)
        duas_palmas = False
        uma_palma = False
    elif uma_palma:
        ws.write_all(0x000000)
        uma_palma = False
        utime.sleep(0.1)
```

13.3 Reagindo aos sons

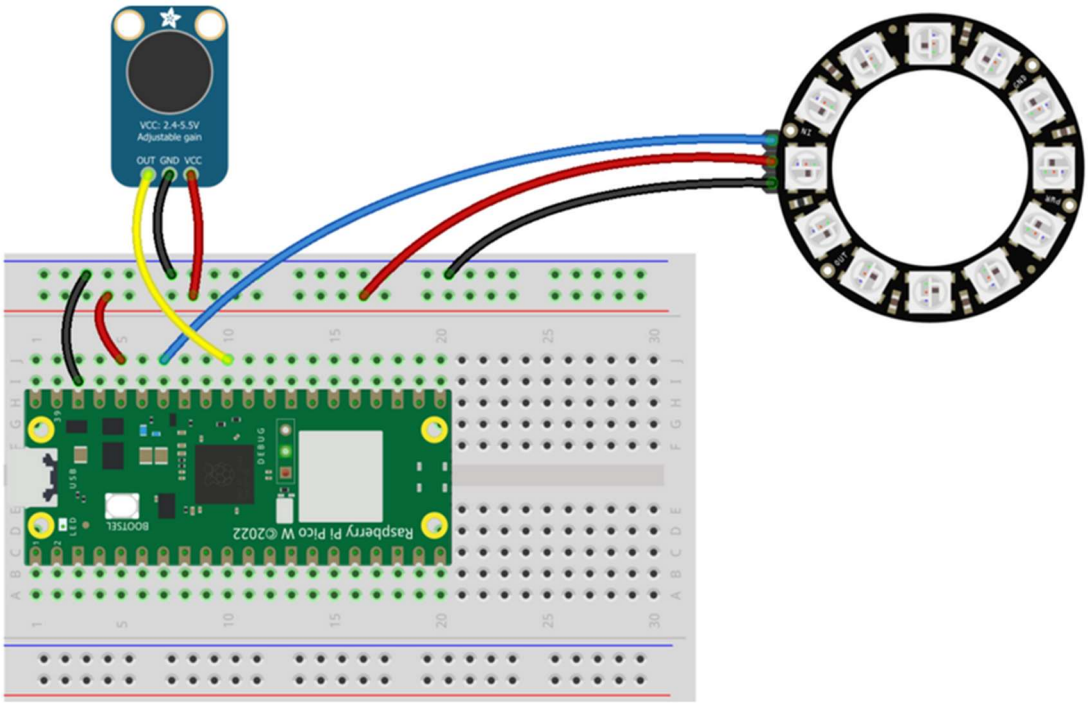
Fizemos a luminária acender e apagar por som, mas não temos nenhuma informação de intensidade, apenas de que foi mais alto que um certo limiar e que se repetiu rapidamente. Mas que tal agora fazer com que diferentes níveis de som variem o brilho e a cor dos LEDs?

Material necessário

- Raspberry Pi Pico W
- Protoboard 400 pontos
- Jumper Macho-Fêmea
- Módulo amplificador de microfone
- Anel de LEDs
- Cabo USB

Montagem do circuito

Para esta atividade vamos conectar o Anel de LEDs à Pico W usando a protoboard conforme a imagem a seguir



Programa

Veja o código completo abaixo

```
# Codigo 13.3 - Reagindo a sons

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import machine

import utime

# Importacao das bibliotecas customizadas

from ws2812 import WS2812

# Declaracao do microfone

mic = machine.ADC(26)


# Declaracao da fita de LEDs
```

```
ws = WS2812(machine.Pin(28), 12)
# Captacao do ruido ambiente
ruido_ambiente = []
for i in range(100):
  ruido_ambiente.append(mic.read_u16())
  utime.sleep(0.001)
# Laco de execucao
while True:
  leitura = mic.read_u16()
  if leitura > max(ruido_ambiente) or leitura < min(ruido_ambiente):
    leitura = leitura*(4096)/((65535)+4096)
    leitura = round(leitura)
    brilho = abs((leitura-2048)/2048)
    brilho = round(brilho,2)
    if brilho < 0.33:
      ws.write_all([0,0,round(255*brilho)])
    elif brilho >= 0.33 and brilho < 0.66:
      ws.write_all([0,round(255*brilho),0])
    else:
      ws.write_all([round(255*brilho),0,0])
  else:
    ws.write_all([0,0,0])
  utime.sleep(.05)
```

14. Projetos para consolidar conhecimentos

Que tal fazer um monitor do ambiente, mostrando no display a temperatura, umidade e a luminosidade relativas? Para isso iremos utilizar novamente o LDR, o DHT11 e o display LCD.

Material Necessário

Neste projeto, precisamos dos seguintes componentes:

- Raspberry Pi Pico W
- Protoboard 400 pontos
- Jumpers Macho-Macho
- Sensor DHT11
- Display LCD
- Sensor LDR
- Resistor 10k
- Cabo USB

Montagem do circuito

Nesse projeto faremos as seguintes ligações

Sensor DHT 11

- VCC > Linha positiva da Protoboard
- OUT > GP16
- GND > Linha negativa da Protoboard

Resistor 10k e LDR

Um pino do resistor a linha positiva da Protoboard e o outro a um dos pinos do LDR

O pino do LDR conectado ao resistor de 10k também deve ser ligado ao pino GP26 da Pico W. O outro pino do LDR deve ser conectado a linha negativa da protoboard

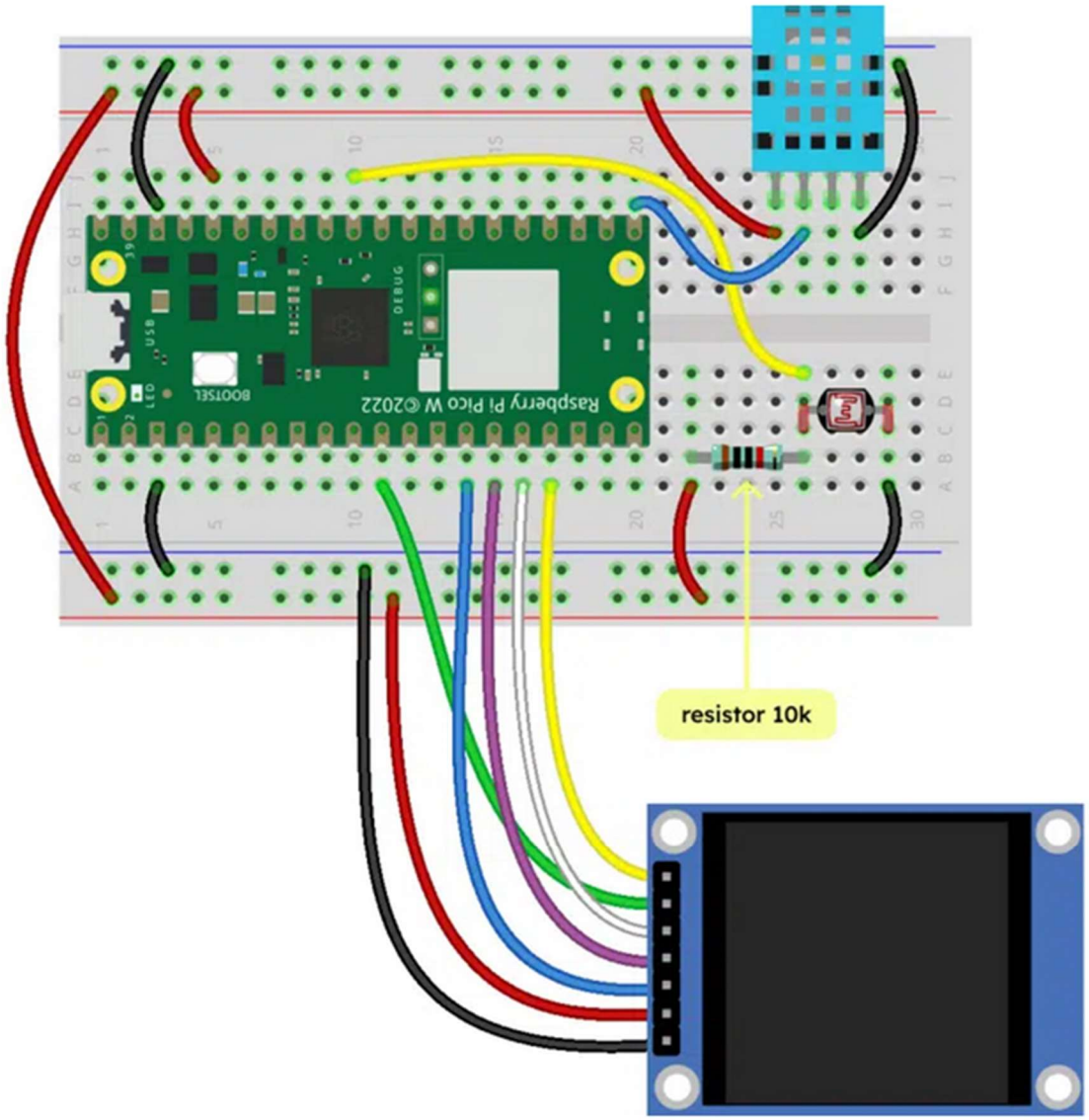
Display LCD

Os pinos do display LCD devem ser ligados da seguinte forma

- VCC > Linha positiva da Protoboard

- GND > Linha negativa da protoboard
- SCL > GP10
- SDA > GP11
- RES > GP12
- DC > GP8
- BLK > GP13

Veja o esquemático completo abaixo



Programa

Neste programa não teremos nenhuma novidade em termos de funções e estruturas.

Confira se as bibliotecas **colour.py**, **dht.py** e **LCD.py** estão instaladas na pasta lib da Pico W. Caso não estejam e precise de um lembrete de como instalar, confira a aula 4.4 Como fazer upload de bibliotecas na Pico W.

Com as bibliotecas instaladas, confira o código completo abaixo

```
# Codigo 14.1 - Monitoramento do ambiente

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

from machine import Pin, I2C

import utime

import framebuf

# Importacao das bibliotecas customizadas

from dht import DHT11

import LCD

from colour import colour

# Declaracao do LCD

LCD = LCD.LCD_1inch3()

# Declaracao do sensor de temperatura e umidade

pin = Pin(16, Pin.IN, Pin.PULL_UP)

sensor = DHT11(pin)

# Declaracao do foto resistor

foto_resistor = machine.ADC(26)

valor_de_baixa_luminosidade=65535

valor_de_alta_luminosidade=0
```

```
utime.sleep(5)  # Atraso para a inicializacao correta do sensor
# Laco de execucao
while True:
  valor_luminosidade  = foto_resistor.read_u16()
  try:
    sensor.measure()
    temperatura = sensor.temperature()
    umidade = sensor.humidity()
  except:
    temperatura = 'err'
    umidade = 'err'
  porcentagem_iluminacao = ((valor_de_baixa_luminosidade-
valor_luminosidade)/valor_de_baixa_luminosidade)
  string_temp = "Temp: {} C".format(temperatura)
  string_umid = "Umidade: {}%".format(umidade)
  string_lumi = "Luz: {}%".format(round(porcentagem_iluminacao,3)*100)
  LCD.fill(colour(40,40,40))
  LCD.show()
  LCD.printstring(string_temp,17,30,3,0,0,colour(244,255,113))
  LCD.printstring(string_umid,17,80,3,0,0,colour(244,255,113))
  LCD.printstring(string_lumi,17,120,3,0,0,colour(244,255,133))
  LCD.show()
  utime.sleep(4)
```

14.2 Sirene

Já imaginou como são os sistemas que acionam as luzes dos carros de polícia e bombeiros, com todos os seus modos, luzes e sons? Vamos fazer um sistema parecido com esses nesse projeto. Acionaremos o Anel de LEDs simulando um

giroflex, ou lanternas especiais de emergência, que é o termo técnico segundo a autoridade de trânsito brasileira, e o buzzer de forma a simular um sistema de alerta de um veículo de emergência.

Material Necessário

- Protoboard 400 pontos
- Placa Raspberry Pi Pico W
- Jumper Macho-Macho
- Buzzer
- Resistor 220ohms
- Anel de LEDs RGB
- Cabo USB

Montagem do circuito

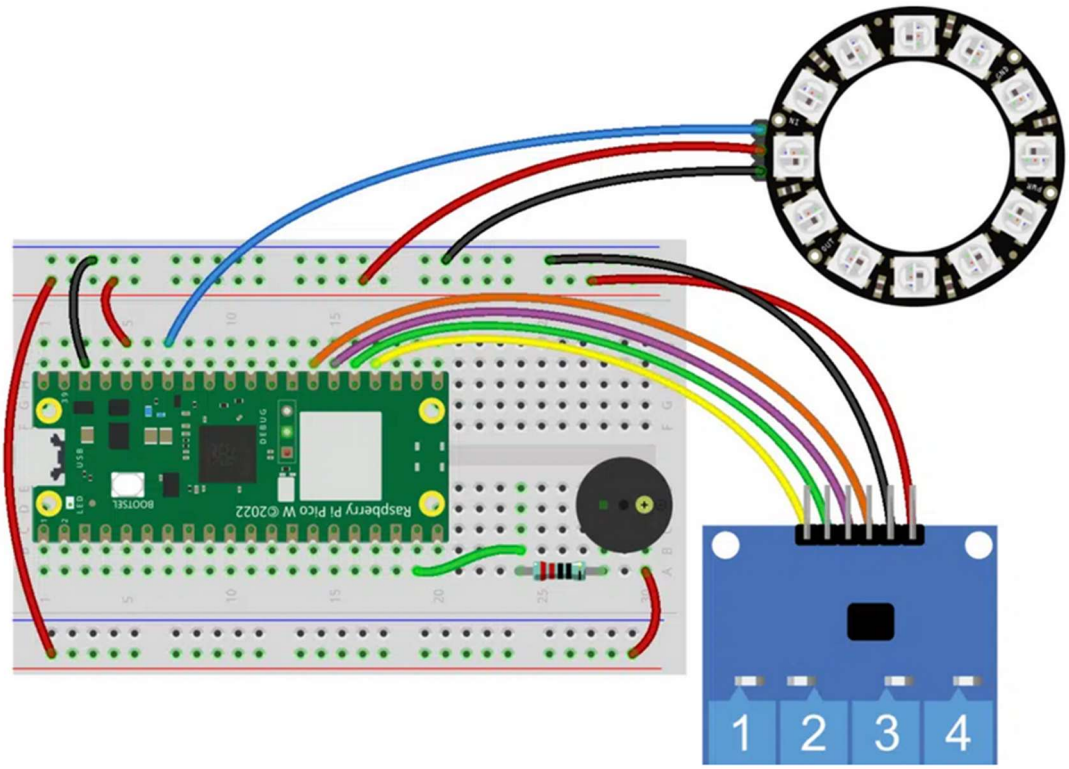
Para montar o circuito iremos seguir a montagem da aula 6.3, teclado de cores, acrescentando o buzzer e o resistor de 220 ohms da seguinte forma

Pino + no buzzer na linha positiva da protoboard

Pino – do buzzer em um dos pinos do resistor de 220 ohms

O outro pino do resistor no pino GP14 da Pico W

Veja o esquemático completo abaixo



Programa

Para esse programa vamos aproveitar os dois núcleos de processamento da Pico W, e você vai aprender a usar a biblioteca _thread e aproveitar o máximo de processamento da sua placa

Primeiro importamos a biblioteca _**threads** como já aprendemos a fazer com outras bibliotecas. Então, para utilizar o segundo núcleo nós definimos a função que irá rodar com qualquer outra função.

Agora para iniciar essa função em outro núcleo utilizamos a estrutura a seguir.

_thread.start_new_thread(nucleo_secundario,())

Essa função inicia a função passada entre parênteses no núcleo secundário. o segundo parênteses após a vírgula é para passagem de parâmetros, mas serve para funções mais avançadas que não iremos utilizar.

Precisamos conferir também se a biblioteca **ws2812.py** está instalada na pasta lib. Caso precise de um lembrete de como instalar bibliotecas, consulte a aula 4.4 Como fazer upload de bibliotecas na Pico W.

Vamos usar o botão 1 para ligar a luz vermelha e azul, o botão 2 para ligar a luz vermelha e azul com o efeito estroboscópico intercalado, o 3 para ligar a sirene e o 4 para desligar tudo de uma vez.

Veja o código completo abaixo

```
# Codigo 14.2 - Sirene
# Bibliotecas necessarias para o codigo
# Importacao das bibliotecas padrao
import machine
import utime
import _thread
# Importacao das bibliotecas customizadas
from ws2812 import WS2812
# Declaracao do anel de LEDs
ws = WS2812(machine.Pin(28), 12)
# Declaracao do buzzer
buzzer = machine.PWM(machine.Pin(14, machine.Pin.OUT))
# Declaracao dos botoes
botao_1 = machine.Pin(18, machine.Pin.IN)
botao_2 = machine.Pin(19, machine.Pin.IN)
botao_3 = machine.Pin(20, machine.Pin.IN)
botao_4 = machine.Pin(21, machine.Pin.IN)
# Declaracao das variaveis
estado_botao_1 = 0
estado_botao_2 = 0
estado_botao_3 = 0
estado_botao_4 = 0
giroflex_simples_ligado = False
giroflex_com_estrobo_ligado = False
sirene_ligada = False


# Funcao que comanda o anel de LEDs em animacao de giroflex simples
```

```
# e o buzzer (se estiver ativa a sirene)
def giroflex_simples():
  global giroflex_simples_ligado
  j=0
  tempo_desde_ultima_mudanca = utime.ticks_ms()
  buzzer.freq(500)
  proximoMultiplicador = 3
  while(giroflex_simples_ligado):
    if sirene_ligada:
      duty = 32767
    else:
      duty = 65535
    buzzer.duty_u16(duty)
    for i in range(12):
      if ((i+j)%12)//6 < 1:
        ws.write(i,0xFF0000)
      elif ((i+j)%12)//6 == 1:
        ws.write(i,0X0000FF)
    if ((utime.ticks_ms() - tempo_desde_ultima_mudanca > 300)):
      buzzer.freq(500*proximoMultiplicador)
      if proximoMultiplicador == 3:
        proximoMultiplicador = 1
      else:
        proximoMultiplicador = 3
      tempo_desde_ultima_mudanca = utime.ticks_ms()

    if j == 2000000000:
```

```
        j=0
      j+=1
      utime.sleep_ms(20)
  ws.write_all(0x000000)
  buzzer.duty_u16(65535)
# Funcao que comanda o anel de LEDs em animacao de giroflex com efeito estroboscopico
# e o buzzer (se estiver ativa a sirene)
def giroflex_com_estrobo():
  global giroflex_com_estrobo_ligado
  j=0
  tempo_desde_ultima_mudanca = utime.ticks_ms()
  buzzer.freq(500)
  proximoMultiplicador = 3
  while(giroflex_com_estrobo_ligado):
    if sirene_ligada:
      duty = 32767
    else:
      duty = 65535
    buzzer.duty_u16(duty)
    for i in range(12):
      if ((i+j)%12)//6 < 1:
        ws.write(i,0xFF0000)
      elif ((i+j)%12)//6 == 1:
        ws.write(i,0X0000FF)
    if ((utime.ticks_ms() - tempo_desde_ultima_mudanca > 300)):
      buzzer.freq(500*proximoMultiplicador)
```

```
    if proximoMultiplicador == 3:
        proximoMultiplicador = 1
    else:
        proximoMultiplicador = 3
    tempo_desde_ultima_mudanca = utime.ticks_ms()
if j%59 == 0:
    ws.write_all(0xFFFFFF)
    buzzer.freq(500)
    utime.sleep_ms(40)
    ws.write_all(0xFF0000)
    buzzer.freq(100)
    utime.sleep_ms(40)
    ws.write_all(0xFFFFFF)
    buzzer.freq(500)
    utime.sleep_ms(40)
    ws.write_all(0x0000FF)
    buzzer.freq(100)
    utime.sleep_ms(40)
    ws.write_all(0xFFFFFF)
    buzzer.freq(500)
    utime.sleep_ms(40)
    ws.write_all(0xFF0000)
    buzzer.freq(100)
    utime.sleep_ms(40)
    ws.write_all(0xFFFFFF)
    buzzer.freq(500)
    utime.sleep_ms(40)
```

```
            ws.write_all(0x0000FF)
            buzzer.freq(100)
            utime.sleep_ms(40)
            buzzer.freq(500*proximoMultiplicador)
        if j == 2000000000:
            j=0
        j+=1
        utime.sleep_ms(20)
    ws.write_all(0x000000)
    buzzer.duty_u16(65535)
# Funcao que comanda o buzzer caso os giroflex nao estejam ligados em simultaneo
def sirene():
    if sirene_ligada:
        duty = 32767
    else:
        duty = 65535
    buzzer.duty_u16(duty)
    while sirene_ligada and not giroflex_com_estrobo_ligado and not giroflex_simples_ligado:
            buzzer.freq(500)
            utime.sleep_ms(40)
            buzzer.freq(100)
            utime.sleep_ms(40)
            buzzer.freq(500)
            utime.sleep_ms(40)
            buzzer.freq(100)
```

```
        utime.sleep_ms(40)
        buzzer.freq(500)
        utime.sleep_ms(40)
        buzzer.freq(100)
        utime.sleep_ms(40)
        buzzer.freq(500)
        utime.sleep_ms(40)
        buzzer.freq(100)
        utime.sleep_ms(40)
    buzzer.duty_u16(65535)
# Funcao que sera executada no segundo nucleo da pico W em paralelo a do nucelo principal
def nucleo_secundario():
    # Laco de execucao do nucleo secundario
    while True:
        giroflex_simples()
        giroflex_com_estrobo()
        sirene()
# Inicializacao da nova rotina no nucelo secundario e espera para que ela inicie com sucesso
_thread.start_new_thread(nucleo_secundario,())
utime.sleep(0.25)
# Laco de execucao
while True:
    estado_botao_1 = botao_1.value()
    estado_botao_2 = botao_2.value()
    estado_botao_3 = botao_3.value()
```

```
estado_botao_4 = botao_4.value()
if estado_botao_1 == 1:
    if giroflex_simples_ligado:
        giroflex_simples_ligado = False
    elif giroflex_com_estrobo_ligado:
        giroflex_com_estrobo_ligado = False
        giroflex_simples_ligado = True
    else:
        giroflex_simples_ligado = True
elif estado_botao_2 == 1:
    if giroflex_com_estrobo_ligado:
        giroflex_com_estrobo_ligado = False
    elif giroflex_simples_ligado:
        giroflex_com_estrobo_ligado = True
        giroflex_simples_ligado = False
    else:
        giroflex_com_estrobo_ligado = True
elif estado_botao_3 == 1:
    sirene_ligada = not sirene_ligada
elif estado_botao_4 == 1:
    giroflex_com_estrobo_ligado = False
    giroflex_simples_ligado = False
    sirene_ligada = False
utime.sleep(0.2)
```

14.3 Contagem de passagem

E que tal monitorar quantas pessoas ou animais passam por um determinado lugar? Vamos utilizar o sensor de movimento PIR para isso e mostrar a contagem no Display

Material Necessário

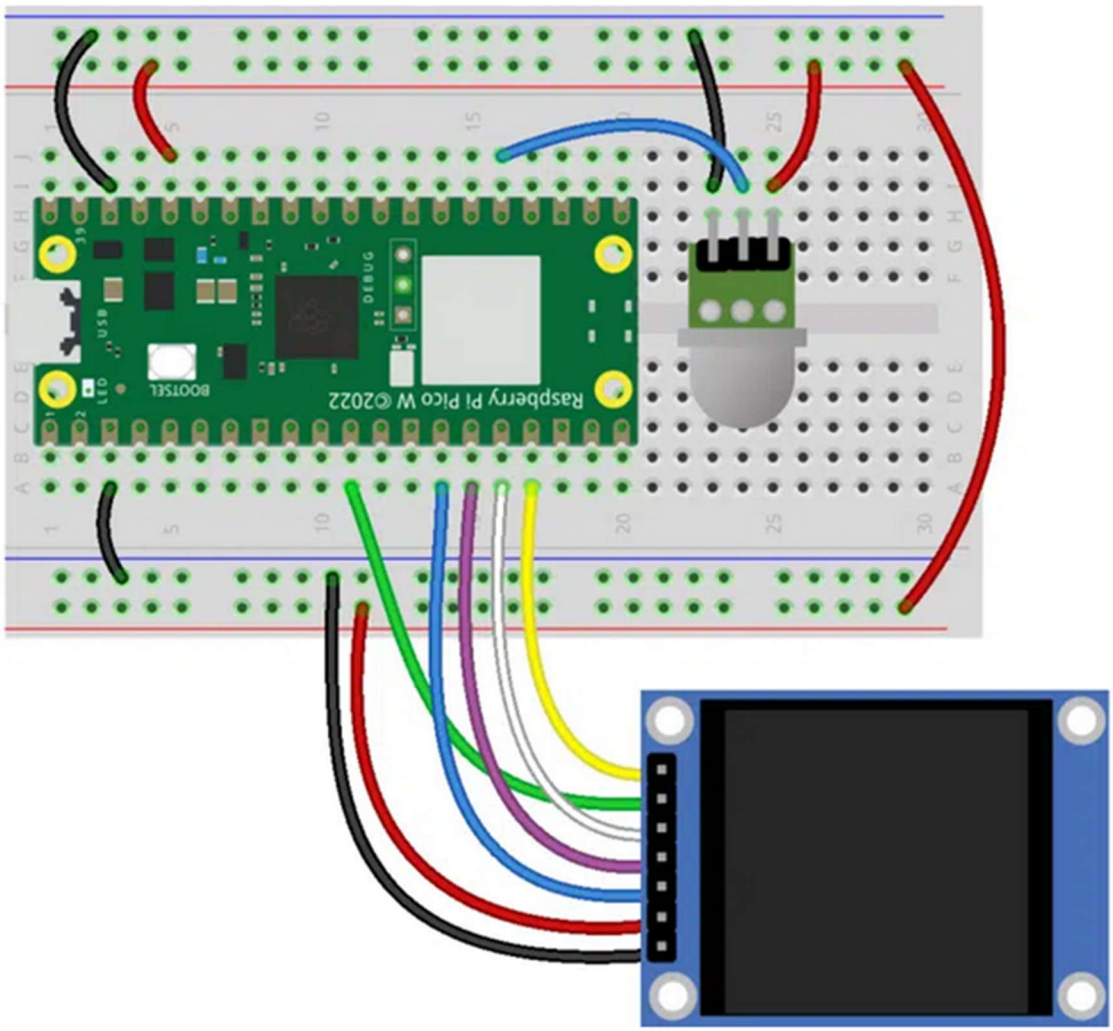
- Raspberry Pi Pico W
- Protoboard 400 pontos
- Jumpers Macho-Macho
- Jumpers Macho-Femea
- Sensor PIR
- Display IPS LCD 1.3" 240×240
- Cabo USB

Montagem do circuito

A montagem do sensor PIR será igual à da aula 7.2, mas não iremos colocar o LED dessa vez.

O display seguirá a mesma montagem da aula 11.2.

Veja o esquemático completo abaixo



Programa

Não haverá nenhuma novidade nesse projeto. Confira se as bibliotecas **colour.py** e **LCD.py** estão instaladas. Caso precise de um lembrete, veja a aula 4.4 Como fazer upload de bibliotecas na Pico W.

Veja o código completo abaixo

```
# Codigo 14.3 - Contagem de passagem

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import machine

import utime
```

```
# Importacao das bibliotecas customizadas
import LCD
from colour import colour
# Declaracao do LCD
display = LCD.LCD_1inch3()
# Declaracao do sensor de movimento PIR
pir_sensor = machine.Pin(19, machine.Pin.IN)
# Declaracao das variaveis
numero_de_movimentacoes = 0
atualiza_display = False
# Funcao chamada ao concluir a deteccao de uma movimentacao pelo sensor
def movimento_detectado(_):
  global numero_de_movimentacoes
  global atualiza_display
  numero_de_movimentacoes += 1
  atualiza_display = True
# Declaracao da interrupcao que sera iniciadas ao encerrar a deteccao de uma movimentacao
# pelo sensor
pir_sensor.irq(trigger=machine.Pin.IRQ_FALLING, handler=movimento_detectado)
# Exibicao das informacoes que permanecerao fixas no display
display.fill(colour(0,0,0))
display.printstring('Contagem de',22,30,3,0,0,colour(244,255,113))
display.printstring('pessoas',60,70,3,0,0,colour(244,255,113))
display.show()
atualiza_display = True
# Laco de execucao
```

```
while True:
  if atualiza_display:
    display.fill_rect(100,120,240,40,colour(0,0,0))
    if numero_de_movimentacoes < 10:
      display.printstring('{}'.format(numero_de_movimentacoes),112,120,3,1,0,colour(244,255,113))
    elif numero_de_movimentacoes >= 10 and numero_de_movimentacoes < 100:
      display.printstring('{}'.format(numero_de_movimentacoes),105,120,3,1,0,colour(244,255,113))
    elif numero_de_movimentacoes > 10 and numero_de_movimentacoes < 1000:
      display.printstring('{}'.format(numero_de_movimentacoes),95,120,3,1,0,colour(244,255,113))
    else:
      display.printstring('{}'.format(numero_de_movimentacoes),85,120,3,1,0,colour(244,255,113))
    atualiza_display = False
  utime.sleep(0.1)
```

14.4 Jukebox

Fizemos alguns sons e alertas com o buzzer, mas se quisermos ir mais além e tocar músicas mais complexas. Vamos montar uma jukebox, em que você aperta um dos botões e uma música irá tocar. Você terá quatro músicas a disposição e poderá mudar a programação quantas vezes quiser para variar as músicas.

Material Necessário

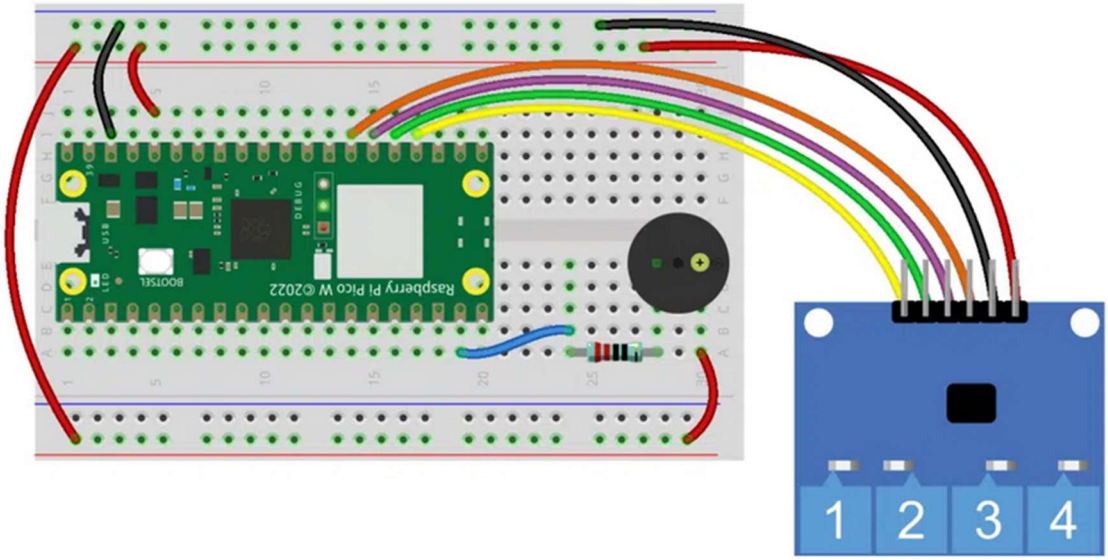
- Raspberry Pi Pico W
- Protoboard 400 pontos
- Jumpers Macho-Macho
- Jumpers Macho-Femea

- Sensor de Toque Capacitivo TTP224
- Buzzer
- Resistor 220ohms
- Cabo USB

Montagem do circuito

A montagem será a mesma da aula 10.3 – Fazendo Música

Veja o esquemático completo abaixo



Programa

Para esse projeto precisamos da biblioteca **melodias.py**, que é onde estão salvas as notas e durações para as músicas que podemos tocar na jukebox.

Para tocar as músicas vamos usar a função **play**() da biblioteca, que recebe como parâmetros o pino onde o buzzer está ligado e a música que será tocada. A lista de músicas está dentro do próprio arquivo da biblioteca e você deve escrever da mesma forma que está na biblioteca.

Além disso, não teremos novidades de estruturas. Veja o código completo abaixo

Codigo 14.4 - Jukebox

```
# Bibliotecas necessarias para o codigo
# Importacao das bibliotecas padrao
from machine import Pin,PWM
import utime
# Importacao das bibliotecas customizadas
from melodias import *
# Declaracao do buzzer
buzzer = PWM(Pin(14))
buzzer.freq(1000)
buzzer.duty_u16(65535)
# Declaracao dos botoes
botao_1 = machine.Pin(18, machine.Pin.IN)
botao_2 = machine.Pin(19, machine.Pin.IN)
botao_3 = machine.Pin(20, machine.Pin.IN)
botao_4 = machine.Pin(21, machine.Pin.IN)
# Declaracao das variaveis
tocar_musica = False
estado_botao_1 = 0
estado_botao_2 = 0
estado_botao_3 = 0
estado_botao_4 = 0
# Laco de execucao
while True:
    estado_botao_1 = botao_1.value()
    estado_botao_2 = botao_2.value()
    estado_botao_3 = botao_3.value()
    estado_botao_4 = botao_4.value()
```

```
if estado_botao_1 == 1:
    play(buzzer,musicas["pinkpanther"])
elif estado_botao_2 == 1:
    play(buzzer,musicas["nevergonnagiveyouup"])
elif estado_botao_3 == 1:
    play(buzzer,musicas["starwars"])
elif estado_botao_4 == 1:
    play(buzzer,musicas["thelionesleepstonight"])
```

14.5 Controle de acesso

Talvez você more, já tenha visitado alguém que mora em um condomínio ou acessado algum estabelecimento com portaria eletrônica ou controle de acesso. Que tal entender um pouco da lógica de funcionamento por trás desses sistemas montando o seu próprio?

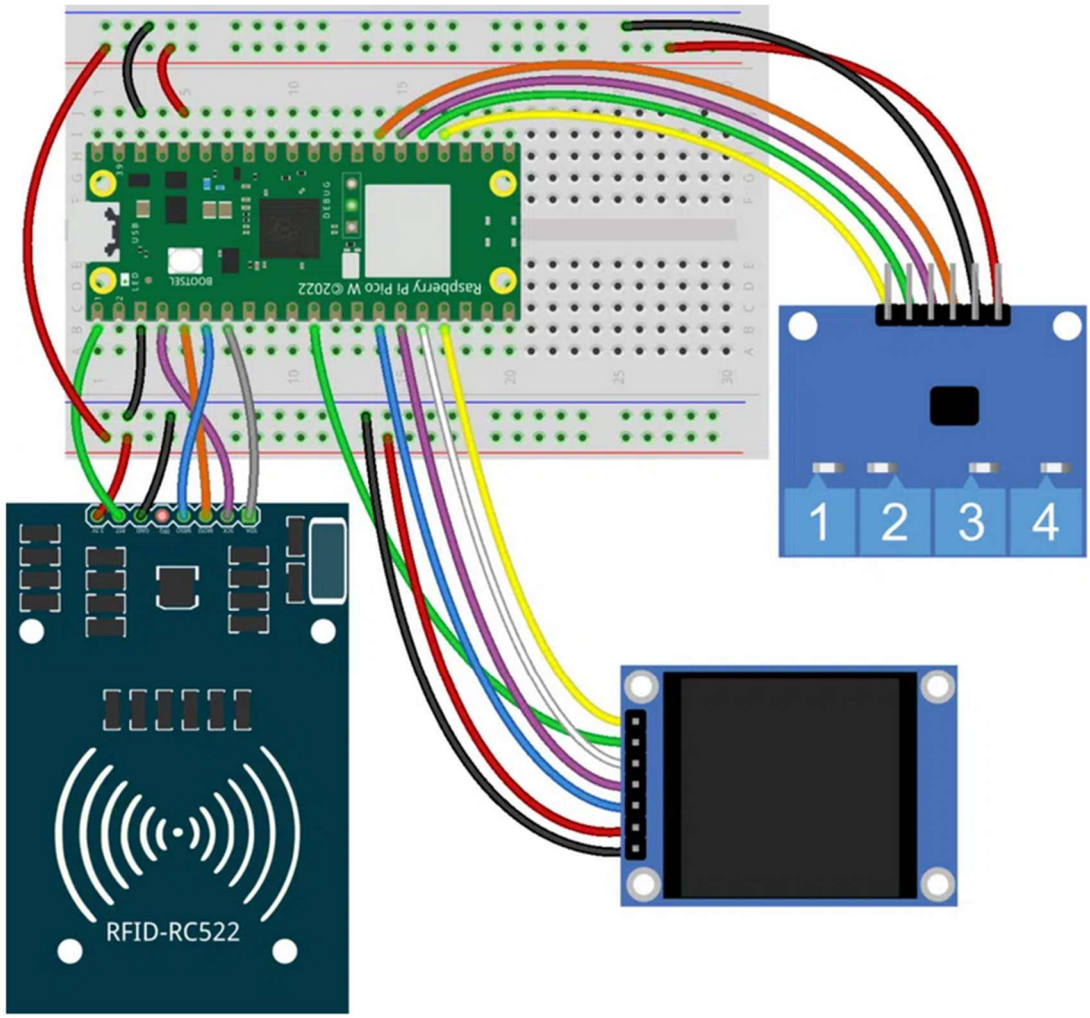
Vamos utilizar o leitor RFID para ler o cartão ou tag para liberar acesso, o sensor touch para inserir a senha de cadastro de cartão ou a senha de acesso e o display para mostrar as informações, liberação ou impedimento de acesso.

Material Necessário

- Raspberry Pi Pico W
- Protoboard 400 pontos
- Jumpers Macho-Macho
- Jumpers Macho-Femea
- Sensor de Toque Capacitivo TTP224
- Display IPS LCD 1.3" 240×240
- Módulo Leitor Rfid
- Cabo USB

Montagem do circuito

Vamos montar os componentes na placa nas mesmas posições que montamos em projetos anteriores, veja o esquemático completo abaixo



Programa

Precisamos das bibliotecas **colour.py**, **LCD.py**, e **mfrc522** instaladas na pasta lib da Pico W. Confira se estão instaladas. Se não estiverem, faça a instalação. Você pode revisar como é o procedimento na aula 4.4 Como fazer upload de bibliotecas na Pico W.

O controle de acesso terá duas senhas que você configura na programação, uma para a gravação do cartão e uma para liberação de acesso. Você deve digitar a senha de gravação do cartão para entrar no modo de gravação, que será indicado

na tela e então aproximar o cartão do leitor para a memorização. Feita a memorização, o cartão liberará o acesso. Ao digitar a senha de acesso você também terá a mensagem de acesso liberado.

O código de exemplo permite apenas um cartão memorizado por vez.

Apesar de grande, o código não traz nenhuma novidade em termos de funções e blocos. Confira o código completo abaixo:

```
# Codigo 14.5 - Controle de acesso

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import machine

import utime

import _thread

# Importacao das bibliotecas customizadas

from mfrc522 import SimpleMFRC522

import LCD

from colour import colour

# Declaracao do LCD

display = LCD.LCD_1inch3()

# Declaracao do leitor RFID

reader = SimpleMFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=5,rst=0)

# Declaracao dos botoes

botao_1 = machine.Pin(18, machine.Pin.IN)

botao_2 = machine.Pin(19, machine.Pin.IN)

botao_3 = machine.Pin(20, machine.Pin.IN)

botao_4 = machine.Pin(21, machine.Pin.IN)

# Declaracao das variaveis

entrando_senha = False

contador_digitos_inseridos = 0
```

```
contador_digitos_anterior = 0
tecla_pressionada = 0
tempo_ultimo_toque = 0
tempo_ultima_liberacao = 0
 tempo_ultima_leitura = 0
tempo_ultima_atualizacao_display = 0
inicia_cadastro = False
cartao_liberado = [-1]
cartao_lido = [-1]
senha_cadastro = [1,2,3,4]
senha_acesso = [4,3,2,1]
senha_digitada = ['','','','']

# Funcao chamada ao apertar o botao 1
def button_1_isr(pin):
  global tempo_ultimo_toque
  if not utime.ticks_ms() - tempo_ultimo_toque < 200:
    global entrando_senha
    if entrando_senha:
      global tecla_pressionada
      tecla_pressionada = 1
      global contador_digitos_inseridos
      contador_digitos_inseridos += 1
    else:
      entrando_senha = True
      tecla_pressionada = 1
      contador_digitos_inseridos += 1
```

```
        tempo_ultimo_toque = utime.ticks_ms()
# Funcao chamada ao apertar o botao 2
def button_2_isr(pin):
    global tempo_ultimo_toque
    if not utime.ticks_ms() - tempo_ultimo_toque < 200:
        global entrando_senha
        if entrando_senha:
            global tecla_pressionada
            tecla_pressionada = 2
            global contador_digitos_inseridos
            contador_digitos_inseridos += 1
        else:
            entrando_senha = True
            tecla_pressionada = 2
            contador_digitos_inseridos += 1
        tempo_ultimo_toque = utime.ticks_ms()
# Funcao chamada ao apertar o botao 3
def button_3_isr(pin):
    global tempo_ultimo_toque
    if not utime.ticks_ms() - tempo_ultimo_toque < 200:
        global entrando_senha
        if entrando_senha:
            global tecla_pressionada
            tecla_pressionada = 3
            global contador_digitos_inseridos
            contador_digitos_inseridos += 1
        else:
```

```
            entrando_senha = True
            tecla_pressionada = 3
            contador_digitos_inseridos += 1
        tempo_ultimo_toque = utime.ticks_ms()
# Funcao chamada ao apertar o botao 4
def button_4_isr(pin):
    global tempo_ultimo_toque
    if not utime.ticks_ms() - tempo_ultimo_toque < 200:
        global entrando_senha
        if entrando_senha:
            global tecla_pressionada
            tecla_pressionada = 4
            global contador_digitos_inseridos
            contador_digitos_inseridos += 1
        else:
            entrando_senha = True
            tecla_pressionada = 4
            contador_digitos_inseridos += 1
        tempo_ultimo_toque = utime.ticks_ms()
# Declaracao das interrupcoes que serao iniciadas ao toque dos botoes
botao_1.irq(trigger=botao_1.IRQ_FALLING,handler=button_1_isr)
botao_2.irq(trigger=botao_2.IRQ_FALLING,handler=button_2_isr)
botao_3.irq(trigger=botao_3.IRQ_FALLING,handler=button_3_isr)
botao_4.irq(trigger=botao_4.IRQ_FALLING,handler=button_4_isr)

# Funcao que realiza o cadastro do cartao
def cadastra_cartao(cartao_lido):
```

```
    global cartao_liberado
    global inicia_cadastro
    global tempo_ultima_atualizacao_display
    if cartao_lido[0] != -1:
        print(cartao_lido[0])
        cartao_liberado[0] = cartao_lido[0]
        cartao_lido[0] = -1
        display.fill(colour(0,255,0))
        display.printstring('Cartao',45,30,3,0,0,colour(0,0,0))
        display.printstring('Cadastrado!',20,90,3,0,0,colour(0,0,0))
        display.show()
        print("Cartao cadastrado!")
        inicia_cadastro = False
        tempo_ultima_atualizacao_display = utime.ticks_ms()
# Funcao que nega o acesso a cartoes ou senhas nao autorizadas
def barra_acesso():
    global tempo_ultima_atualizacao_display
    display.fill(colour(255,0,0))
    display.printstring("Acesso",60,60,3,0,0,colour(0,0,0))
    display.printstring("negado!",57,120,3,0,0,colour(0,0,0))
    display.show()
    print("Acesso negado!")
    tempo_ultima_atualizacao_display = utime.ticks_ms()
# Funcao que libera o acesso a cartoes ou senhas autorizados
def libera_acesso():
    global cartao_lido
    global tempo_ultima_liberacao
```

```
    global tempo_ultima_atualizacao_display
    cartao_lido[0] = -1
    display.fill(colour(0,255,0))
    display.printstring("Acesso",60,60,3,0,0,colour(128,128,128))
    display.printstring("liberado!",47,120,3,0,0,colour(128,128,128))
    display.show()
    print("Acesso liberado!")
    tempo_ultima_liberacao = tempo_ultima_atualizacao_display = utime.ticks_ms()
# Funcao que sera executada no segundo nucleo da pico W em paralelo a do
nucelo principal
def nucleo_secundario():
    global cartao_lido
    global contador_digitos_inseridos
    global contador_digitos_anterior
    global senha_cadastro
    global senha_acesso
    global senha_digitada
    global entrando_senha
    global cartao_liberado
    global inicia_cadastro
    global tempo_ultima_atualizacao_display
    # Laco de execucao do nucleo secundario
    while True:
        if not inicia_cadastro and utime.ticks_ms() -
tempo_ultima_atualizacao_display > 1000:
            display.fill(colour(244,255,113))
            display.printstring('Controle',45,30,3,0,0,colour(0,0,0))
            display.printstring('de',100,90,3,0,0,colour(0,0,0))
```

```
        display.printstring('Acesso',65,150,3,0,0,colour(0,0,0))
        display.show()
    if entrando_senha and utime.ticks_ms() - tempo_ultimo_toque < 10000:
        if contador_digitos_inseridos < 4 and contador_digitos_inseridos > contador_digitos_anterior:
            senha_digitada [contador_digitos_inseridos-1] = tecla_pressionada
            contador_digitos_anterior = contador_digitos_inseridos
        elif contador_digitos_inseridos == 4:
            senha_digitada [contador_digitos_inseridos-1] = tecla_pressionada
            if senha_digitada == senha_cadastro:
                print("Aproxime o cartao")
                display.fill(colour(0,133,220))
                display.printstring('Aproxime',45,30,3,0,0,colour(255,255,255))
                display.printstring('o',105,90,3,0,0,colour(255,255,255))
                display.printstring('cartao',60,150,3,0,0,colour(255,255,255))
                display.show()
                tempo_ultima_atualizacao_display = utime.ticks_ms()
                inicia_cadastro = True
                contador_digitos_inseridos = 0
                contador_digitos_anterior = 0
                entrando_senha = False
                senha_digitada = ['','','','']
            elif senha_digitada == senha_acesso:
                libera_acesso()
                contador_digitos_inseridos = 0
                contador_digitos_anterior = 0
                entrando_senha = False
```

```
            senha_digitada = ['','','','']
        else:
            barra_acesso()
            contador_digitos_inseridos = 0
            contador_digitos_anterior = 0
            entrando_senha = False
            senha_digitada = ['','','','']
    elif entrando_senha:
        contador_digitos_inseridos = 0
        contador_digitos_anterior = 0
        entrando_senha = False
        senha_digitada = ['','','','']
    if not inicia_cadastro and (cartao_lido[0] == cartao_liberado[0] and
cartao_liberado[0] != -1
       and utime.ticks_ms() - tempo_ultima_liberacao > 1000):
        libera_acesso()
    elif not inicia_cadastro and cartao_lido[0] != cartao_liberado[0] and
cartao_lido[0] != -1:
        barra_acesso()
        cartao_lido[0] = -1
    utime.sleep(0.1)
# Inicializacao da nova rotina no nucelo secundario e espera para que ela inicie
com sucesso
_thread.start_new_thread(nucleo_secundario,())
utime.sleep(0.25)
# Laco de execucao
while True:
  if inicia_cadastro:
```

```
        cartao_lido[0] = reader.read_id()
        tempo_ultima_leitura = utime.ticks_ms()
        cadastra_cartao(cartao_lido)
    elif utime.ticks_ms()- tempo_ultima_leitura > 1000:
        cartao_lido[0] = reader.read_id()
        tempo_ultima_leitura = utime.ticks_ms()
    utime.sleep(0.1)
```

15. Projetos IoT

E agora que você já sabe muito sobre a Pico W, Micropython e como interagir com sensores, luzes e sons, vamos aproveitar o W da nossa placa. Ele indica que a placa possui conexão Wi-Fi, então você pode conectá-la à sua rede sem fio e tanto buscar dados na internet quanto enviá-los. Isso será muito útil para os projetos IoT que separamos para você, além de tantos outros que você possa imaginar.

15.1 O que é IoT?

IoT é a sigla do termo em inglês **Internet of Things**, ou **Internet das Coisas** em português. O termo é creditado a Kevin Ashton, um pesquisador do MIT. Tudo começou em 1999, numa tentativa de chamar a atenção de executivos de uma grande empresa para utilizar tags RFID nos produtos para auxiliar o controle de estoque. Ele montou uma apresentação chamada “That ‘Internet of Things’ Thing” ou “Aquela coisa de internet das coisas”. Naquela época a internet estava engatinhando ainda, mas já chamava a atenção e essa foi a intenção de Kevin. Mas a semente do IoT estava plantada, bem como a ideia de o estoque se “autorregular”, ao detectar que o estoque se esgotou a ideia era que fosse disparado um pedido de reposição de forma automática.

A internet em si depende da interação humana, acessar um site, carregar um arquivo. A ideia da internet das coisas é que as “coisas” enviem dados de forma autônoma para a internet, seja uma leitura de temperatura, um alerta. Imagine que a sua geladeira está conectada a internet, detecta que você usou a última caixa de leite e acrescenta automaticamente à sua lista de compras mais leite. Aí

está uma das muitas faces da internet das coisas, dispensar a interação humana para várias tarefas.

15.2 Acessando a rede

Para os projetos IoT, a primeira coisa que você precisa saber é como conectar a Pico W a internet. Nesta aula iremos criar um código para conectar sua Pico W à internet. Faremos um arquivo separado para as credenciais de rede e outros dados que devemos evitar de compartilhar com o código principal.

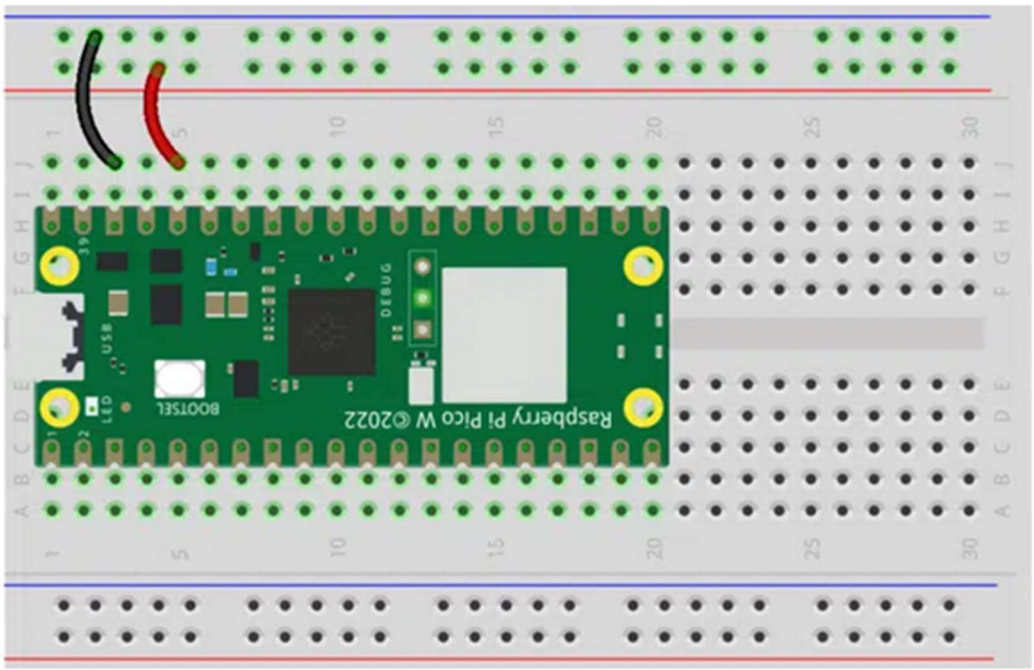
Material necessário

Neste projeto, precisamos dos seguintes componentes:

- Raspberry Pi Pico W
- Protoboard 400 pontos
- Cabo USB

Montagem do circuito

Esta atividade não conecta nenhum outro componente a Pico W, basta conectá-la ao computador com o cabo USB



Programa

Conectar a Pico W à Wi-Fi é um processo relativamente simples e que precisa de apenas 5 linhas de código, sendo que você precisa trocar “SSID” e “SENHA” pelo nome da sua rede sem fio e senha, colocando entre aspas como estão SSID e SENHA:

1import network

2wlan = network.WLAN(network.STA_IF)

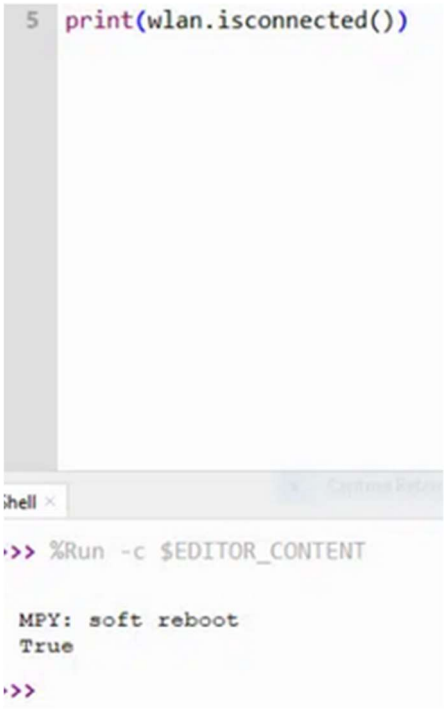
3wlan.active(True)

4wlan.connect("SSID","SENHA")

5print(wlan.isconnected())

A biblioteca network implementa todo o protocolo de comunicação e criptografia necessário para a conexão de rede. O parâmetro **STA_IF** indica que a Pico W trabalhará em modo cliente, conectando-se a uma rede existente. Tudo que precisamos fornecer é o **ssid** e a **senha** da rede.

Caso tudo corra bem, você verá a mensagem True no terminal. Se ver a mensagem False, tente rodar novamente. Se ainda ver false, cheque se o ssid e a senha da rede estão corretos.



Como nem sempre as conexões são instantâneas, é bom que coloquemos um tempo de espera antes de ter certeza que a conexão está funcionando, e também retornar um erro caso não consiga realizar a conexão. O código abaixo faz essa função

```
import network

import time

wlan = network.WLAN(network.STA_IF)

wlan.active(True)

wlan.connect("SSID","SENHA")

# Espera por 10 segundos pela conexão Wi-Fi

wait = 10

while wait > 0:

if wlan.status() < 0 or wlan.status() >= 3:

  break

wait -= 1

print('aguardando conexao...')

time.sleep(1)

# Verifica o status da conexão após a espera

if wlan.status() != 3:

raise RuntimeError('a conexao falhou')

else:

print('conectado')

print('IP: ', wlan.ifconfig()[0])
```

Caso tudo corra bem, você verá a mensagem ‘conectado’ e o IP que a placa obteve da rede no terminal.

Apesar de rápidos e práticos, os códigos acima apresentam um problema. O SSID e a senha da rede sem fio estão no código, então se você quiser compartilhar seus códigos com amigos, ou até mesmo em repositórios públicos, irá compartilhar também o acesso da sua rede sem fio. Para evitar esse problema iremos criar um arquivo separado para armazenar as informações privadas de SSID e senha da rede. Você pode chamar o arquivo da forma que quiser, mas no tutorial chamaremos de **credenciais.py**

Dentro do arquivo **credenciais.py** colocaremos o seguinte texto:

1ssid = 'SSID',

2senha = 'SENHA'

Troque o **SSID** e **SENHA** entre aspas simples pelas suas informações, que também deverão ficar entre aspas simples.

Salve dentro da pasta lib da Raspberry Pi Pico o arquivo credenciais.py

Vamos repetir o código com espera agora utilizando a estrutura com as credenciais separadas.

```
import network

import utime

import credenciais


wlan = network.WLAN(network.STA_IF)
```

```
wlan.active(True)
wlan.connect(credenciais.ssid,credenciais.senha)
# Espera por até 10 segundos pela conexão Wi-Fi
wait = 10
while wait > 0:
  if wlan.status() < 0 or wlan.status() >= 3:
    break
  wait -= 1
  print('aguardando conexao...')
  utime.sleep(1)
# Verifica o status da conexão após a espera
if wlan.status() != 3:
  raise RuntimeError('a conexao falhou')
else:
  print('conectado')
  print('IP: ', wlan.ifconfig()[0])
```

Agora você já sabe conectar a sua placa a Wi-Fi para fazer os projetos que desejar, mas para facilitar um pouco a vida, em especial porque todos os próximos projetos dependem da conexão, vamos criar um arquivo que será responsável por fazer a conexão, dessa forma você não precisará copiar e colar o código de conexão a cada projeto.

Crie um arquivo, no tutorial iremos chamá-lo de realiza_conexao.py, mas você pode chamar como preferir, desde que altere de acordo no código.

```
import network
import time
import credenciais


def realiza_conexao():
```

```
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(credenciais.ssid, credenciais.senha)

# Espera por até 10 segundos pela conexão Wi-Fi
wait = 10
while wait > 0:
  if wlan.status() < 0 or wlan.status() >= 3:
    break
  wait -= 1
  print('Aguardando conexao...')
  time.sleep(1)

# Verifica o status da conexão após a espera
if wlan.status() != 3:
  raise RuntimeError('a conexao falhou')
else:
  print('conectado')
  ip=wlan.ifconfig()[0]
  print('IP: ', ip)
  return ip
```

Agora salve o arquivo na pasta lib dentro do Raspberry Pi Pico W e para os próximos códigos você fará a importação da biblioteca da seguinte forma:

```
1from realiza_conexao import *
```

Isso significa que estamos importando todas as funções contidas em realiza_conexão

E então fazemos a chamada no código da função

```
1realiza_conexao()
```

Caso precisemos do endereço de IP no código principal podemos atribuir o valor que realiza_conexao retorna a uma variável, como o exemplo abaixo

```
1ip = realiza_conexao()
```

Agora você tem todas as ferramentas necessárias para conectar sua Pico W à Wi-Fi e desbloquear uma nova gama de projetos. Nas próximas atividades apresentaremos alguns exemplos, mas o limite é a sua criatividade.

15.3 Conectando à luz do @CheerLigths

O projeto CheerLights surgiu quando seu criador, Hans Scharler, visitou uma loja de materiais de construção em 2011 e viu luzes de natal controladas por controle remoto. Seu vizinho já havia colocado luzes coloridas e música na decoração, e então Hans pensou se as suas luzes estivessem conectadas às do seu vizinho. E se as luzes do quarteirão inteiro estivessem conectadas, luzes da cidade toda, e, porque não, milhares de luzes ao redor do mundo todo? A internet pode manter a sincronia de todas elas. E a partir daí e das ideias de IoT Hans criou o projeto e engajou uma comunidade para fazer luzes interconectadas pelo mundo todo, ensinando e servindo de porta de entrada para conceitos de projetos IoT.

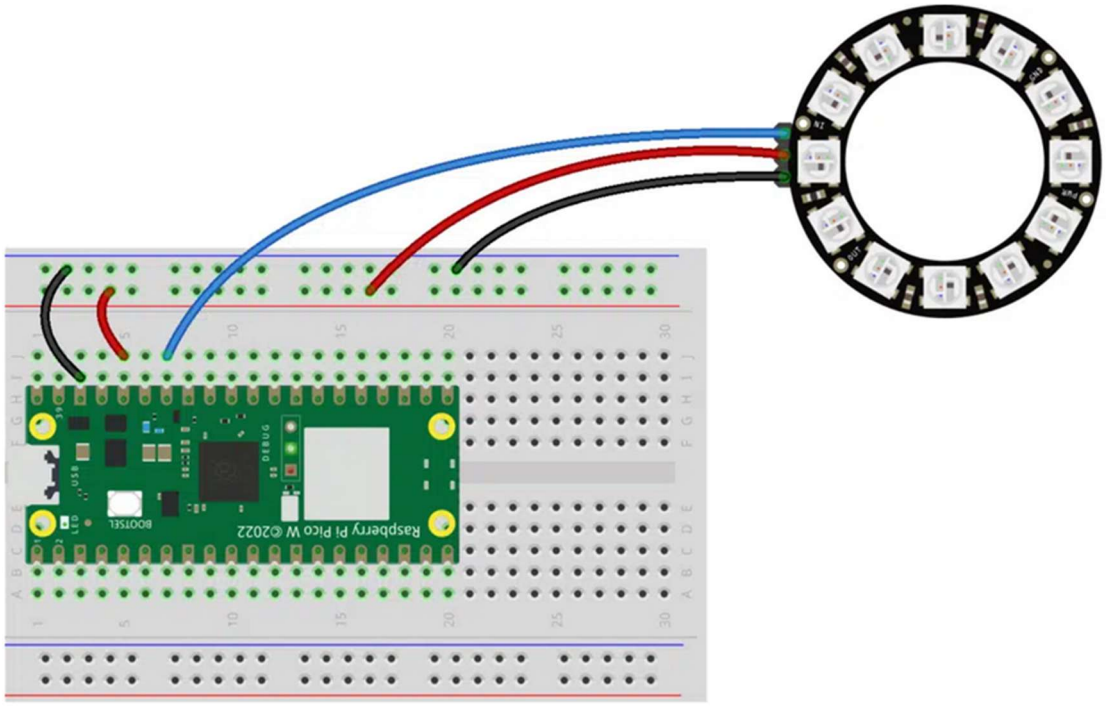
Material necessário

Neste projeto, precisamos dos seguintes componentes:

- Anel de LED
- Protoboard 400 pontos
- Placa Raspberry Pi Pico W
- Jumpers Macho-Macho
- Cabo USB

Montagem do circuito

A montagem será igual a da atividade 5.6, conforme o esquemático abaixo



Programa

O programa usa algumas bibliotecas como a **urequests** e **json**. A **urequests** realiza a comunicação de internet, contendo os protocolos adequados e realizando a troca de informações. Já a **json** trata o arquivo recebido para extrair as informações necessárias.
Para esse código precisamos dos arquivos **credenciais.py**, **realiza_conexao.py** e **ws2812.py** carregados nas bibliotecas da placa.

```
import urequests

import json

import time

import machine

from ws2812 import WS2812

from realiza_conexao import *

ip = realiza_conexao()

print ('IP: ',ip)


ws = WS2812(machine.Pin(28), 12)
```

```
def get_colour():
    url = "http://api.thingspeak.com/channels/1417/field/2/last.json"
    try:
        r = urequests.get(url)
        if r.status_code > 199 and r.status_code < 300:
            cheerlights = json.loads(r.content.decode('utf-8'))
            print(cheerlights['field2'])
            colour = int('0x'+cheerlights['field2'][1:7])
            r.close()
            return colour
        else:
            print('Erro HTTP:',r.status_code)
            return None
    except Exception as e:

        print(e)
        return None
while True:
    colour = get_colour()
    if colour is not None:
        ws.write_all(colour)
    time.sleep(60)
```

15.11 Sistema de segurança

Na atividade 14.2 aprendemos a fazer uma sirene e luzes de alerta. Mas e se quisermos automatizar isso? Vamos usar o sensor de presença para disparar a sirene automaticamente e também vamos usar a conexão Wi-Fi para enviar uma notificação pelo aplicativo Telegram para avisar que houve uma detecção de movimento.

Você pode enviar o texto **Armar** no telegram para ligar o alarme e **Desarmar** para desligar.

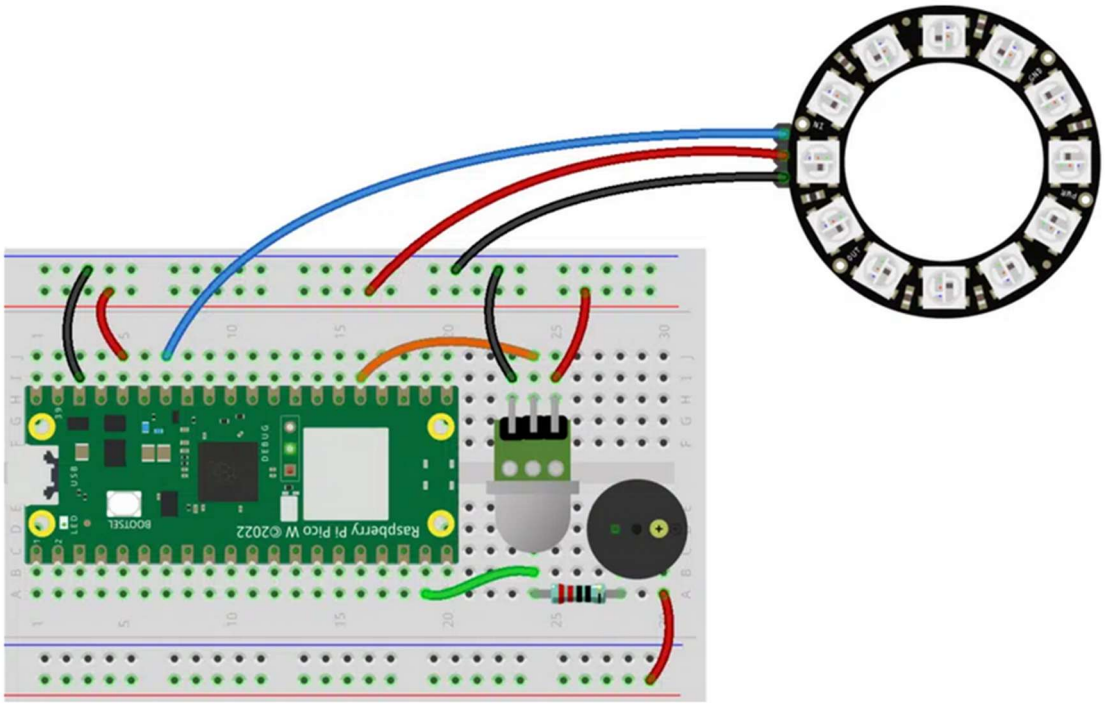
Material necessário

Neste projeto, precisamos dos seguintes componentes:

- Protoboard 400 pontos
- Placa Raspberry Pi Pico W
- Jumper Macho-Macho
- Anel de LEDs
- Sensor de presença PIR
- Resistor 220 ohms
- Buzzer
- Cabo USB

Montagem do circuito

A montagem será conforme o esquemático abaixo



Programa

Para o código precisamos ter os arquivos **credenciais.py** e **realiza_conexao.py** carregados nas bibliotecas da placa.

Você pode conferir o código completo abaixo.

```
# Codigo 15.11 - Sistema de Segurança

# Bibliotecas necessarias para o codigo

# Importacao das bibliotecas padrao

import machine

import urequests

import utime

import _thread

# Importacao das bibliotecas customizadas

from realiza_conexao import *

import credenciais

from ws2812 import WS2812
```

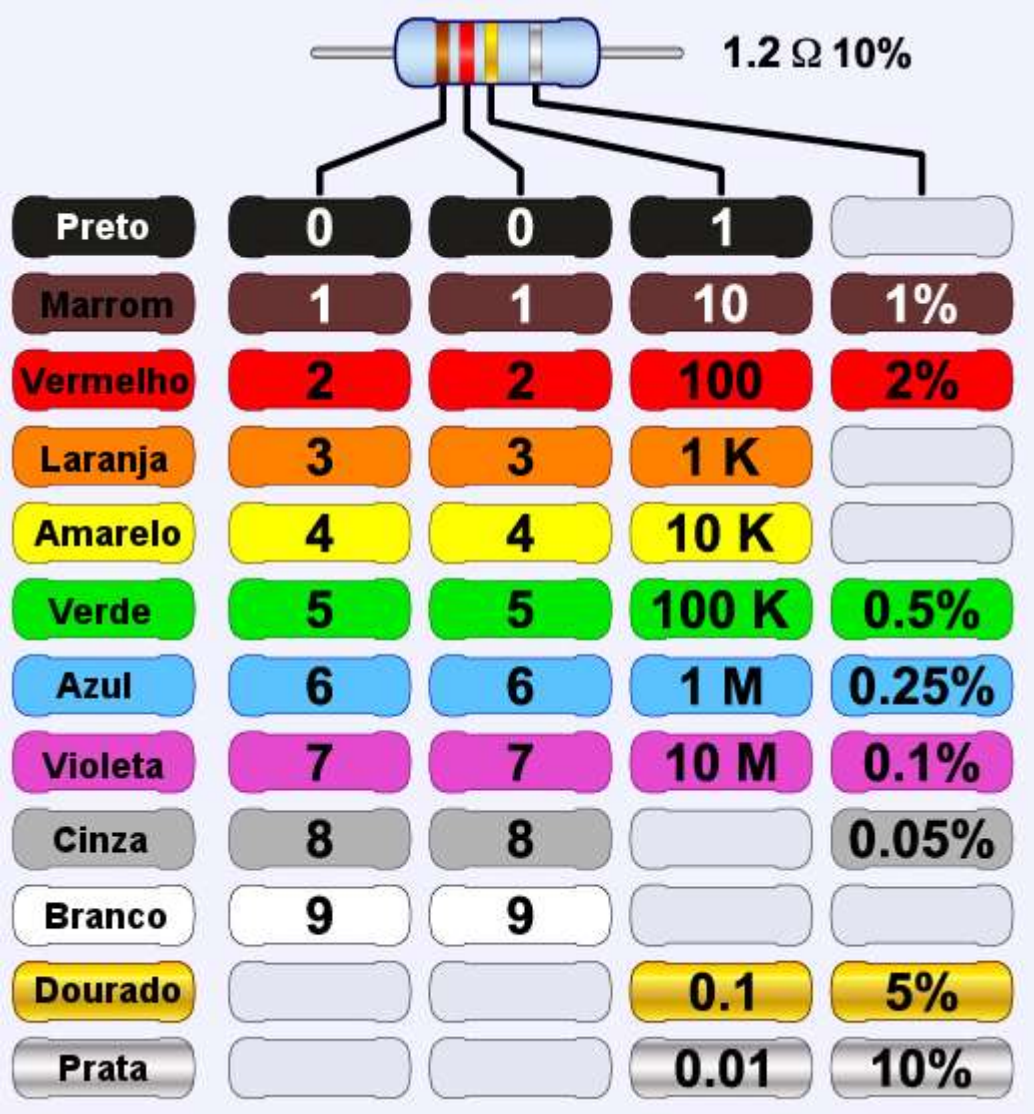
```
# Declaracao do Anel de LEDs
ws = WS2812(machine.Pin(28), 12)
ws.write_all(0x000000)
# Declaracao do buzzer
buzzer = machine.PWM(machine.Pin(14, machine.Pin.OUT))
buzzer.duty_u16(65535)
# Declaracao do sensor PIR
pir_sensor = machine.Pin(16, machine.Pin.IN)
# Declaracao das variaveis
alarme_armado = False
alarme_disparado = False
movimento_detectado = False
global URL_da_requisicao
global OFFSET
global URL_de_envio
OFFSET = 0
URL_da_requisicao = 'https://api.telegram.org/bot' + credenciais.bot_token
+'/getUpdates'
URL_de_envio = 'https://api.telegram.org/bot' + credenciais.bot_token
+'/sendMessage'
# Funcao que formata o retorno da mensagem do telegram
def extrai_resultado (dict):
  array_da_resposta = dict['result']
  if array_da_resposta == []:
    return False
  else:
    array_resultante = array_da_resposta[0]
```

```
        return array_resultante
# Funcao que envia mensagem para o chat
def envia_mensagem (chatId, message):
  urequests.post(URL_de_envio + "?chat_id=" + str(chatId) + "&text=" + message)
# Funcao que executa as rotinas de disparo do alarme
def da_alarme ():
  global alarme_disparado
  alarme_disparado = True
  _thread.start_new_thread(aciona_sirene,())
  envia_mensagem(credenciais.id_do_chat,'Alarme disparou!')
# Funcao que recebe a mensagem do chat
def recebe_mensagem (url):
  global OFFSET
  try:
    pacote_bruto = urequests.get(url + "?offset=" + str(OFFSET))
    pacote = pacote_bruto.json()
    resultado_final = extrai_resultado(pacote)
    if resultado_final != False:
      OFFSET = resultado_final['update_id'] + 1
      id_do_chat = resultado_final['message']['chat']['id']
      mensagem = resultado_final['message']['text']
      return mensagem
  except:
    return None
# Funcao chamada ao haver uma deteccao de movimento
def detectou_movimento(_):
  global movimento_detectado
```

```
    movimento_detectado = True
# Declaracao da interrupcao que sera iniciada ao detectar movimento
pir_sensor.irq(trigger=machine.Pin.IRQ_RISING, handler=detectou_movimento)
# Funcao que aciona o giroflex e a sirene
def aciona_sirene():
    j=0
    buzzer.duty_u16(32767)
    timeSinceLastSwitch = utime.ticks_ms()
    buzzer.freq(500)
    proximoMultiplicador = 3
    while j<700 and alarme_armado:
        for i in range(12):
            if ((i+j)%12)//6 < 1:
                ws.write(i,0xFF0000)
            elif ((i+j)%12)//6 == 1:
                ws.write(i,0X0000FF)
        if ((utime.ticks_ms() - timeSinceLastSwitch > 300)):
            buzzer.freq(500*proximoMultiplicador)
            if proximoMultiplicador == 3:
                proximoMultiplicador = 1
            else:
                proximoMultiplicador = 3
            timeSinceLastSwitch = utime.ticks_ms()

        # Inicio das luzes estroboscopicas, remova essa parte do codigo para que elas
nao acontecam
        if j%59 == 0:
            ws.write_all(0xFFFFFF)
```

```
        buzzer.freq(500)
        utime.sleep_ms(40)
        ws.write_all(0xFF0000)
        buzzer.freq(100)
        utime.sleep_ms(40)
        ws.write_all(0xFFFFFF)
        buzzer.freq(500)
        utime.sleep_ms(40)
        ws.write_all(0x0000FF)
        buzzer.freq(100)
        utime.sleep_ms(40)
        ws.write_all(0xFFFFFF)
        buzzer.freq(500)
        utime.sleep_ms(40)
        ws.write_all(0xFF0000)
        buzzer.freq(100)
        utime.sleep_ms(40)
        ws.write_all(0xFFFFFF)
        buzzer.freq(500)
        utime.sleep_ms(40)
        ws.write_all(0x0000FF)
        buzzer.freq(100)
        utime.sleep_ms(40)
        buzzer.freq(500*proximoMultiplicador)
    # Fim das luzes estroboscopicas
    j+=1
    utime.sleep_ms(20)
```

```
    ws.write_all(0x000000)
    buzzer.duty_u16(65535)
    global alarme_disparado
    alarme_disparado = False
# Realiza a conexao Wi-Fi
realiza_conexao()
# Laco de execucao
while True:
    mensagem = recebe_mensagem(URL_da_requisicao)
    if mensagem is not None:
        if mensagem == 'armar' or mensagem == 'Armar' or mensagem == 'ARMAR':
            alarme_armado = True
            envia_mensagem(credenciais.id_do_chat, 'Alarme armado!')

        elif mensagem == 'desarmar' or mensagem == 'Desarmar' or mensagem ==
'DESARMAR':
            alarme_armado = False
            envia_mensagem(credenciais.id_do_chat, 'Alarme desarmado!')
    if alarme_armado and movimento_detectado and not alarme_disparado:
        da_alarme()
        movimento_detectado = False
    elif alarme_armado and movimento_detectado and alarme_disparado:
        movimento_detectado = False
    elif not alarme_armado and movimento_detectado:
        movimento_detectado = False
    utime.sleep(.3)
```


1.2 Ω 10%
Preto 0 0 1
Marrom 1 1 10 1%
Vermelho 2 2 100 2%
Laranja 3 3 1 K
Amarelo 4 4 10 K
Verde 5 5 100 K 0.5%
Azul 6 6 1 M 0.25%
Violeta 7 7 10 M 0.1%
Cinza 8 8 0.05%
Branco 9 9
Dourado 0.1 5%
Prata 0.01 10%