# LED-Glühwürmchen

Ein kleines LED-Glühwürmchen blinkt


- Methodenkarte: LED-Glühwürmchen
- Installation Thonny
- Das Raspberry Pi Pico mit Thonny programmieren
- Zusammenbau
- Der Code
- Online-Ressourcen

# Methodenkarte: LED-Glühwürmchen

# Ein kleines LED-Glühwürmchen blinkt

| Zielgruppe | Dauer | Level | Gruppengröße |
|---|---|---|---|
| ab 8 Jahren | 3,5 bis 7 Stunden | 3 | 2 TN's |

## Kurzbeschreibung

In diesem Projekt wird eine kreativ gestaltete RGB-LED mit dem Mikrocontroller Raspberry Pi Pico verbunden, um sie zum Blinken zu bringen. Das Projekt gehört zum Physical Programming und nutzt eine modulare Toolbox für flexible, erweiterbare Projekte. Pro Set sollten ein bis zwei Teilnehmende zusammenarbeiten und diese Gruppen beibehalten.

> **Ziele**
> - ein Verständnis für Stromkreise vermitteln
> - den kreativen Umgang mit Code und digitalen Technologien fördern
> - Teamfähigkeit stärken und Zusammenarbeit unterstützen

| Material | Werkzeug |
|---|---|
| • Raspberry Pi Pico<br>• Breadboard<br>• Jumper, Dupont Kabel<br>• RGB LED-Diode<br>• | • Lötset/Lötstation<br>• Zangenset<br>• Sortier-/Projektbox |

## Ablauf

1. **Vorbereitung:**
   Zunächst werden alle Bauteile getestet und ein Demomodell gebaut. Alle notwendigen Zusatzmaterialien sollten ausgedruckt und die Programmierumgebung Thonny auf den Computern oder Raspberry-Pi-Geräten installiert werden (siehe QR-Code). Alle Materialien

werden in Projektboxen gepackt, außerdem sollte für Papier, Stifte und Snacks gesorgt sein. Um eine optimale Betreuung zu gewährleisten, sollte der Workshop von zwei Fachkräften durchgeführt werden.

2. **Projektstart:**
   Das Projekt startet mit einer Willkommensrunde. Workshopleitung und Teilnehmende besprechen die Erwartungen an das Projekt und stellen gemeinsame Arbeitsregeln auf, die alle Teilnehmenden unterschreiben. Anschließend werden Ideen für kreative Einsatzmöglichkeiten der RGB-LEDs gesammelt. Die Teilnehmenden bilden Zweiergruppen, geben sich Gruppennamen und beschriften ihre Projektboxen entsprechend. Dieser Einstieg schafft eine produktive Atmosphäre und fördert die Teamdynamik.
3. **Praktische Arbeit, Aufbau und Test:**
   In dieser Phase bauen die Teilnehmenden die Materialien aus den Projektboxen zusammen. Die Workshopleitung unterstützt bei Bedarf. Nach dem Zusammenbau testen die Teilnehmenden die Projekte. Es wird geprüft, ob die LEDs wie geplant leuchten und die Bewegungssensoren korrekt funktionieren. Nun kann reflektiert werden, in welchen Situationen diese Technik sinnvoll eingesetzt werden kann.
4. **Programmierung:**
   Zuerst wird der Raspberry Pi Pico über USB mit einem Computer verbunden. In der Programmierumgebung Thonny können die Teilnehmenden den Code anpassen, z.B. die Blinkfrequenz der LEDs. Um Variationen in den Effekten zu erhalten, können sie mit den Werten experimentieren. Schließlich können die Teilnehmenden weitere Blinkmuster hinzufügen, um die Möglichkeiten der Programmierung zu erweitern und ihre Fähigkeiten zu vertiefen.
5. **Reflexion:**
   Jetzt reflektiert jede Gruppe zunächst individuell, was gut lief und was verbessert werden könnte. Anschließend teilen die Gruppen diese Ergebnisse im Plenum. Alle Teilnehmenden erhalten Feedback zu den Projekten. Gemeinsam werden Ideen entwickelt, wie die erlernten Fähigkeiten in neuen Projekten angewendet werden können.

**Autor*in:** Shelly Pröhl (*Büro Berlin des JFF*)

# Installation Thonny

Wir programmieren das Raspberry Pi Pico mit der Skriptsprache MicroPython in der kostenlosen Entwicklungsumgebung (IDE) Thonny. Dafür verwenden wir einen Computer oder Laptop.

| | |
|---|---|
| Thonny | https://thonny.org/ |
| Micropython | https://micropython.org/ |

# Was ist eine IDE?

Eine IDE (Integrated Development Environment) ist eine Software, die euch beim Schreiben, Testen und Ausführen von Code (Programmen) unterstützt. Sie vereint viele hilfreiche Werkzeuge an einem Ort, darunter:

- **Texteditor:** Zum Schreiben und Bearbeiten von Code.
- **Debugger:** Zum Finden und Beheben von Fehlern im Code.
- **Terminal:** Zum Ausführen des Codes und Anzeigen von Ergebnissen.

# Was ist Thonny?

Thonny ist eine einfache und benutzerfreundliche IDE, die speziell für Python entwickelt wurde. Sie eignet sich besonders gut für Einsteiger*innen, die das Programmieren gerade erst lernen. Thonny bietet eine übersichtliche Benutzeroberfläche und viele hilfreiche Funktionen, die den Einstieg ins Programmieren erleichtern. Es ist ein großartiges Tool, um erste Schritte mit Python und MicroPython zu machen.

drawing drawing

# Installation von Thonny auf verschiedenen Betriebssystemen

**Installation auf Windows**

- **Gehe zur offiziellen Thonny-Website.**
- Klicke auf den Download für Windows.
- Lade die Installationsdatei herunter und öffne sie, wenn der Download abgeschlossen ist.

- Folge den Anweisungen des Installationsassistenten, um Thonny zu installieren.
- Sobald die Installation abgeschlossen ist, kannst du Thonny über das Startmenü öffnen.

**Installation auf Linux (Ubuntu)**

- **Öffne das Terminal** auf deinem Ubuntu-System.
- Gib den folgenden Befehl ein, um das Paket zu aktualisieren:

```
sudo apt update
```

- Installiere Thonny, indem du den folgenden Befehl eingibst:

```
sudo apt install thonny
```

- Warte, bis die Installation abgeschlossen ist. Danach kannst du Thonny im Anwendungsmenü finden und starten.

**Installation auf macOS**

- **Gehe zur offiziellen Thonny-Website.**
- Klicke auf den Download für macOS.
- Lade die .dmg-Datei herunter und öffne sie, wenn der Download abgeschlossen ist.
- Ziehe das Thonny-Symbol in den Programme-Ordner, um die Installation abzuschließen.
- Öffne Thonny, indem du es im Programme-Ordner findest oder über Spotlight suchst.

> **Übung**
>
> Versuche nach der Installation, siehe unten, über die IDE Thonny eine Bibliothek zu installieren, zum Beispiel die Bibliothek 'NeoPixel'.
>
> - Thonny öffnen ->
> - Menü -> Tools -> Manage Plug-ins
> - Suche nach 'neopixel' -> installieren

# Das Raspberry Pi Pico mit Thonny programmieren

Um ein neues Raspberry Pi Pico zu programmieren, müssen wir es zunächst vorbereiten. Es mag anfangs nach vielen Schritten klingen, aber sobald ihr es einmal gemacht habt, geht der Rest richtig schnell! (□ ‘‿ ’□ ) Wir teilen den Prozess in drei Schritte auf:

1. **Installation von MicroPython auf dem Raspberry Pi Pico**
2. **Raspberry Pi Pico mit Thonny öffnen**
3. **Ein Programm auf dem Raspberry Pi Pico speichern/laden**

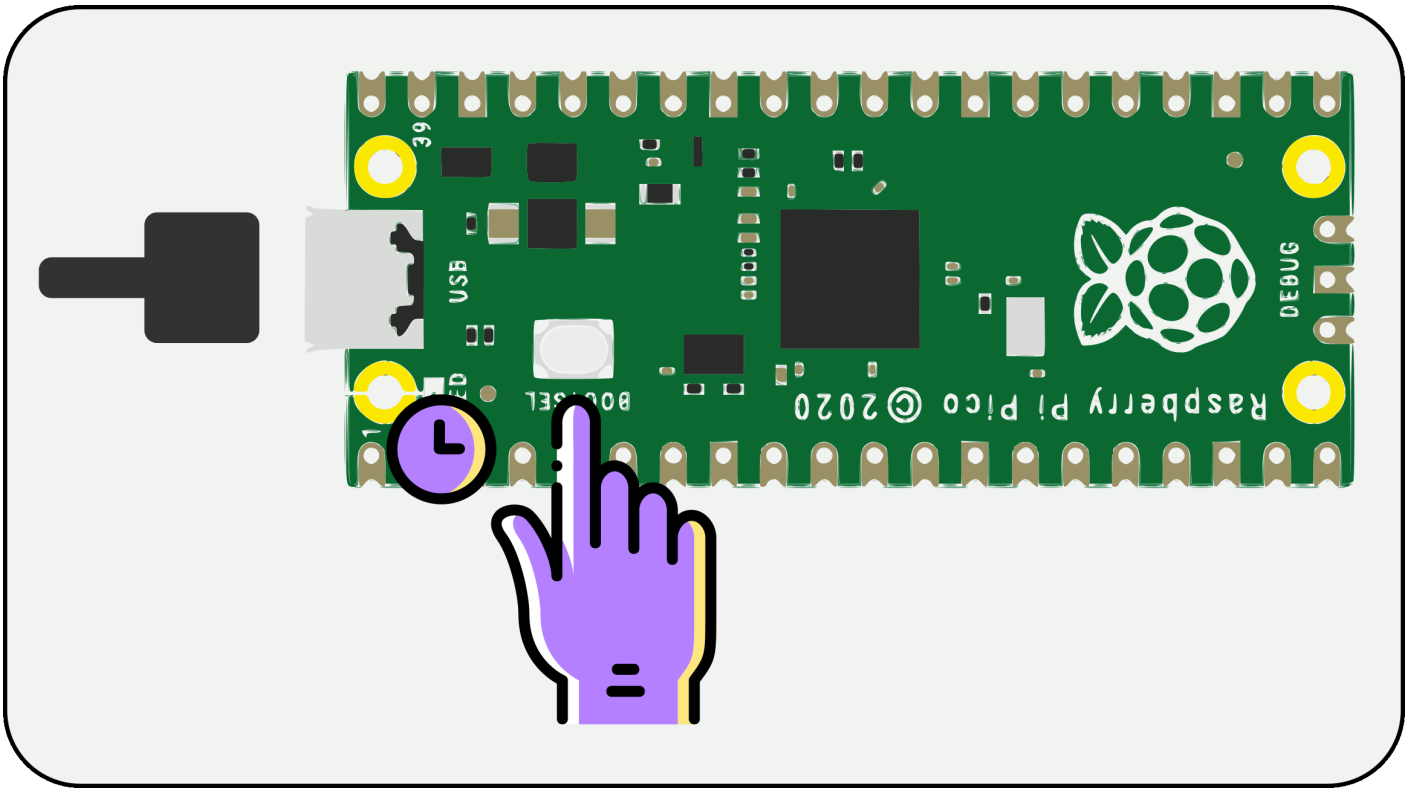
**Installation von Micropython auf dem Raspberry Pi Pico**

- **Ladet euch Micropython herunter:**

  Besucht die MicroPython-Website und ladet die passende UF2-Datei für das Raspberry Pi Pico herunter.

  *(Zum Beispiel v1.24.0 (2024-10-25) .uf2)*
- **Schließt das Raspberry Pi Pico an den Computer an:**
  Verwendet ein Micro-USB-zu-USB-A-Kabel. Haltet dabei den BOOTSEL-Knopf beim Raspberry Pi Pico gedrückt, während ihr das USB-Kabel anschließt.
  *(siehe Animation)*
- **Kopiert die UF2-Datei auf das neue Laufwerk:**
  Sobald das Pico als neues Laufwerk auf eurem Computer erscheint, zieht die heruntergeladene UF2-Datei per Drag-and-Drop in dieses Laufwerk. Dadurch wird MicroPython auf dem Raspberry Pi Pico installiert.

> Während der Installation oder dem Laden von MicroPython auf das Raspberry Pi Pico trennt sich das Laufwerk automatisch vom Computer. Dies zeigt an, dass die Installation abgeschlossen ist. Zieht auf keinen Fall das Kabel während dieses Prozesses vom Pico ab, da dies die Installation unterbrechen und zu Fehlern führen könnte.
>
> Sicherheit geht vor – lasst das Pico in Ruhe arbeiten! (□ ^ ᵕ ^□ )



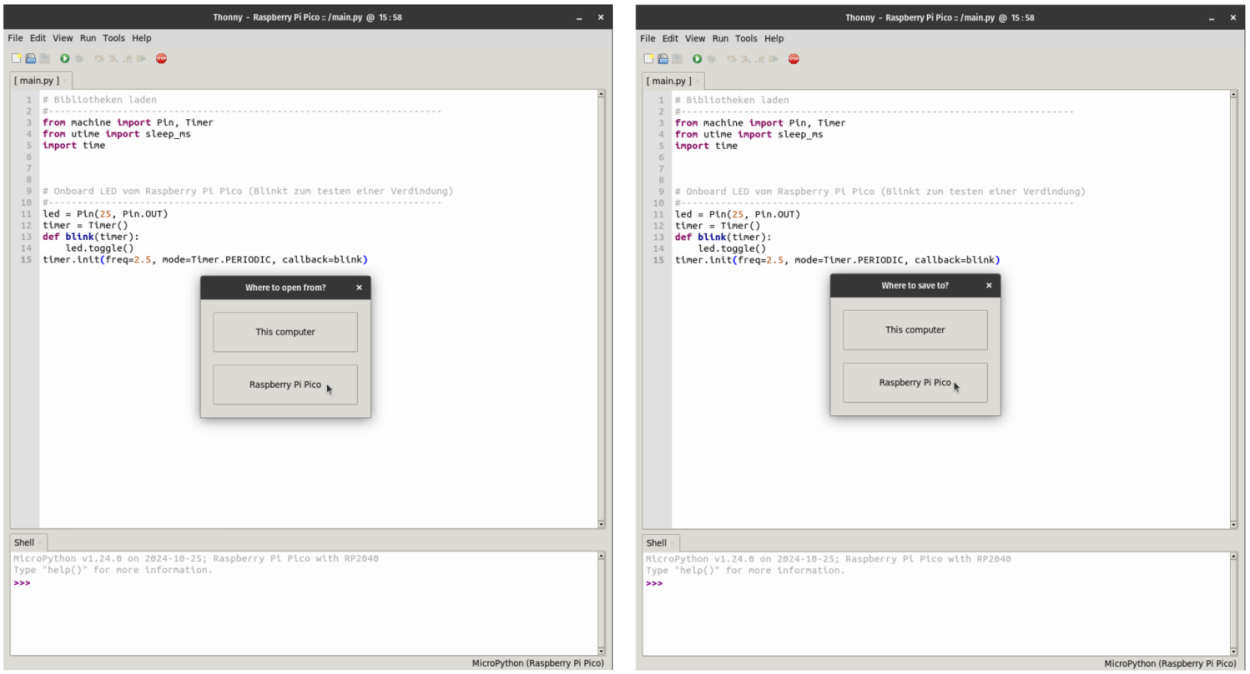

**Mit Thonny ein Programm auf dem Raspberry Pi Pico speichern/laden**

Nachdem ihr Thonny geöffnet habt, könnt ihr über das Menü die Option **Dateien/Files** auswählen, um Dateien zu verwalten.

- Mit der Option **Öffnen/Open** öffnet sich ein Dialog, in dem ihr auswählen könnt, ob ihr eine Datei von eurem Computer oder direkt vom Raspberry Pi Pico laden möchtet.
- Mit der Option **Speichern/Save** öffnet sich ein Dialog, in dem ihr auswählen könnt, wo ihr euer Programm speichern möchtet – entweder auf dem Computer oder auf dem Raspberry Pi Pico.

So könnt ihr eure Programme einfach verwalten und sicherstellen, dass sie immer an der richtigen Stelle gespeichert sind!

> Achtet beim Speichern auf dem Raspberry Pi Pico darauf, dass die Datei den Namen **main.py** hat. Nur mit diesem Dateinamen erkennt das Pico euer Programm automatisch und führt es nach dem Starten aus. ᓚ( ^. .^)



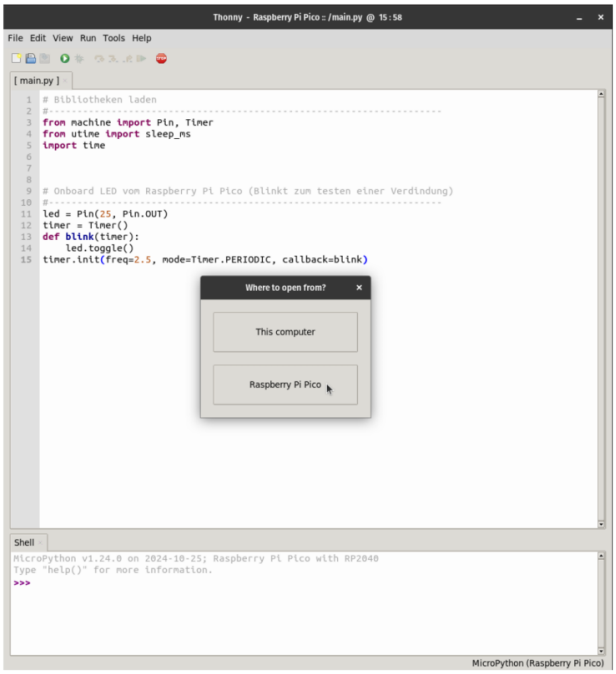

### Raspberry Pi Pico mit Thonny öffnen

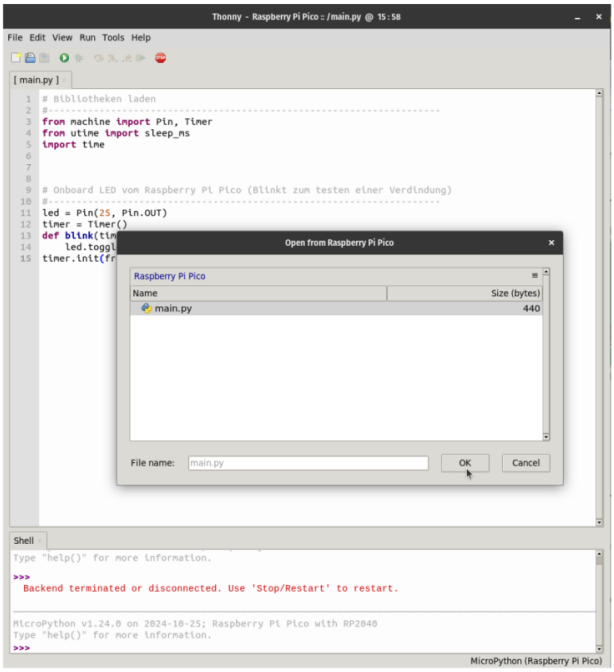
Normalerweise kommuniziert Thonny automatisch mit dem Raspberry Pi Pico, sobald es angeschlossen ist. Wenn ihr Thonny geöffnet habt, geht wie folgt vor:

1. Wählt im Menü die Option **Dateien/Files** aus.
2. Im neuen Dialog könnt ihr das **Raspberry Pi Pico** als Speicherort auswählen.
3. Anschließend könnt ihr eure **main.py**-Datei auf dem Pico finden, auswählen und öffnen.

So könnt ihr sicherstellen, dass euer Programm korrekt geladen und ausgeführt wird!

> Nachdem ihr eure **main.py** zum Öffnen ausgewählt habt, könnt ihr das Programm in Thonny bearbeiten. Wenn ihr Änderungen vornehmt und auf **Speichern** klickt, wird das Programm automatisch auf dem Raspberry Pi Pico aktualisiert und gespeichert. So bleiben eure Änderungen direkt auf dem Pico erhalten! ¯_(□ )_/¯




### Programm starten/stoppen

Nachdem ihr ein Programm erfolgreich auf das Raspberry Pi Pico gespeichert oder übertragen habt, könnt ihr es ausführen und bei Bedarf stoppen:

- **Starten:** Klickt in der Symbolleiste auf den runden **grünen Button**, um das Programm auf dem Raspberry Pi Pico zu starten. Alternativ könnt ihr die **Taste F5** drücken.
- **Stoppen:** Klickt in der Symbolleiste auf den **roten Stop-Button**, um das Programm zu stoppen. Alternativ könnt ihr die Tastenkombination **Strg + F2** verwenden.

Buttons in Thonny zur Steuerung des Programmes auf dem Raspberry Pi Pico



Programm
**starten**




Programm
**stoppen**

# Zusammenbau

drawing

> **Beispiel-Verkabelung (siehe Abbildung oben):**
> **In der Abbildung werden rote und schwarze Kabel (Jumper) verwendet, die jeweils eine spezifische Funktion haben:**
>
> **- Rot (Power/PWR): Liefert Energie an das Bauteil, damit es funktioniert.**
> **- Schwarz (Ground/GRD): Schließt den Stromkreis und leitet überschüssige Energie ab.**

> **Achtung:**
>
> In diesem Projekt ist alles etwas anders als wir es von der üblichen Verkabelung gewöhnt sind, da die Aufgabenteilung der Kabel durch die Eigenschaften der LED etwas verändert sind. Bleib aber unbedingt am Ball und erfahre, warum das alles auch mal anders funktionieren kann! (□ ^ ᵕ ^□ )

**Wie funktioniert eine LEDWas ist eine Anode und was ist eine Kathode?**
Eine **Anode** ist der Pluspol der LED, durch den der Strom hineinfließt. Das ist der **längere Pin** an der LED.
Eine **Kathode** ist der Minuspol der LED, durch den der Strom herausfließt. Das ist der **kürzere Pin** an der LED.

**Warum werden die Anoden- und Kathoden-Pins so angeschlossen?**

1. Die **Anode** (Pluspol) wird an den **3.3V-Pin** des Raspberry Pi Pico angeschlossen, weil sie konstant Strom benötigt, um die LED zum Leuchten zu bringen.
2. Die **Kathoden** (Minuspol) der LED werden über **Vorwiderstände** an die GPIO-Pins des Pico angeschlossen (z. B. GP28 wie in unserem Zusammenbau). Das liegt daran, dass die GPIO-Pins steuern, ob Strom durch die LED fließt oder nicht. Sie sind wie Schalter, die die LED an- und ausschalten können. Der Vorwiderstand schützt die LED vor zu starkem Stromfluss, der sie beschädigen könnte.

**Wie fließt der Strom durch die LED und warum ist die Verkabelung so wie beschrieben sinnvoll?**

Der Strom fließt immer von **Plus (3.3V)** nach **Minus (GND)**. LEDs funktionieren nur, wenn der Strom in diese Richtung fließt – von der **Anode (Pluspol)** zur **Kathode (Minuspol)**. Wenn du die **Anode (langer Pin)** der LED an den **3.3V-Pin** anschließt und die **Kathode (kurzer Pin)** über

einen Vorwiderstand an **GPIO 28** verbindest, funktioniert das folgendermaßen:

1. **3.3V-Pin** liefert konstanten Strom an die LED (Anode).
2. **GPIO 28** steuert den Stromfluss:
   - Wenn GPIO 28 auf **LOW (0V)** gesetzt ist, kann der Strom durch die LED zur Kathode fließen und die LED leuchtet.
   - Wenn GPIO 28 auf **HIGH (3.3V)** gesetzt ist, herrscht an beiden Enden der LED der gleiche Spannungspegel (kein Spannungsunterschied), und die LED bleibt aus.

Das bedeutet, der GPIO-Pin (in deinem Fall Pin 28) agiert hier nicht als Stromquelle, sondern als "Schalter", der entweder **Masse (GND)** anbietet (damit fließt Strom) oder "offen bleibt" (kein Strom fließt).

**Warum nicht umgekehrt (Anode an GPIO, Kathode an GND)?**

Wenn du die **Anode** an GPIO 28 und die **Kathode** an GND anschließt, könntest du die LED auch steuern, aber es ist nicht ideal, denn:

1. GPIO-Pins können nur begrenzt Strom liefern (typischerweise 12–15 mA). Der **3.3V-Pin** kann mehr Strom liefern und ist dafür ausgelegt, Komponenten zu versorgen.
2. Es ist sicherer für den Raspberry Pi Pico, wenn der GPIO-Pin nur die **Masse (LOW)** schaltet, statt als Stromquelle für die LED zu fungieren. So vermeidest du Überlastungen des GPIO-Pins.

> **Zusammenfassung:**
>
> - Die Anode (Pluspol) der LED sollte an den 3.3V-Pin angeschlossen werden, damit die LED eine konstante Stromversorgung bekommt.
> - Die Kathode (Minuspol) sollte an GPIO 28 (oder einen anderen GPIO-Pin) angeschlossen werden, damit du über den GPIO den Stromfluss zur Masse (GND) steuern kannst.
> - Der Vorwiderstand schützt die LED vor zu starkem Stromfluss.

# Der Code

Hier ist ein Beispielcode für eure Programmierung in Micropython in der IDE Thonny auf dem Raspberry Pi Pico.

```
# Bibliotheken laden
#------------------------------------------------------------------------
from picozero import pico_led, LED
from time import sleep

# Pin GP(IO)28 für die Programmierung wählen,
# hier wird der Kathoden-Pin der LED mit dem Pin 28 des Picos verbunden
#------------------------------------------------------------------------
firefly = LED(28)

# Unendlichkeits-Schleife
#------------------------------------------------------------------------
while True:
    # LED an
    firefly.high()
    # Pause für 0.5 ms
    sleep(0.5)
    # LED aus
    firefly.low()
    # Pause für 2.5 ms
    sleep(2.5)
```

# Online-Ressourcen