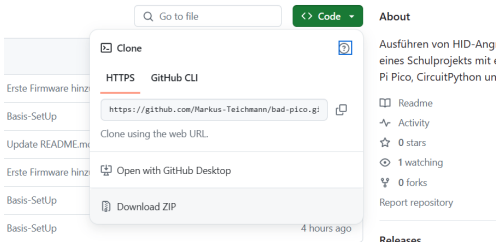
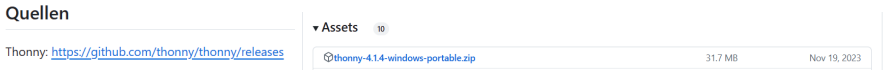
1 Entwicklungsumgbung

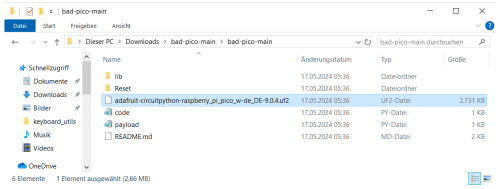
Um mit dem Pico arbeiten zu können braucht er einige Dateien, die die Grundlage bilden. Wir brauchen auch noch ein passendes Programm. In einem ersten Schritt gehen wir dazu auf die Website: `github.com/Markus-Teichmann/bad-pico`. Dieses Projekt laden wir als Zipdatei herunter.



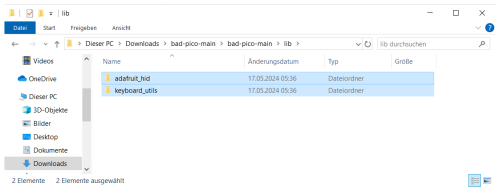
Unter dem Abschnitt Quellen findest du einen Link, der dich zur Downloadseite von thonny bringt. Folgen wir diesem Link und laden von dort die neuste portable Version herunter.



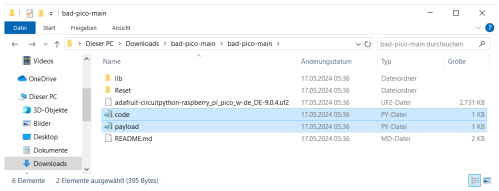
Starte nun eine Dateiverwaltung deiner Wahl und begebe dich mit dieser in den Ordner in welchen du die Zipdateien gespeichert hast. Entzippe die beiden Dateien und Klicke danach auf den bad-pico-main Ordner. Schließe dann den Pico an deinen Computer an und kopiere die hier markierte Datei in den Speicher deines Picos.



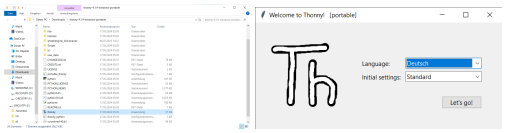
Dieser sollte rebooten und danach als CIRCUITPY wieder auftauchen. Anschließend navigierst du in den bad-pico-main/lib Ordner und kopierst die dortigen Ordner in den CIRCUITPY/lib Ordner.



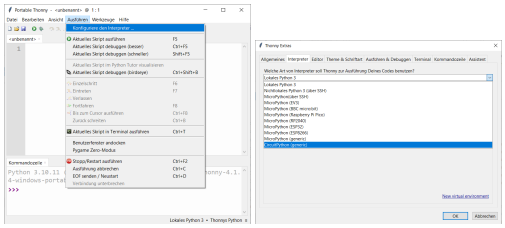
Abschließend sind noch folgende Dateien aus dem bad-pico-main Ordner nach CIRCUITPY zu kopieren.



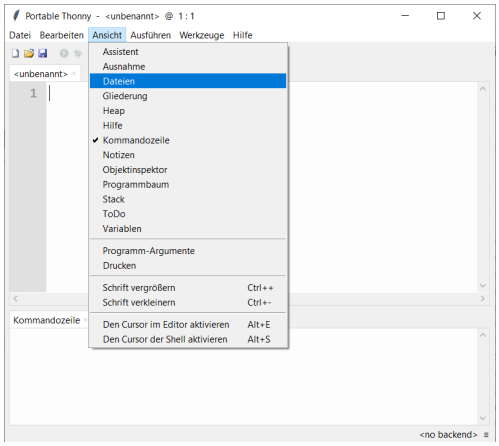
Dein Pico ist vollständig eingerichtet, es fehlt aber noch das passende Programm, um mit ihm zu arbeiten. Wechsle dazu in den thonny-4.1.4-windows-portable Ordner und führe die dortige Anwendung aus.



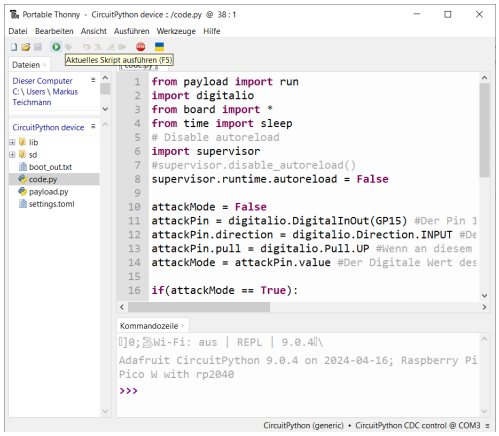
Damit wir mit thonny den Code des picos ausführen können, müssen wir noch den richtigen Interpreter wählen. Klicke dazu auf Ausführen -> Konfiguriere den Interpreter und wähle dann anstatt Lokales Python 3 im Drop-Down Menü CircuitPython (generic) aus.



Klicke auf ok. Um später eine bessere Ansicht auf die Dateien zu haben, klicken wir jetzt auf Ansicht und dann auf Dateien.

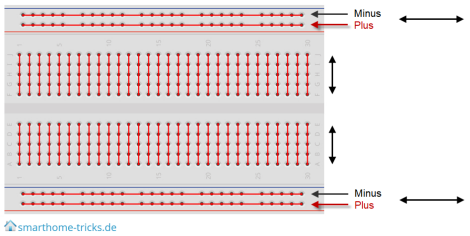


Abschließend werfen wir CIRCUITPY sicher aus, schließen alle Programme und ziehen das Kabel wieder aus dem USB-Port. Dann öffnen wir thonny erneut und verbinden das Kabel. Die Entwicklungsumgebung ist nun vorbereitet. Wenn wir Dateien ausführen wollen, so klicken wir auf den kleinen grünen Pfeil:

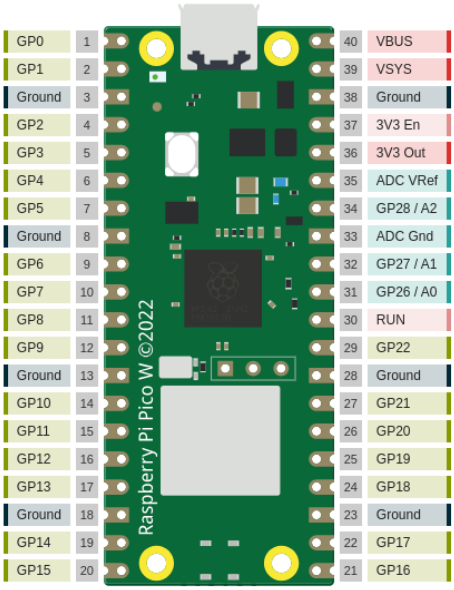


2 GPIO - General purpose Input and Output

Bei einem Steckbrett sind die äußeren Zeilen horizontal und die inneren spalten Vertikal miteinander verbunden:



Der Pico hat die folgende Pin-belegung:



Für uns wichtig sind heute nur die GP und die Ground Anschlüsse.

2.1 attackMode

Nachdem wir vorhin schon gesehen haben, dass wir den Pico auch über die USB-Schnittstelle programmieren werden ist es sinnvoll diese zuerst abzusichern, sodass wir uns nicht selber hacken. Dazu werden wir die GPIO-Pins verwenden und die Datei code.py aus dem Wurzelverzeichnisses des Picos um einige Zeilen ergänzen:

Aufgabe 2.1.1 attackMode

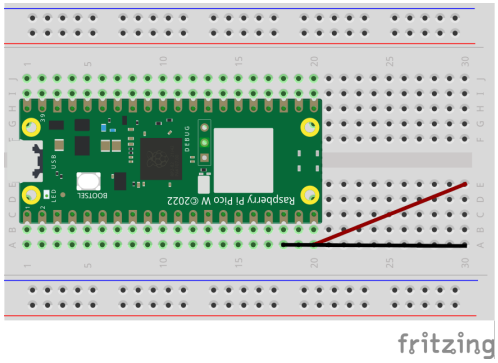
Informiere dich auf der Website `docs.circuitpython.org/en/latest/shared-bindings/digitalio` über die Aspekte:

- DigitalInOut
- Direction
- Pull.UP und Pull.Down
- value

des digitalio Modules. Kommentiere danach den unten stehenden Code, so dass Dritte den Code verstehen können und ergänze die code.py an der richtigen Stelle. Abschließend soll run() nur dann ausgeführt werden, wenn attackMode auf True steht. Implementiere diese Funktionalität.

```
10 attackMode = False
11 attackPin = digitalio.DigitalInOut(GP15)
12 attackPin = digitalio.Direction.Input
13 attackPin.pull = digitalio.Pull.UP
14 attackMode = attackPin.value
```

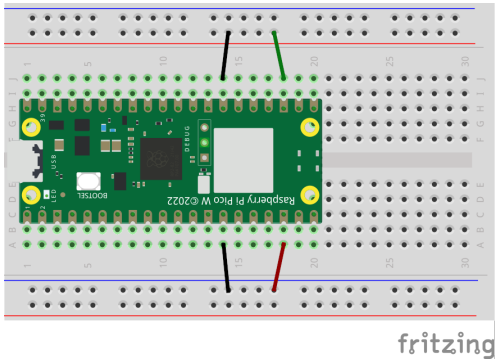
An dieser Stelle steht es dir frei mit den GPIOs weiter zu machen oder zu den HID-Angriffen zu springen.
Hier geht es mit den GPIOs weiter. Der in der Oben bearbeiteten Aufgabe soll dafür sorgen, dass wenn wir eine Verbindung des GP15 zu einem Ground herstellen, dass dann die Zeile mit dem Inhalt run() nicht läuft. Es gibt dafür viele mögliche Umsetzungen. Für eine eher simplere brauchen wir zwei Jumper-Kabel die wir wie folgt anschließen:



Sind nun beide Jumper-Kabel ordentlich gesteckt, so gibt es eine Verbindung zwischen GP15 und einem Ground. Folglich sollten wir sicher sein, sofern die Abfrage der Bedingung korrekt implementiert wurde.

2.2 LEDs

Zum Anschließen der LEDs brauchen wir die beiden LEDs und zwei 180 Ohm Widerstände. Diese sollten wie folgt gesteckt werden:



Dabei sind die zwei Widerstände in Schwarz, die rote LED in Rot und die grüne LED in Grün dargestellt.

Aufgabe 2.2.1 LED - Beleuchtung

Verwende die oben angegebene Pinaufstellung, um mit den aus der letzten Aufgabe bekannten Methoden die rote LED 10 mal blinken zu lassen, wenn die Zeile mit dem Inhalt run() ausgeführt wird und die grüne LED 10 mal blinken zu lassen, wenn sie nicht ausgeführt wird.
Tipp: `youtube.com/watch?v=fTeiPwYLw8w`

3 HID - Human Interface Devices und Bad USBs

Alle in dieser Woche gezeigten Angriffe machen sich eine grundlegende Schwachstelle zunutze. Computer vertrauen bei HID - Geräten wie Tastaturen und Mäuse noch immer darauf, dass an der Tastatur ein Mensch sitzt. Das muss aber bei weitem nicht so sein. Heute wollen wir uns genau diese Schwachstelle zunutze machen und ebenfalls auf diesem Weg Angriffe starten.

3.1 Wichtige Methoden für payload.py

```
1 sleep(10)
2 send(RUN, 1)
3 get_code('Hallo Welt!')
```

In der ersten Zeile befindet sich der sleep() Befehl. Dieser akzeptiert einen numerischen Wert, der eine Zeit in Sekunden beschreibt. Werte wie 0.5 oder 0.25 sind auch möglich. Der Befehl bewirkt, dass in dem gegebenen Zeitraum keine weiteren Keyboardeingaben gemacht werden. Das ist besonders dann hilfreich, wenn gerade darauf gewartet wird, dass ein Programm gestartet wird oder aus sonstigen Gründen gerade kein Input erwünscht ist.
An zweiter Stelle steht der send() Befehl. Er erhält zwei Parameter, wobei der zweite optional ist und angibt wie lange nach dem Drücken, der angegebenen Tasten geschlafen (sleep()) werden soll. Der erste Parameter gibt die Tasten an, die der Pico für uns drücken soll. In diesem Fall ist mit RUN die Windows Taste und r gemeint.
Zuletzt betrachten wir den get_code() Befehl. Dieser erwartet immer einen String und liefert einen für den Computer verständlichen Code zurück, der die entsprechenden Tasten drückt. Eine Kombination der letzten beiden Befehle der Form: send(get_code('Hallo Welt!')) ist also durchaus üblich.

3.2 Cheat Sheet - spezieller Tasten

Da nicht jede Taste einer Tastatur sich in einem Alphabet befindet, gibt es diese Liste solcher Tasten. Ihre Bedeutung ergibt sich häufig selbst.

- BACKSPACE
- DEL
- DOWNARROW
- END
- ENTER
- ESC
- HOME
- INSERT
- LALT
- LCONTROL
- LEFTARROW
- LGUI
- LSHIFT
- PAGEDOWN
- PAGEUP
- PAUSE
- PRTSCR
- RALT
- RCONTROL
- RGUI
- RIGHTARROW
- RSHIFT
- SPACE
- TAB
- UPARROW
- WIN
- SHIFT
- ALT
- CTRL
- COMMAND
- OPTION
- CTRLALTDEL
- RUN
- SPOTLIGHT
- CLOSE

Aufgabe 3.2.1 Attack on Python

Ergänze mit den oben genannten Befehlen und den Sondertasten, die Datei payload.py die du im CIRCUITPY Ordner findest, so dass beim Ausführen „notepad“ (der standard Texteditor von Windows) geöffnet wird und ein beliebiger Text in den Editor eingetragen wird.

Solltest du noch nicht an den GPIO Pins gearbeitet haben, so ist jetzt der richtige Zeitpunkt damit anzufangen. Ansonsten kannst du dir gerne auch ein weiteres Angriffsziel überlegen und dafür ein passendes Skript basteln.
Möglicherweise interessantes Projekt:
`youtube.com/watch?v=JON76zbiL1o`
Weitere interessante Quellen:
`github.com/dsymbol/ducky-payloads/tree/main/payloads`
`github.com/hak5/usbrubberducky-payloads/tree/master/payloads`