Leg los mit CircuitPython

Suchst du einen leichten Zugang in die Welt der Microcontroller und Elektronikprojekte? Oder bist du schon ein alter Hase und hast einfach Lust auf etwas Neues? Es gibt eine Menge Gründe, CirycuitPython auszuprobieren!

In diesem Artikel erfährst du, was CircuitPython ist und was du brauchst, um damit durchzustarten. Wir schreiben zusammen etwas Python-Code und lassen ein paar LEDs wilde Sachen machen.

Was ist CircuitPython?

Das New Yorker Unternehmen Adafruit hat im Sommer 2017 ihre „neue“ Programmiersprache vorgestellt: CircuitPython. Hierbei handelt es sich um einen sogenannten Fork von MicroPython, einer anderen Sprache, die ebenso für die Programmierung von Microcontrollern entwickelt wurde. Beide Programmiersprachen basieren auf Python 3 – **CircuitPython wurde jedoch mit dem Ziel entwickelt, besonders einsteigerfreundlich zu sein.**

Python gilt als leicht zu erlernende Programmiersprache und wird mittlerweile auch bereits in der Schule unterrichtet. Dabei ist Python nicht nur etwas für Einsteiger – auch komplexe Projekte im Bereich der künstlichen Intelligenz werden z.B. mit Python realisiert.

Welche Vorteile hat CircuitPython?

Abgesehen davon, dass Python relativ leicht zu lernen ist, hat der Einsatz von CircuitPython noch weitere Vorteile.

Vermutlich hast du bereits in der Arduino IDE Sketches in C++ geschrieben und auf dein Board geladen. Dann weißt du, dass jede Änderung im Code mit einem längeren Update des Boards verbunden ist: Der Sketch wird kompiliert und dann vollständig auf den Microcontroller geladen – was ganz schön lange dauern kann.

Anders bei CircuitPython: Wenn du dein Board per USB an deinen Computer anschließt, wird es als Laufwerk erkannt. Hierauf befindet sich der Python-Code z.B. in der Datei **code.py** – diese Datei kannst du theoretisch mit jedem Text-Editor öffnen und verändern. **Sobald du deine Änderungen gespeichert hast, resettet sich dein Board automatisch und läuft mit dem neuen Code – eine Sache von wenigen Sekunden.**

Der große Unterschied hierbei ist, dass C++ kompiliert wird und Python (also auch CircuitPython) interpretiert. Wie bitte? Um nicht den Rahmen dieses Artikel zu sprengen, lies bitte hier mehr über Kompilieren und Interpretieren, wenn dich das Thema interessiert.

Die REPL

Das klingt schon ziemlich schnell, aber es geht noch schneller. Wenn auf deinem Board CircuitPython läuft, kannst du mit ihm über die REPL (engl. Read–eval–print loop) kommunizieren. Das heißt, du kannst Befehle direkt an deinen Microcontroller senden, die dieser dann sofort ausführt.

Das ist besonders praktisch zum Debuggen, also um schnell mal auszuprobieren, ob eine LED richtig angeschlossen ist oder eine Funktion so funktioniert, wie du es dir vorgestellt hast. Mit der Arduino IDE und C++ kann das mitunter frustrierend

sein, da du auch für kleine Änderungen im Sketch alles wieder neu kompilieren und hochladen musst.

Später lernst du in diesem Artikel den **Mu Editor** kennen. Dann werden wir genauer auf die REPL eingehen.

Und die Nachteile?

Natürlich hat der Einsatz von CircuitPython auch Nachteile. Ein gewichtiger ist der Support mit Bibliotheken: Adafruit und die Community haben bereits für viele Sensoren und Bauteile passende Bibliotheken veröffentlicht und stellen diese selbstverständlich kostenfrei zur Verfügung. An die Menge, die dir in der Arduino IDE zur Verfügung steht, kommen sie aber **noch** nicht heran.

Da CircuitPython jedoch eine noch relativ junge Technologie ist, sind wir davon überzeugt, dass sich das im Laufe der Zeit ändern wird. Bereits jetzt veröffentlicht Adafruit regelmäßig neue Bundles mit Bibliotheken, die dir das Leben leichter machen.

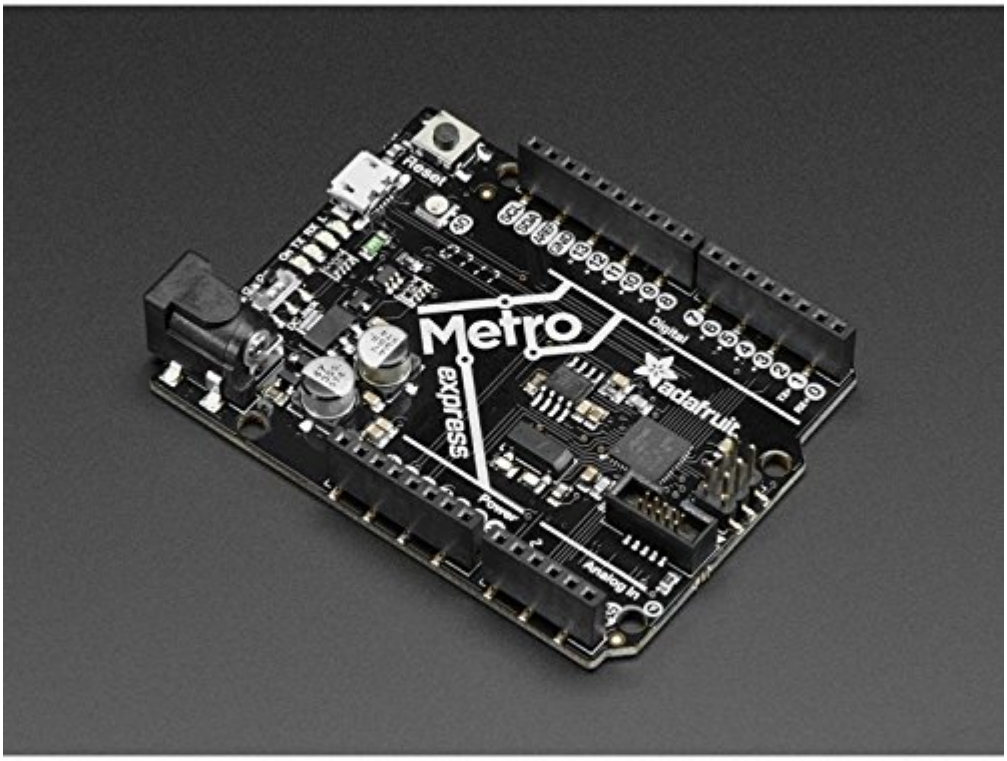
Stichwort Geschwindigkeit: Die Geschwindigkeit beim Entwickeln ist bei CircuitPython zwar höher, dafür können Programme jedoch langsamer laufen als es beim kompilierten Code in C++ der Fall ist. Aber dieser Unterschied dürfte bei den meisten Hobby-Anwendungen kaum ins Gewicht fallen.

Der passende Microcontroller

Der einfachste Weg, mit CircuitPython und Microcontrollern zu experimentieren, ist, ein speziell dafür ausgelegtes Board zu kaufen. Adafruit hat hier eine fast schon überwältigende Menge an verschiedenen Boards in vielen Größen und für viele Anwendungsbereiche produziert.

Du kannst CircuitPython aber auch auf einigen Arduino-Boards wie dem **Zero** oder dem **MKR Zero** zum Laufen bringen. In der Dokumentation von Adafruit erfährst du mehr darüber.

In diesem Artikel verwenden wir das Board **Metro M0 Express** von Adafruit. Es hat dieselben Maße wie ein Arduino UNO und verfügt über ähnlich viele Analog- und Digitalpins. Außerdem hat es 4 LEDs und einen NeoPixel an Bord und verfügt über 2MB Speicher. Genug Platz für deinen Code, Dateien und Bibliotheken.



Adafruit Metro M0 Express, entwickelt für CircuitPython, ATSAMD21G18 (3505)
39,99 €

Alle Boards, die CircuitPython unterstützen, findest du in einer Übersicht von Adafruit.

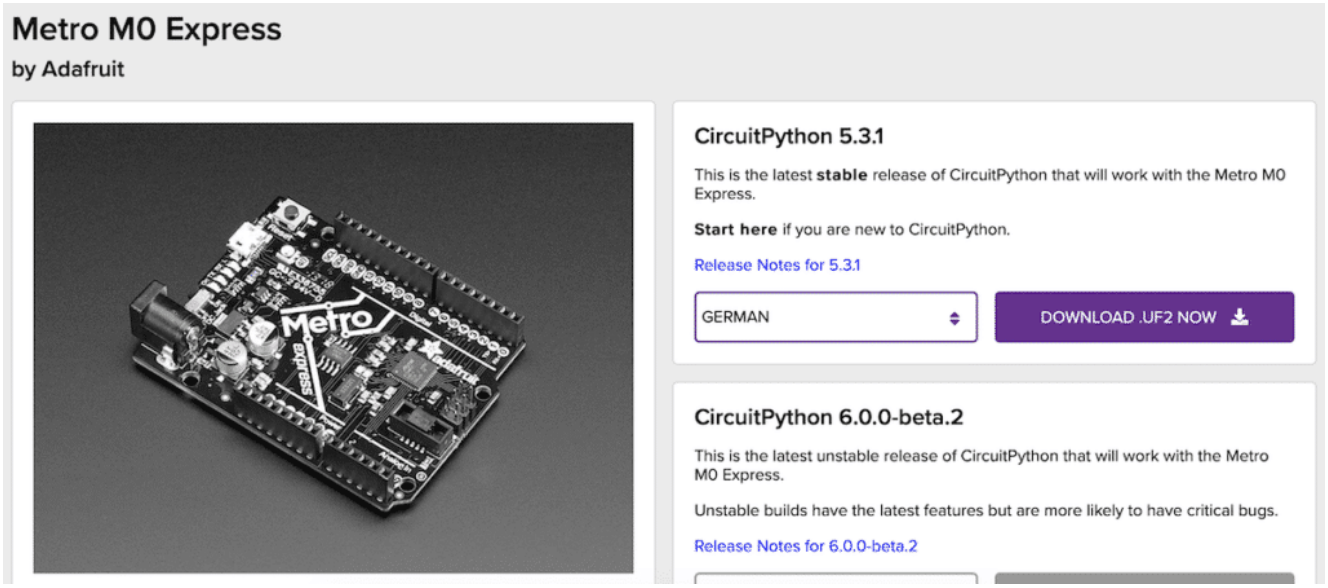
Vorbereitungen

Du hast dich entschlossen, mit CircuitPython loszulegen? Großartig! Bevor du anfängst, musst du allerdings noch etwas Zeit in Vorbereitungen stecken.

Zunächst solltest du dir die passende Firmware für dein Board, die aktuellen Bibliotheken und einen passenden Editor herunterladen und installieren.

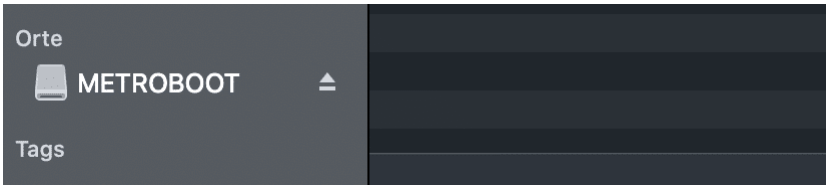
Die Firmware

Suche in der Übersicht von Adafruit nach deinem Board und klicke darauf. Anschließend siehst du die verfügbaren Versionen der Firmware. Achte darauf, die neueste **stable** Version herunterzuladen – in unserem Beispiel die Version 5.3.1



Schließe nachdem Download dein Board per USB an deinem Computer an. Als nächstes musst du es in den Boot-Modus versetzen: Das machst du, indem du zweimal kurz hintereinander den Reset-Button auf dem Board drückst.

Adafruit weist darauf hin, dass das in der „richtigen“ Geschwindigkeit erfolgen soll – wie auch immer, wenn es geklappt hat, findest du im Explorer (oder Finder auf deinem Mac) ein neues Laufwerk. Der Name des Laufwerks setzt sich aus dem Boardnamen und „BOOT“ zusammen. In unserem Fall heißt es also **METROBOOT**.



Ziehe nun die Datei, die du vorhin heruntergeladen hast, auf

dieses Laufwerk. Damit installierst du die Firmware. Nach wenigen Sekunden sollte das Laufwerk automatisch ausgeworfen werden und dafür ein anderes mit dem Namen **CIRCUITPY** erscheinen.

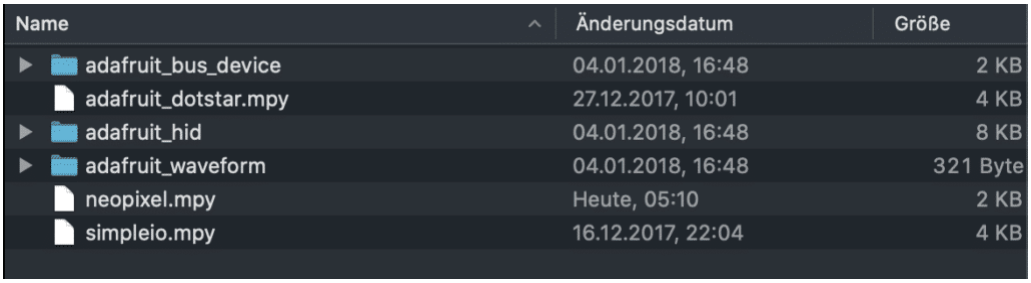
Wenn das der Fall ist, hat alles funktioniert und dein Board ist bereit. Falls es nicht klappt, wirf einen Blick in die Dokumentation von Adafruit – hier werden einige weitere Hilfestellungen gegeben.

Die Bibliotheken

Jetzt wo dein Board bereit ist, fehlen noch die Bibliotheken. Wie eingangs erwähnt, gibt es bereits für viele Anwendungen eine Bibliothek, die dir Arbeit abnimmt.

Adafruit bietet hier ganze Bundles an, die du dir hier herunterladen kannst. Lade dir von dort das Bundle herunter, das deiner Firmware-Version entspricht. Wenn du also eine Firmware in der Version 5.x installiert hast benötigst du das Bibliotheken-Bundle mit derselben Versionsnummer.

Entpacke die heruntergeladene Datei und öffne den Ordner. Im Unterordner **lib** findest du nun alle verfügbaren Bibliotheken. Suche dir die aus, die du benötigst und kopiere sie in den gleichnamigen Ordner des Laufwerks **CIRCUITPY**. Wenn dein Board genug Speicher besitzt, kannst du auch einfach alle Bibliotheken dort rüberkopieren.



Wenn du auf Probleme stößt, konsultiere die Dokumentation von Adafruit zum Thema Bibliotheken. Hier werden recht viele Sonderfälle besprochen.

Examples

Aus der Arduino IDE kennst du sicherlich die Beispiele, die den meisten Bibliotheken beiliegen. Diese sind recht nützlich, um herauszufinden, welche Funktionen eine Bibliothek bietet und wie du diese verwendest.

Diese Beispiele gibt es auch für CircuitPython. Auf der Seite, von der du das Bibliotheken-Bundle heruntergeladen hast, kannst du unter **Bundle Examples** eine ZIP-Datei mit Beispielen herunterladen.

Wie du diese Beispiele auf deinem Board ausführst, sehen wir gleich. Zunächst fehlt jedoch noch der passende Editor.

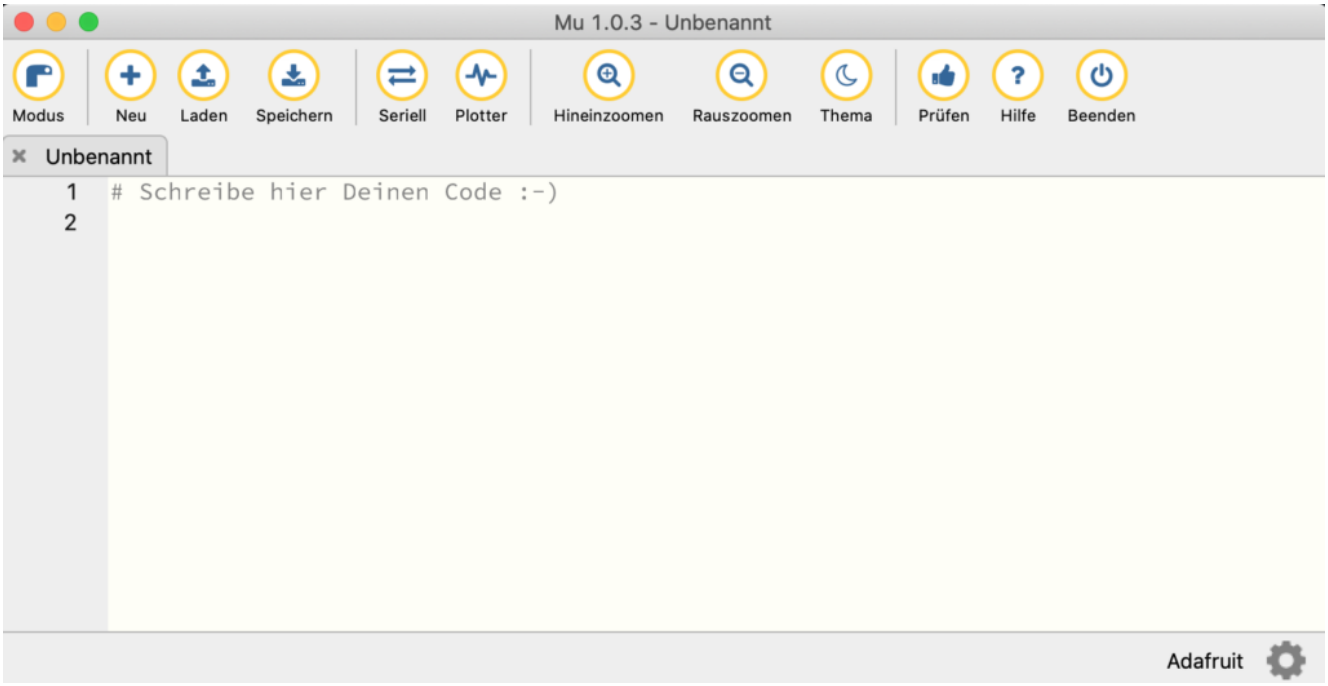
Der Mu Editor

Um Code in CircuitPython schreiben zu können, benötigst du natürlich einen Editor. Hier reicht im Prinzip der einfachste Text Editor – wir möchten dir jedoch den **Mu Editor** ans Herz legen. Der ist gratis, einsteigerfreundlich und für CircuitPython ausgelegt. So findest du darin z.B. einen Seriellen Monitor und Plotter wie du es vermutlich schon von der Arduino IDE gewohnt bist. Außerdem kannst du hier auch die oben beschriebene REPL problemlos verwenden.

Lade dir die aktuelle Version hier von der offiziellen Seite für dein Betriebssystem herunter.

Erste Schritte

Öffne nun den Mu Editor. Als nächstes schauen wir uns die wichtigsten Funktionen an. Danach schreibst du auch schon deinen ersten Code in Python und bringst ihn auf deinem Board zum Laufen.



Im oberen Bereich findest du eine Leiste mit Funktionen:

Modus: Hier kannst du einstellen, für welchen Zweck du den Mu Editor verwenden möchtest. Unserer heißt **Adafruit CircuitPython**.

Neu: Erstellt eine neue leere Datei.

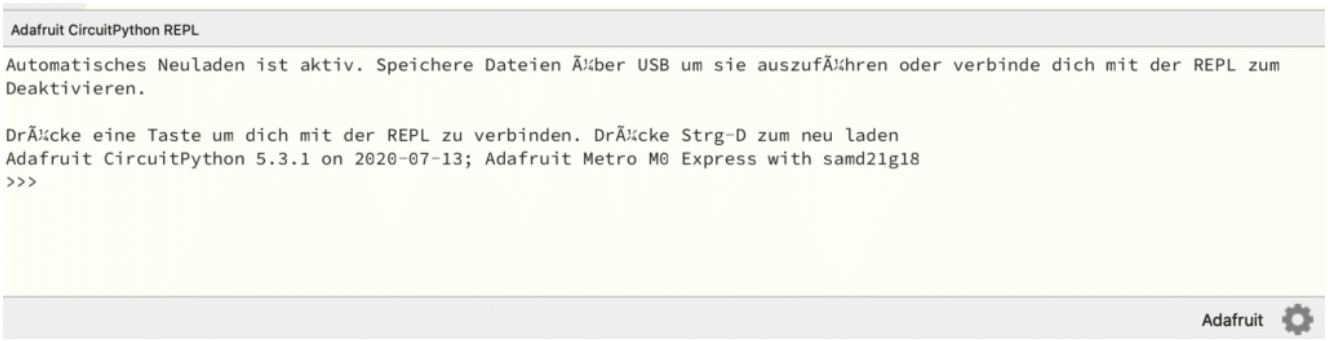
Laden & Speichern: Das ist sicherlich klar. Ein Hinweis vorab: Wenn dein Board deinen Code automatisch starten soll, musst du den Dateinamen **cody.py** verwenden.

Seriell & Plotter: Diese beiden kennst du sicherlich schon von der Arduino IDE. Hier kannst du wie gewohnt Daten einsehen und Statements ausgeben, die dir beim Debuggen helfen.

Prüfen: Hiermit kannst du deinen Code auf Fehler checken lassen.

Die REPL öffnen und verwenden

Um die REPL zu öffnen und direkt mit deinem Board interagieren zu können, öffne zunächst über den entsprechenden Button oben den Seriellen Monitor. Drücke anschließend gleichzeitig **Strg + C** und danach eine beliebige Taste. Nun öffnet sich die REPL:



Zeit für einen kurzen Test: Kopiere den folgenden Code, füge ihn in deiner REPL ein und drücke Enter:

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT
```

Jetzt kannst du direkt die Onboard-LED anschalten, indem du folgendes eingibst und wieder Enter drückst:

```
led.value = True
```

Ist sie angegangen? Dann schalte sie versuchsweise wieder aus, indem du **led.value** auf **False** setzt. Hinweis: Auf den meisten kompatiblen Boards kannst du die interne LED über den Pin **D13** ansteuern. Bei einigen ist es jedoch **D17** und einige wenige haben keine interne LED.

So speicherst du dein Programm auf dem Board

Die REPL ist toll, um schnell etwas zu testen. In den meisten Fällen wirst du jedoch ein reguläres Programm in CircuitPython schreiben und direkt auf deinem Board abspeichern.

Dein Board sucht automatisch nach einer Datei mit dem Namen **code.py**. Wenn es keine findet, hält es Ausschau nach der Datei

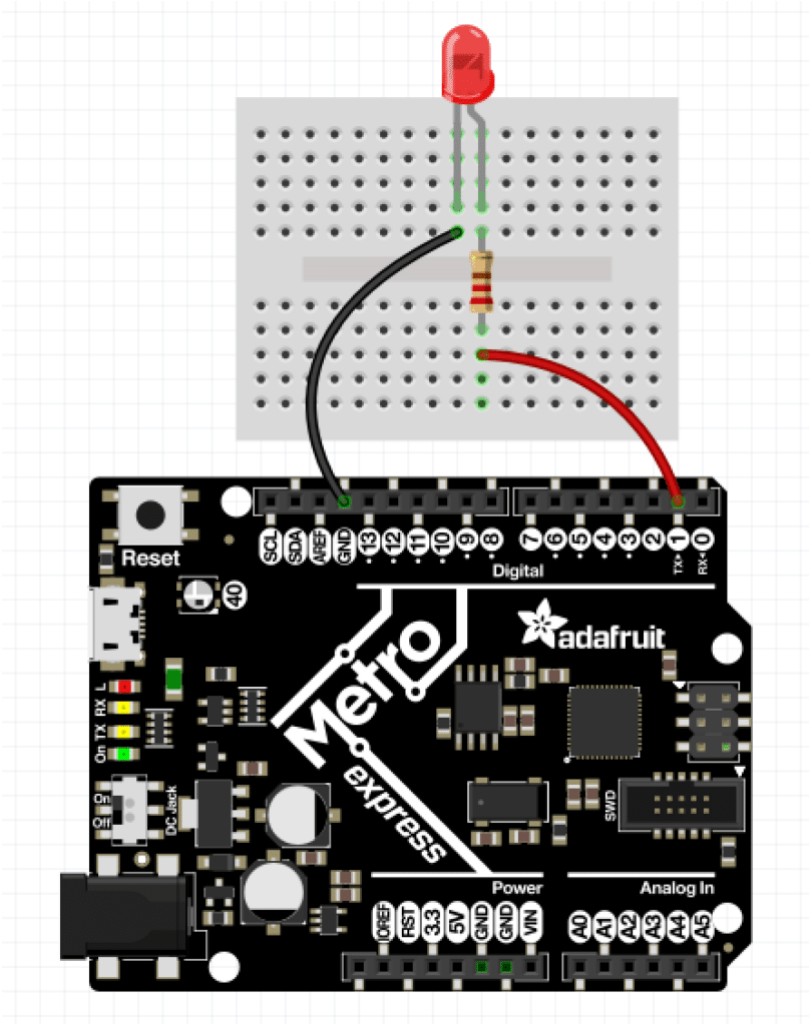
main.py und führt diese aus. **Heißt konkret:** Speichere das Programm, das ausgeführt werden soll unter dem Namen **code.py** direkt auf dem Laufwerk CIRCUITPY deines Boards ab.

Immer wenn dein Board hier eine Änderung feststellt – weil du den Code im Mu Editor aktualisiert und gespeichert hast – rebootet es und führt den neuen Code aus. Das ganze dauert in der Regel nur wenige Sekunden.

Dein erstes Programm in CircuitPython

Jetzt wird es Zeit, dein erstes Programm zu schreiben, auf deinem Board abzuspeichern und auszuführen. Auch für dieses Beispiel soll erst einmal eine blinkende LED reichen.

Schnapp dir ein Breadboard, eine LED und einen Vorwiderstand und baue folgendes auf:



Kopiere nun folgenden Code in eine leere Datei in deinem Mu Editor und speichere sie als **code.py** ab.

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D1)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Deine LED sollte nun im Halbsekundentakt an- und wieder ausgehen. Werfen wir nun einen Blick auf die einzelnen Teile

des Codes.

Zunächst bindest du drei Bibliotheken ein so wie du es aus der Arduino IDE kennst. Viele Bibliotheken sind in extra Dateien ausgelagert und müssen von dir in den Ordner **lib** auf deinem Board gespeichert werden, bevor du sie verwenden kannst.

Die drei Bibliotheken in diesem Beispiel sind jedoch Teil von CircuitPython. Das heißt, du musst nichts weiter tun, bevor du sie verwenden kannst, außer sie einzubinden:

```
import board
import digitalio
import time
```

Die Bibliothek **board** benötigst du für den Zugriff auf deine Hardware, **digitalio** kümmert sich um die Inputs und Outputs des Boards und mit **time** kannst du dein Programm „schlafen legen“ – ähnlich wie du das mit **delay()** in C++ machst.

Anschließend legst du den Pin der LED (D1) und seine „Richtung“ (OUPUT) fest. Auch das kennst du aus der Arduino IDE von der Funktion **pinMode()**.

```
led = digitalio.DigitalInOut(board.D1)
led.direction = digitalio.Direction.OUTPUT
```

Zuletzt erstellst du einen While-Loop, der ununterbrochen läuft. Hier schaltest du die LED mit **led.value = True** ein, wartest eine halbe Sekunde und schaltest sie für eine weitere halbe Sekunde wieder aus.

```
while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Spiele an den Werten etwas herum, um zu sehen, wie sich die

Intervalle der LED verändern. Wenn du dein Programm speicherst, sollten deine Änderungen nach wenigen Sekunden sichtbar werden.

Wie geht es weiter?

Du hast nun alles, was du brauchst, um mit CircuitPython loszulegen. Zeit für deine Experimente. Schaue als nächstes in den Bibliotheken nach, für welche deiner Bauteile und Sensoren du hier etwa passendes findest. Wirf anschließend einen Blick in die Beispiel-Codes, die jeder Bibliothek beiliegen und beginne deine Abenteuer mit CircuitPython!