KI & Maschinelles Lernen auf dem Mikrocontroller

Teil IB: Klassisches ML auf dem Circuit Python

Urheber: Natalia Jörg und Thomas Jörg, natalia@iludis.de, thomas@iludis.de

Stand 25. November 2024

```
import time
import board
from adafruit_circuitplayground import cp
#import simpleio
import digitalio
import analogio
import neopixel

cp.pixels.fill((0, 0, 0))
cp.pixels.show()

while True:
    cp.pixels.fill((10, 10

    with open("\color.csv"
        if cp.button_b:
            cp.pixels.fill

            cp.pixels[1]=(
            redRaw = cp.li
            time.sleep(0.1
            cp.pixels.fill(

            cp.pixels[1]=(0, 255, 0)
            greenRaw = cp.light
            time.sleep(0.1)
            cp.pixels.fill((0, 0, 0))

            cp.pixels[1]=(0, 0, 255)
            blueRaw = cp.light
            time.sleep(0.1)
```

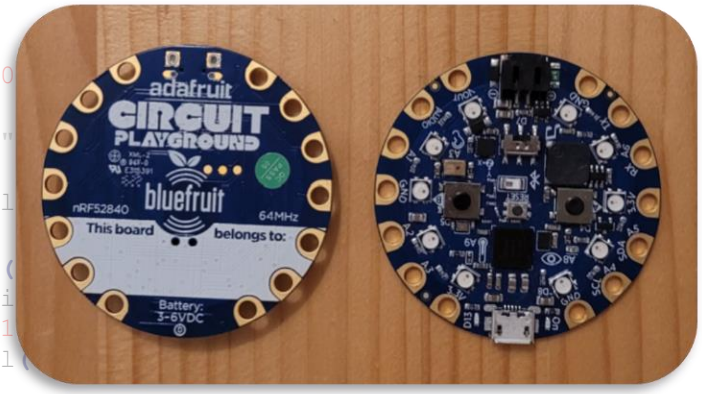


Abb. Eigenes Foto Circuit Playground

Warum MicroPython?

Um einem erfahrenen Informatiklehrer zu erklären, warum MicroPython gegenüber Arduino bei der Programmierung von Mikrocontrollern bevorzugt werden könnte, könnten folgende Punkte hilfreich sein:

1. Einfachheit und Lesbarkeit von Python: MicroPython basiert auf Python, einer Sprache, die für ihre Einfachheit und Lesbarkeit bekannt ist. Python wird oft als eine der einfachsten Sprachen für Anfänger angesehen, was für Lehrzwecke sehr vorteilhaft ist.

2. Schnelle Entwicklung: Aufgrund der Einfachheit von Python können Projekte schneller entwickelt werden. Dies ist besonders nützlich in Bildungsumgebungen, wo die Zeit oft begrenzt ist.

3. Interaktive Entwicklung: Micro Python unterstützt interaktive Befehlseingabe (REPL), was bedeutet, dass Schüler Code-Abschnitte ausführen und sofort Feedback erhalten können. Dies erleichtert das Experimentieren und Lernen.

4. Umfangreiche Bibliotheken und Community: Python verfügt über eine große Auswahl an Bibliotheken und eine starke Community. Dies kann besonders für fortgeschrittene Projekte von Vorteil sein, bei denen spezialisierte Funktionen oder Unterstützung erforderlich sind.

5. Plattformübergreifende Kompatibilität: Python ist eine plattformübergreifende Sprache, was bedeutet, dass die auf Micro Python basierenden Projekte leichter auf andere Python-kompatible Plattformen übertragbar sind.

6. Besser für fortgeschrittene Konzepte geeignet: Da Python eine vollwertige Programmiersprache ist, eignet sie sich besser zum Unterrichten fortgeschrittenerer Programmierkonzepte, die über das hinausgehen, was mit der Arduino-Skriptsprache möglich ist.

7. Zukunftsfähigkeit: Python ist eine der am meisten gefragten und verwendeten Programmiersprachen in der Industrie. Das Erlernen von Python durch die Verwendung von Micro Python kann den Schülern helfen, Fähigkeiten zu entwickeln, die direkt im Berufsleben anwendbar sind.

Vorbereitung von Hardware und Software

Einrichtung der Hardware

Wir arbeiten mit Micropython.

In unseren Beispielen verwenden wir das Board „Circuit Playground“ von Adafruit in zwei Versionen: Bluefruit und Express (Express gilt als veraltet und ist mittlerweile schwierig zu beschaffen). Beide basieren auf Circuit Python als eine Implementierung von MicroPython.

Folgende Vorbereitungen müssen getroffen werden (gilt für beide):

- Das Python-Betriebssystem als uf2-Datei herunterladen:

  Bluefruit: https://circuitpython.org/board/circuitplayground_bluefruit/
  Express: https://circuitpython.org/board/circuitplayground_express/
  Detaillierte Anleitung: https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/circuitpython

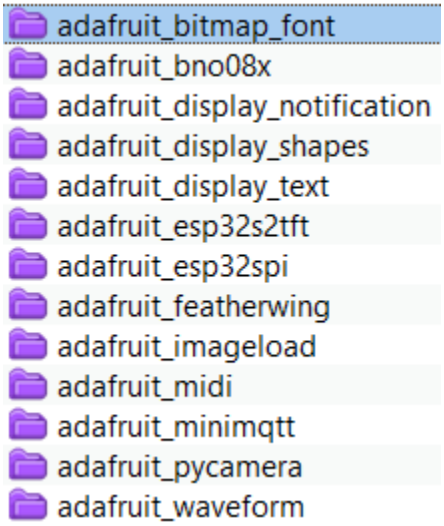
  ACHTUNG! Nicht mit „Bootloader verwechseln, ein Update ist normalerweise nicht nötig!

- Das Board mit Doppelklick auf den RESET-Knopf in den „OS-Modus“ versetzen – das Board erscheint dann im Windows Explorer als Laufwerk „CPLAYBTBOOT“. Die kleine rote LED (D13) leuchtet und der LED-Ring leuchtet grün.

- Jetzt kann die uf2-Datei auf das Laufwerk kopiert werden per Drag & Drop. Dies führt sofort zur Installation des Betriebssystems und zu einem Neustart. Nach dem Neustart geht das Board in den Programmiermodus und wird als „CIRCUITPY“-Laufwerk angezeigt.

- Es müssen nun Python-Libraries installiert werden. Diese findet man hier: https://circuitpython.org/libraries

  eine detaillierte Anweisung findet man hier: https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/circuit-playground-bluefruit-circuitpython-libraries
  Achtung! Kopiert man alle Libraries auf den Mikrocontroller, so fehlt Speicherplatz. Man sollte also vorher einige überflüssige Libraries löschen, zum Beispiel diese hier (weil es sich um externe Sensoren handelt, die so noch nicht angeschlossen sind):



- **Hinweis:** Bei älteren Mikrocontrollern kann ein Update des Bootmanagers notwendig sein.
  Quelle für Updates: https://github.com/adafruit/Adafruit_nRF52_Bootloader/releases/tag/0.8.2
  Anleitung für das Update: https://learn.adafruit.com/adafruit-circuit-playground-bluefruit/update-bootloader-use-command-line

Verwendete Editoren

Für die Programmierung in CircuitPython wird der Einsatz von Mu-Editor empfohlen.

Das Training kann mit Jupyter Notebooks stattfinden.

Die oben genannten Tools können auch als portable Versionen aus dem Edupyter-Paket verwendet werden.

Quelle: https://www.edupyter.net/

Hinweis: Beide Szenarien funktionieren auf beiden Mikrocontrollern, sie sind nur beispielhaft für die jeweiligen Exemplare ausformuliert.

Szenario 1: (Circuit Playground Express, eher veraltet)

Wird vor dem Mikrocontroller die Hand gehalten – messen sowohl der Licht- als auch der Temperatursensor die Änderungen. Im einfachsten Beispiel sind die Trainingsdaten so aufgenommen, dass die Entscheidung, ob Hand vor dem Mikrocontroller gehalten wird oder nicht, nur anhand der Temperatur getroffen werden kann.

Das Projekt besteht aus den folgenden Dateien:

- *boot.py* - wechselt in Abhängigkeit von der Stellung des Schiebereglers in den Datenerfassungsmodus
- *code_aufnahme.py* - speichert die Sensordaten in Form einer csv-Datei
- *code_prediction.py* - wertet die Sensordaten nach Modell aus: z.B. Hand = grünes Licht, keine Hand = blaues Licht
- *decTreeClassifier.py* - Während des Trainings generiertes Modell
- *decTree_train.py* - Training des Modells

Hinweise zum Umgang mit Circuit Playground

1) Mit dem kleinen Schieberegler lässt sich der Schreibzugriff auf den Mikrocontroller steuern.

   Steht der Regler auf „Ground“ ⊖ - ist der Schreibzugriff vom Betriebssystem aus (somit auch über Mu-Editor) gesperrt. In diesem Fall kann das vorher hochgeladenes Programm die Daten aufnehmen und in eine csv-Datei im Speicher vom Mikrocontroller schreiben.

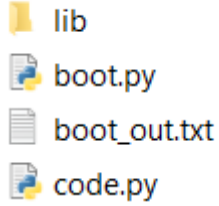
   Um neues Programm, z.B. zur Inferenz, auf den Mikrocontroller zu speichern, stellt man den Regler wieder auf „Plus“ ⊕ .

2) Circuit Playground kann nur zwei Dateien ausführen: *boot.py* und *code.py*. Das bedeutet, dass o.g. Programme wie *code_aufnahme.py* oder *code_prediction.py* müssen auf dem Mikrocontroller jeweils in *code.py* umbenannt werden.
   Die Datei *boot.py* wird nur einmalig beim Anschließen der Stromversorgung oder beim Reset ausgeführt - vergleichbar mit setup() Funktion vom Arduino.

   Die Datei *decTreeClassifier.py* wird im Skript *code_prediction.py* importiert und muss daher auch auf das Laufwerk vom Circuit Playground (CIRCUITPY) kopiert werden.

Skript zum Einstellen des Schreibzugriffs

Schieberegler steht in Richtung “Ground”.

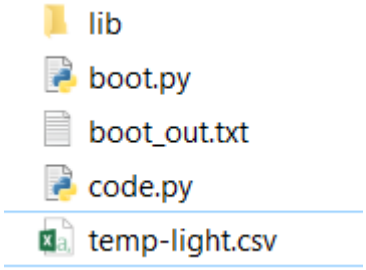


Inhalt der boot.py:

```
import storage
from adafruit_circuitplayground.express import cpx
storage.remount("/", cpx.switch)
```

Inhalt der code.py:

```
import time
from adafruit_circuitplayground.express import cpx
cpx.pixels[1] = (0, 0, 90)  # Pixel 1 auf blau als Anzeige während der Sammlung von
Daten
num_readings = 50  # wie viele Werte man aufnehmen möchte
try:
    with open("/temp-light.csv", "a") as fp:
        fp.write('Temperature, Light Intensity\n')  # Überschriftenzeile
        for x in range(0, num_readings):
            temp = cpx.temperature
            fp.write(str(temp) + "," + str(cpx.light) + "\n")
            time.sleep(2)     # Warte 2 Sekunden
            if cpx.button_a:
                break
        cpx.pixels[1] = (0, 90, 0)  # Pixel 1 auf grün wenn fertig
```



Daten aufbereiten

Bei der Aufnahme der Daten ist darauf zu achten, dass die Werte der jeweiligen **Klassen nicht vermischt** werden.
Bei erneuter Ausführung wird die csv-Datei nicht ersetzt, sondern erweitert.
Ergänzt durch die Zugehörigkeit der Klasse, sieht die fertige csv-Datei wie folgt aus:

temperature, light, class	25.6062,44,0	28.6031,191,1	29.8815,34,1	23.4333,293,0
26.2484,2,0	25.6946,8,0	28.6486,33,1	29.9504,92,1	23.5205,11,0
26.2041,3,0	25.7389,11,0	28.7394,4,1	29.8585,166,1	23.5205,312,0
26.2041,3,0	25.6283,3,0	28.8075,61,1	29.8815,162,1	23.4988,312,0
26.2041,3,0	25.6283,4,0	28.8302,192,1	29.8815,55,1	23.5641,4,0
26.2041,3,0	25.6725,3,0	28.9667,3,1	29.8815,201,1	23.6078,312,0
25.2312,3,0	25.6283,3,0	29.0349,114,1	29.8815,126,1	23.5859,312,0
25.2532,2,0	25.6946,2,0	29.0349,78,1	30.6201,7,1	23.6296,57,0
25.2091,3,0	26.0488,4,0	29.0577,223,1	30.5969,3,1	23.6951,296,0
25.2532,4,0	25.8273,4,0	29.1718,120,1	30.5969,13,1	23.6078,4,0
25.2532,3,0	25.8052,3,0	29.2173,207,1	30.4811,6,1	23.6514,220,0
25.2312,2,0	25.761,2,0	29.2629,114,1	30.5275,8,1	23.6514,312,0
25.2972,47,0	25.7831,3,0	29.4,157,1	30.458,14,1	23.6732,6,0
25.2753,145,0	25.761,2,0	29.3772,15,1	30.4117,16,1	23.6732,312,0
25.3413,131,0	25.8273,3,0	29.4688,221,1	30.2963,4,1	23.7169,188,0
25.3855,4,0	25.761,3,0	29.4916,6,1	30.3655,25,1	23.7169,53,0
25.3855,89,0	25.8052,3,0	29.5603,212,1	30.4117,12,1	23.8042,94,0
25.3634,47,0	28.2411,115,1	29.5833,250,1	30.4811,84,1	23.7388,312,0
25.4517,158,0	28.2863,23,1	29.652,131,1	30.5043,4,1	23.826,312,0
25.4296,39,0	28.3088,9,1	29.7437,166,1	30.6432,109,1	23.8042,54,0
25.4296,308,0	28.3315,72,1	29.6978,275,1	30.7361,49,1	23.826,9,0
25.5178,308,0	28.3767,86,1	29.8125,187,1	30.6897,101,1	23.8042,57,0
25.562,16,0	28.3993,237,1	29.8585,15,1	30.7129,18,1	
25.5841,8,0	28.4446,14,1	29.9044,166,1	30.9932,105,1	
25.6283,12,0	28.5126,150,1	29.9275,9,1	23.3898,12,0	

Training des ML Modells mit sklearn und Konvertierung in Python mit m2cgen:


```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import m2cgen as m2c
df = pd.read_csv('temp-light.csv',sep=',')
df.head(2)
features = ['temperature','light']
X = df[features]
y = df['class']
clf = DecisionTreeClassifier().fit(X, y)
predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")
model_to_python = m2c.export_to_python(clf)
print(model_to_python)
with open('decTreeClassifier.py', 'w') as file:
 file.write(model_to_python)
```

Im einfachsten Fall sieht die erzeugte Python-Datei so aus:

```
def score(input):
    if input[0] <= 27.578700065612793:
        var0 = [1.0, 0.0]
    else:
        var0 = [0.0, 1.0]
    return var0
```

Circuit Python-Code mit DecisionTree

- lib
- boot_out.txt
- code.py
- decTreeClassifier.py

```
import time
import decTreeClassifier as dt
from adafruit_circuitplayground.express import cpx

cpx.pixels.brightness = 0.2

while True:
    temp = int(cpx.temperature)
    licht = int(cpx.light)
    input = [temp, licht]
    pred = dt.score(input)
    print("prediction result: {}".format(int(pred[1])))
    if pred[1] < 1:
        cpx.pixels.fill((0, 0, 255)) # 0 - keine Hand
    else:
        cpx.pixels.fill((0, 255, 0)) # 1 - Hand wird erkannt
    cpx.pixels.show()
    time.sleep(0.5)
```

Szenario 2: (Circuit Playground Bluefruit)

Auch ohne spezielle Farbsensoren kann die Farbe über die Intensität der Reflexion des jeweiligen Farbkanals erkannt werden. Man schaltet den benachbarten Pixel hintereinander auf maximal Helligkeit des jeweiligen Farbkanals und misst mit dem einfachen Lichtsensor die Helligkeit des reflektierten Lichts.

Das Projekt besteht aus den folgenden Dateien:

- *boot.py* - wechselt in Abhängigkeit von der Stellung des Schiebereglers in den Datenerfassungsmodus
- *code_aufnahme.py* - speichert die Sensordaten in Form einer csv-Datei
- *code_prediction.py* - wertet die Sensordaten nach Modell aus: z.B. blaues Objekt = blaues Licht, kein Objekt = kein Licht
- *decTreeClassifier.py* - Während des Trainings generiertes Modell
- *decTree_train.py* - Training des Modells

Physikalischer Hintergrund:

1. *Wird ein roter Körper mit weißem Licht angestrahlt, so reflektiert er hauptsächlich die roten Wellenlängen und absorbiert die anderen Wellenlängen.*
2. *Wird der gleiche, bei weißem Licht rote Körper mit grünem Licht beleuchtet, so ist er schwarz, da er die grünen Wellenlängen des Beleuchtungslichts komplett absorbiert.*
3. *Wird der gleiche, bei weißem Licht rote Körper mit z.B. gelbem oder blauem Licht beleuchtet, so wird er, abhängig davon welche der auftreffenden Wellenlängen absorbiert bzw. reflektiert werden, in einem anderen Farbton erscheinen als bei weißem Licht.*

Quelle: https://lehrerfortbildung-bw.de/st_digital/medienkompetenz/gestaltung-farbe/physik/subtrakt/sub-lich/

Hinweise zum Umgang mit Circuit Playground

3) Mit dem kleinen Schieberegler lässt sich der Schreibzugriff auf den Mikrocontroller steuern.

   Steht der Regler auf „Ground“ ⊖ - ist der Schreibzugriff vom Betriebssystem aus (somit auch über Mu-Editor) gesperrt. In diesem Fall kann das vorher hochgeladenes Programm die Daten aufnehmen und in eine csv-Datei im Speicher vom Mikrocontroller schreiben.

   Um neues Programm, z.B. zur Inferenz, auf den Mikrocontroller zu speichern, stellt man den Regler wieder auf „Plus“ ⊕ .

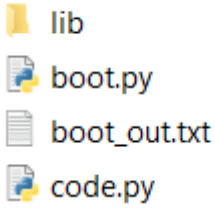
4) Circuit Playground kann nur zwei Dateien ausführen: *boot.py* und *code.py*. Das bedeutet, dass o.g. Programme wie *code_aufnahme.py* oder *code_prediction.py* müssen auf dem Mikrocontroller jeweils in *code.py* umbenannt werden.

   Die Datei *boot.py* wird nur einmalig beim Anschließen der Stromversorgung oder beim Reset ausgeführt -vergleichbar mit setup() Funktion vom Arduino.

   Die Datei *decTreeClassifier.py* wird im Skript *code_prediction.py* importiert und muss daher auch auf das Laufwerk vom Circuit Playground (CIRCUITPY) kopiert werden.

Daten sammeln

Schieberegler steht in Richtung “Ground”.

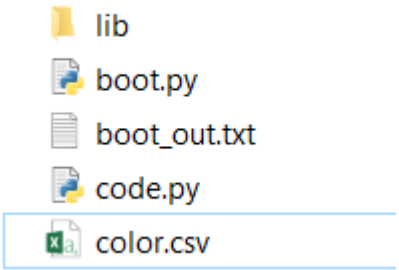


Inhalt der boot.py:

```
import storage
from adafruit_circuitplayground import cp
storage.remount("/", cp.switch)
```

Inhalt der code.py:

```
import time
import board
from adafruit_circuitplayground import cp
#import simpleio
import digitalio
import analogio
import neopixel

cp.pixels.fill((0, 0, 0))
cp.pixels.show()

while True:
    cp.pixels.fill((10, 10, 10))

    with open("\color.csv", "a") as fp:
        if cp.button_b:
            cp.pixels.fill((0, 0, 0))

            cp.pixels[1]=(255, 0, 0)
            redRaw = cp.light
            time.sleep(0.1)
            cp.pixels.fill((0, 0, 0))

            cp.pixels[1]=(0, 255, 0)
            greenRaw = cp.light
            time.sleep(0.1)
            cp.pixels.fill((0, 0, 0))

            cp.pixels[1]=(0, 0, 255)
            blueRaw = cp.light
            time.sleep(0.1)
            cp.pixels.fill((0, 0, 0))


            fp.write(str(redRaw) + "," + str(greenRaw) + ","
+ str(blueRaw) + str("\n"))
            time.sleep(0.2)
```



Daten aufbereiten

In diesem Fall müssen die aufgenommene RGB-Werte mit der zugehörigen Klasse ergänzt werden. Wir haben für jede Farbe eine gesonderte csv-Datei erstellt, um das Labeln zu erleichtern. Anschließend werden alle Werte in einer csv-Datei (`'color_gelabelt.csv'`) zusammengefasst.

r,g,b,c	15,96,187,0	171,148,22,1	125,58,20,1	5,5,4,2
9,82,139,0	10,97,194,0	164,168,21,1	112,40,13,1	2,2,3,2
11,118,182,0	14,80,186,0	217,272,31,1	103,38,14,1	4,5,4,2
15,150,236,0	10,82,180,0	142,65,4,1	129,22,6,1	2,2,2,2
15,154,200,0	12,93,172,0	203,132,9,1	82,10,5,1	3,2,2,2
17,158,241,0	13,92,194,0	180,104,3,1	77,12,6,1	5,4,5,2
15,157,206,0	10,102,184,0	179,201,37,1	64,10,5,1	5,5,5,2
16,163,232,0	12,90,176,0	228,299,56,1	58,11,6,1	5,5,5,2
15,147,195,0	11,96,187,0	233,253,27,1	67,12,6,1	5,4,4,2
16,136,200,0	14,93,199,0	215,181,15,1	81,19,10,1	5,4,5,2
13,165,209,0	12,93,202,0	203,143,4,1	140,74,31,1	5,5,5,2
14,151,217,0	14,91,213,0	198,137,3,1	214,99,37,1	4,5,5,2
9,50,106,0	10,105,205,0	114,25,11,1	226,93,43,1	5,5,5,2
	14,100,219,0	182,79,27,1	220,74,34,1	4,5,5,2
	10,24,4,1	147,63,27,1	184,66,26,1	
	186,153,23,1	154,66,24,1	189,81,29,1	

Training des ML Modells mit sklearn und Konvertierung in Python mit m2cgen:


```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import m2cgen as m2c
df = pd.read_csv('color_gelabelt.csv',sep=',')
df.head(1)
features = ['r','g','b']
X = df[features]
y = df['c']
clf = DecisionTreeClassifier().fit(X, y)
predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")
model_to_python = m2c.export_to_python(clf)
print(model_to_python)
with open('decTreeClassifier.py', 'w') as file:
 file.write(model_to_python)
```

In diesem Fall erzeugte Python-Datei des Models sieht so aus:

```
(Inhalt der Datei decTreeClassifier.py)


def score(input):
    if input[0] <= 82.5:
        if input[1] <= 43.5:
            var0 = [0.0, 0.0, 1.0]
        else:
            var0 = [1.0, 0.0, 0.0]
    else:
        var0 = [0.0, 1.0, 0.0]
    return var0
```

Circuit Python-Code mit DecisionTree

Schieberegler steht in Richtung “Plus”.

- lib
- boot_out.txt
- code.py
- decTreeClassifier.py

```
import time
import decTreeClassifier as dt
from adafruit_circuitplayground import cp

r = 0
g = 0
b = 0

while True:
    cp.pixels[0]=(255, 0, 0)
    r = cp.light
    time.sleep(0.5)
    cp.pixels.fill((0, 0, 0))

    cp.pixels[0]=(0, 255, 0)
    g = cp.light
    time.sleep(0.5)
    cp.pixels.fill((0, 0, 0))

    cp.pixels[0]=(0, 0, 255)
    b = cp.light
    time.sleep(0.5)
    cp.pixels.fill((0, 0, 0))

    input = [r, g, b]
    print(input)
    pred = dt.score(input)
    print("prediction result: ",pred)
    if pred[0] > 0:
        cp.pixels.fill((0, 0, 128)) # 0 - Objekt ist blau
    if pred[1] > 0:
        cp.pixels.fill((128, 128, 0)) # 1 - Objekt ist gelb/orange
    if pred[2] > 0:
        cp.pixels.fill((0, 0, 0)) # 2 - kein Objekt vor dem Sensor
    cp.pixels.show()
    time.sleep(1)
```