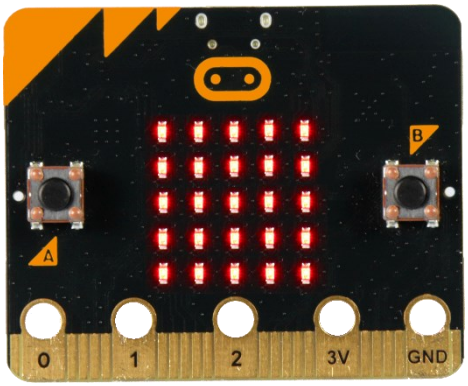
Digitales Lehren & Lernen
Didaktik der Informatik

Mikrocontroller-Programmierung mit micro:bit

Fortsetzung für Fortgeschrittene

Von der block- zur textbasierten Programmierung



Andreas Walter
Julian Flach

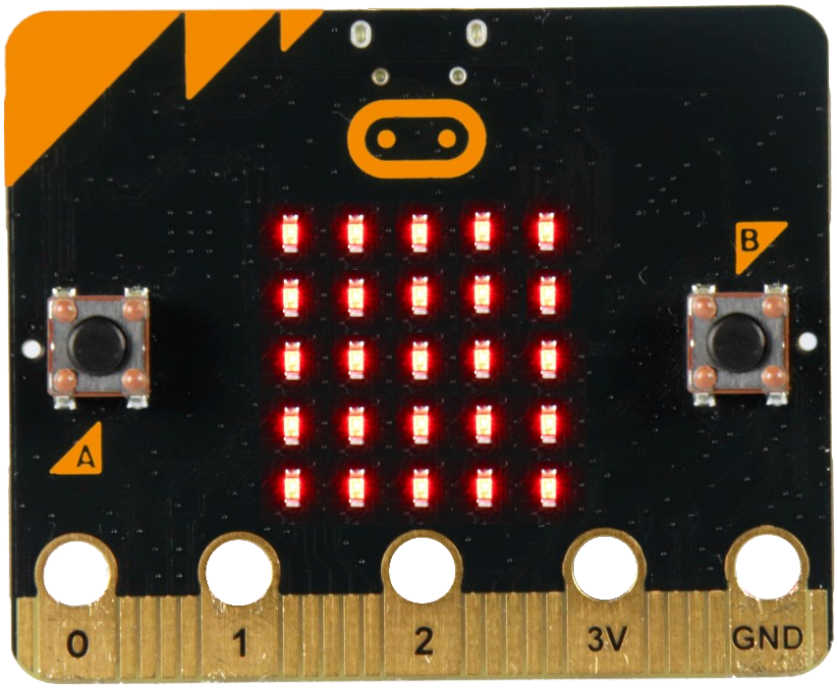
Digitales Lehren und Lernen &
Didaktik der Informatik
Universität Bayreuth
dlldi.uni-bayreuth.de

Dieses Skript gehört:

_____

Version: 01.02.2024 10:24 Uhr

Mikrocontroller-Programmierung mit micro:bit
Fortsetzung für Fortgeschrittene



Digitales Lehren und Lernen &
Didaktik der Informatik
Universität Bayreuth
dlldi.uni-bayreuth.de

Sicherheitshinweise

Lies dir die folgenden Punkte genau durch! Es wird an manchen Stellen im Skript noch einmal ausdrücklich auf wichtige Aspekte hingewiesen. Diese Hinweise solltest du bereits auf einem gesonderten Blatt bekommen und unterschrieben (Foto oder Scan) an uns geschickt haben.

Die Verwendung des BBC micro:bit ist einfach. Er ist jedoch so konzipiert, dass alle elektronischen Teile offen zugänglich sind. Bei einer sachgemäßen Verwendung mit Sorgfalt und Vorsicht ist die Benutzung aber ungefährlich. Deshalb beachte die folgenden Hinweise:

1. Anleitung befolgen
   Halte dich bei den folgenden Experimenten an die Anleitung. Beachte die angegebenen Schaltungen beim Aufbau der Experimente.

2. Stromversorgung
   Verwende nur das von uns gelieferte Batteriepack (und die Batterien) oder das von uns mitgelieferte USB-Kabel für die Stromversorgung deines micro:bit. Schließe auf keinen Fall beides gleichzeitig an!

   Verwende keinen portablen Batterie-Ladegeräte oder USB-Netzteile, da eine Überspannung den micro:bit zerstören könnte.

3. Lass deinen micro:bit nicht unbeaufsichtigt an der Stromversorgung stecken

4. Schaltungen stromfrei aufbauen
   Verändere keine Schaltungen, während der micro:bit mit der Stromversorgung verbunden ist. Trenne ihn vorher vom Strom.

5. Verwende nur die von uns mitgelieferten bzw. im Experiment beschriebenen Anschlussteile

6. Beschädigter micro:bit
   Falls dein micro:bit beschädigt ist, kontaktiere uns per E-Mail (Seite 5). Schließe ihn nicht mehr an die Stromversorgung an!

7. Vermeide heiße und kalte Temperaturen, lass den micro:bit nicht in der Sonne liegen

8. Bringe den micro:bit niemals mit Wasser oder nassen Händen in Berührung!

9. Vermeide Kurzschlüsse
   Berühre den micro:bit nicht mit metallischen Gegenständen, außer es wird ausdrücklich in der Anleitung verlangt (z. B. mit Krokodilklemmen).

   Verbinde niemals direkt einen Plus- mit einem Minuspol ohne einen „Verbraucher“ dazwischen! Dies könnte zu einem Kurzschluss führen und den micro:bit oder deinen Computer beschädigen.

10. Berühren der Kontaktflächen
    Der Strom, der an den Kontaktflächen anliegt, ist so gering, dass man ihn überhaupt nicht spürt! Anders als Strom aus der Steckdose ist hier die Spannung so gering, dass sie für gesunde Menschen ungefährlich ist. Berühre die Kontaktflächen trotzdem nur dann, wenn es ausdrücklich in der Anleitung beschrieben ist.

11. Trenne Kabelverbindungen nur, indem du am Stecker und nicht am Kabel ziehst

Weitere Hinweise zur Verwendung finden sich im „safety guide“ des micro:bit.

Allgemeine Informationen

Herzlich Willkommen im Kurs
„Mikrocontroller-Programmierung mit micro:bit –
Fortsetzung für Fortgeschrittene“

Für Kursinformationen sieh bitte
an dieser Stelle im gedruckten Skript nach!

- Falls du Fragen hast:
  - Stelle sie uns bei einem unserer Termine.
  - Nutze das Forum im E-Learning-Kurs.
  - Schreibe uns eine E-Mail.
- In diesem Skript sind immer wieder Lösungen angegeben. Deine Lösungen müssen nicht immer genauso aussehen, um zu funktionieren. Oft gibt es mehrere Wege, die ans Ziel führen!
- Bitte verwende den Rücksendeaufkleber und sende das Hardware-Paket nach dem Workshop an uns zurück (siehe Datum oben)! Dieses Skripts darfst du behalten.

Inhalte

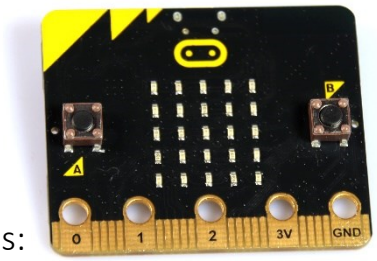
Experimente

Hinweis

Falls du bereits den ersten Teil dieser Workshop-Reihe („*für Einsteiger*“) besucht hast, kannst du nach dem Spiel auf der nächsten Seite direkt zu Seite 12 gehen.

Unboxing und erster Überblick

Was ist der micro:bit überhaupt?



- Öffne das Paket, dass du von uns erhalten hast!
- Darin befindet sich der micro:bit. Er sieht ungefähr so aus:

- Schieße das mitgelieferte Batteriepack auf der Rückseite an.

Achtung!

Beachte auf Seite 4: „2. Stromversorgung“

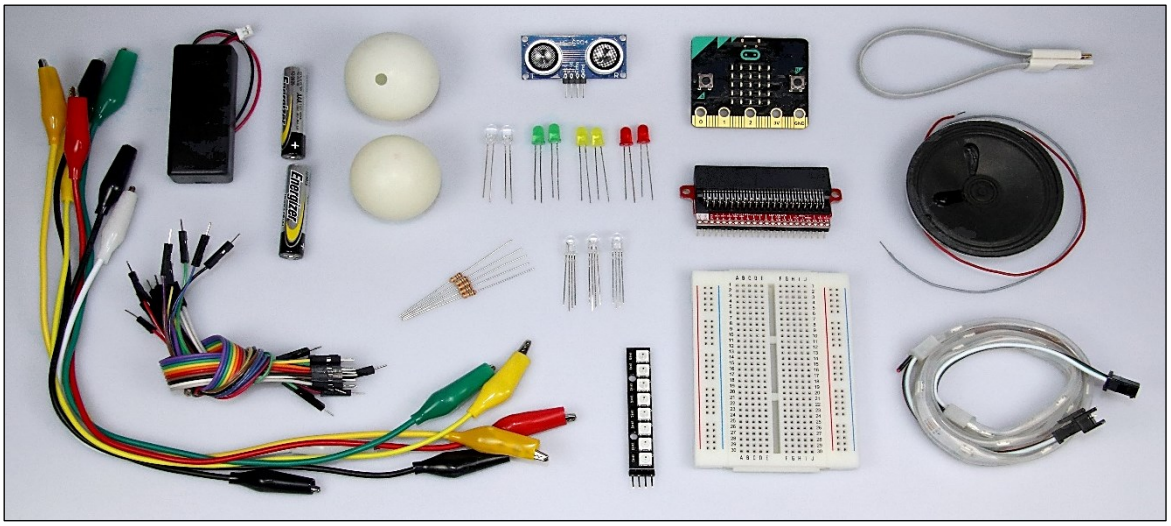
Spiel „micro:Bingo“

Auf dem micro:bit befindet sich eine kleine Abwandlung von „Bingo“:
Auf der LED-Matrix der Vorderseite werden zufällig Punkte angezeigt. Ziel ist es, alle Punkte in einer Reihe oder einer Diagonale zu haben.

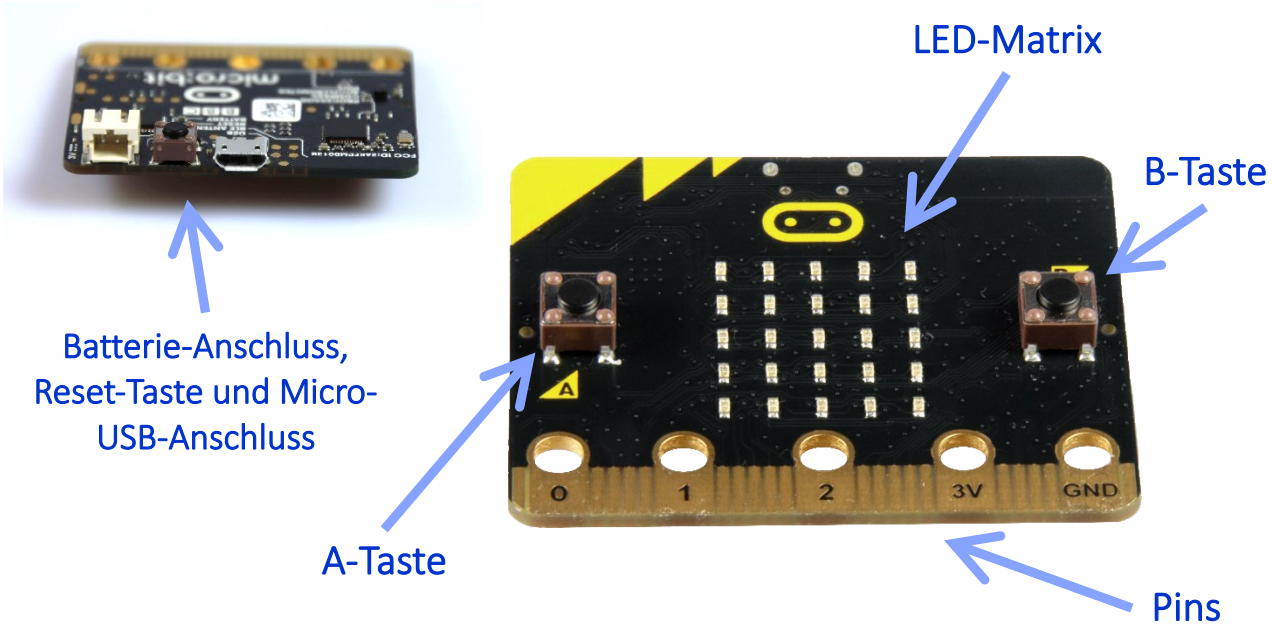
Drücke die Reset-Taste, um eine neue Verteilung der Punkte zu erzeugen. Wie oft musst du drücken, bis du ein Bingo erzielst? Wir haben es bisher nicht geschafft…

Meine Bestleistung

Folgende Dinge findest du sonst noch in deinem Paket



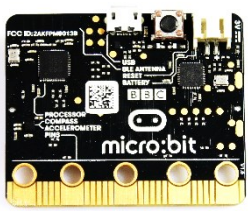
Wie funktioniert der micro:bit?



Der micro:bit ist ein sogenannter *Mikrocontroller*, ein einfacher kleiner Computer. Er kann *programmiert* werden, wie z.B. mit dem Spiel von gerade.

Wie jeder Computer arbeitet er nach dem **EVA-Prinzip:**

Sensoren



Mikrocontroller

Aktoren

Erinnerung:
Die Programmierumgebung MakeCode

Der micro:bit kann mit einer **blockbasierten Programmieroberfläche** wie beispielsweise *MakeCode* programmiert werden. Öffne dazu die folgende Internetseite mit einem Webbrowser:

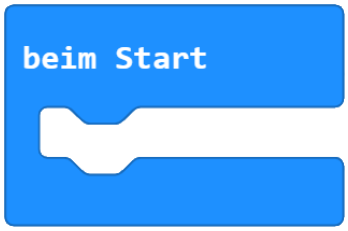
makecode.microbit.org

Dort kannst du ein neues Programm mit der Schaltfläche „Neues Projekt“ anlegen und ihm einen Namen geben.

Anschließend gelangst du zu der Programmierumgebung MakeCode:

micro:bit, lächle doch mal!

Jeder Block, den du zum Programmieren verwendest, muss sich in einem Startblock befinden.



Für den Moment genügt uns der Block „**beim Start**“. Blöcke hierin werden ausgeführt, sobald der Mikrocontroller startet, d.h. beim Anschließen des Stromes oder nach dem Drücken der Reset-Taste.

Den Block „**dauerhaft**“ brauchen wir erst später. Hierin werden die Blöcke immer wieder ausgeführt. Das heißt, wenn der Inhalt „abgearbeitet“ ist, wird damit wieder von vorne begonnen.

Um die LEDs auf der LED-Matrix zu steuern, gibt es verschiedene Blöcke. Einer ist zum Beispiel der folgende aus der Kategorie „Grundlagen“ (blau):

Bestimme durch Anklicken, welche LEDs der Matrix am micro:bit leuchten sollen oder nicht.

Experiment 0: Smiley

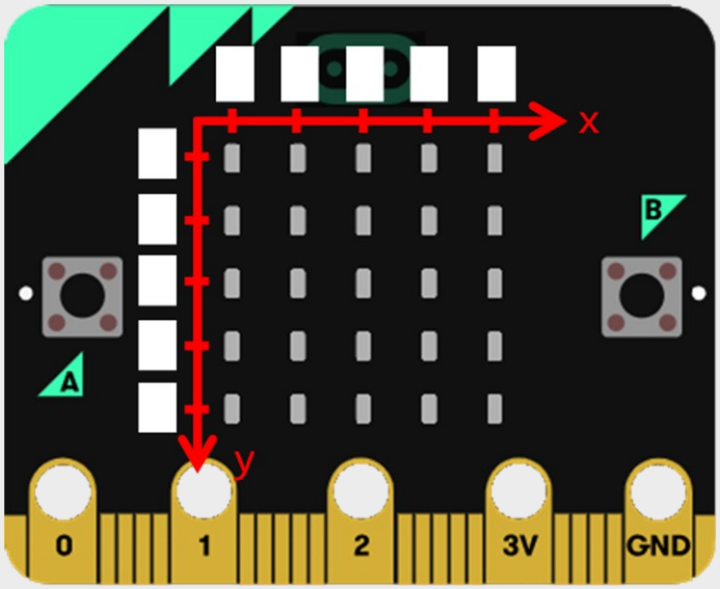
Lasse auf der Simulation in MakeCode einen Smiley mit dem obigen Block anzeigen.

Punkt, Punkt, Punkt

Wir wollen nun Stück für Stück das Spiel „micro:Bingo“ von Seite 8 nachprogrammieren. Dazu müssen wir erst wissen, wie man einzelne Punkte auf der LED-Matrix einzeichnet.

Information

Die LEDs auf dem micro:bit sind wie in einem *Koordinatensystem* angeordnet. Es sieht ein wenig anders aus, als du es aus dem Matheunterricht gewohnt bist. Es ist wie folgt aufgebaut:



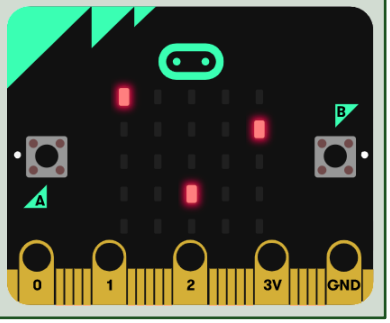
Die Einzelnen „Pixel“ der Matrix lassen sich mit dem folgenden Block der Kategorie „LED“ einschalten:

Experiment 1: Welcher Punkt bist du?

Versuche mit dem lila Block von oben das nebenstehende Bild auf deiner Simulation nachzubauen. Was stellst du hierbei fest?

Wie ist das Koordinatensystem aufgebaut?

Beschrifte seine Achsen in den weißen Kästen.



Alles nur Zufall?

Hinweis

Beim Zählen in der Informatik beginnt man fast immer bei der Zahl **0**!

Im nächsten Schritt soll der angezeigte Punkt an einer **zufälligen Position** sein. Hilfreich ist hier aus der Kategorie „Mathematik“:

Experiment 2a: Zufällige Punkte

Ändere dein Programm, sodass ab sofort die beiden Koordinaten nicht mehr fest, sondern **zufällig** sind.

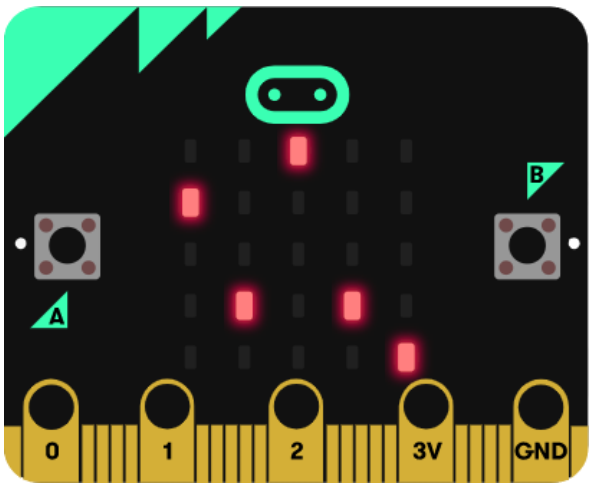
Achte dabei auf die richtigen Grenzen für die Zufallszahlen!

Wir kommen nun unserem micro:Bingo immer näher! Als nächstes müssen wir unser Programm so erweitern, dass **pro Spalte** ein zufälliger Pixel angezeigt wird.

Experiment 2b: Zufällige Punkte pro Spalte

Überlege dir, welche Koordinate „festgehalten“ werden muss und fülle deine LED-Matrix mit einer zufällig leuchtenden LED pro Spalte.

Achtung: Hier musst du manche Blöcke mehrmals verwenden!



Die Zählschleife

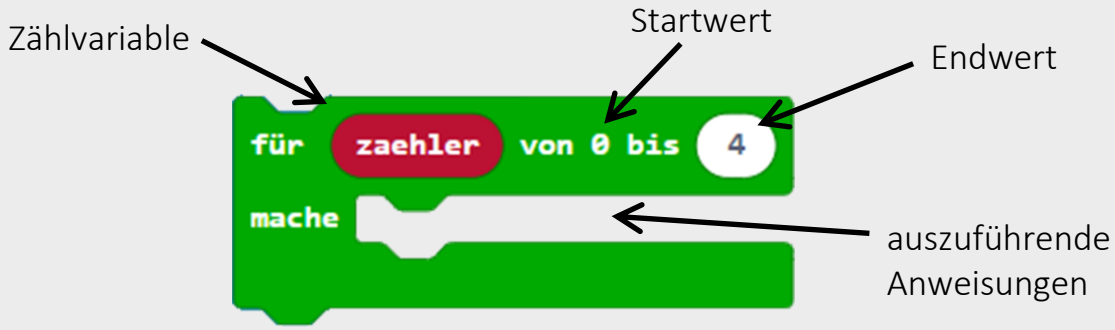
Du wirst feststellen, dass dein Programm einige Doppelungen enthält.

Lösung zu Experiment 2b

In der Regel versucht man so etwas beim Programmieren zu vermeiden, um das Programm so einfach wie möglich machen.

Information

Um Doppelungen wie oben zu vereinfachen, bietet es sich an, eine Anweisung mehrfach zu wiederholen. Hierzu verwendet man eine **Zählschleife** (oder auch **for-Schleife** genannt). Sie ist wie folgt aufgebaut:



Mit diesem Block wird „`zaehler`“ von **0** bis **4** jeweils um **1** hochgezählt. Pro Zählvorgang wird das Innere des Blockes einmal „abgearbeitet“.

Hier hat die Zählvariable „`zaehler`“ im ersten Durchlauf den Startwert **0**, im zweiten den Wert **1**, dann **2** … und im letzten den Endwert **4**.

Die Schleife wird also insgesamt 5-mal durchlaufen.

Vereinfache nun dein micro:Bingo!

Experiment 2c: Zufällige Punkte pro Spalte mit Zählschleife

Ändere dein Programm aus dem vorigen Experiment so ab, dass statt der Doppelungen eine Zählschleife mit der Zählvariable „`zaehler`“ verwendet wird.

Überlege dir:
Wie muss das Innere der Schleife aussehen?
Welche Anweisung möchtest du wie oft wiederholen?

Beachte

Bei Bedarf kann die Zählvariable umbenannt werden. Klicke dazu einfach mit der rechten Maustaste auf sie im for-Block. Vermeide ä ,ö, ü und ß im Namen.

Um im Inneren einer Zählschleife den aktuellen Wert der Zählvariable zu erhalten, kannst du den Block

aus der Kategorie „Variablen“ verwenden.

Herzlichen Glückwunsch! Dein micro:Bingo ist nun fertig.

Wenn du möchtest, kannst du es mit MakeCode auf deinen realen micro:bit übertragen. Eine Anleitung hierzu findest du im Anhang auf Seite 60.

Programmieren mit MicroPython

Sicher hast du schon einmal bemerkt, dass bei größeren Programmen die Programmierfläche in MakeCode sehr unübersichtlich werden kann. Deshalb programmieren Informatiker normalerweise nicht ***blockbasiert,*** sondern ***textbasiert***. Man spricht dann auch von einem **Programmcode**.

Ab sofort wollen auch wir textbasiert arbeiten. Wir verwenden dazu die Programmiersprache **Python** (bzw. etwas vereinfacht: **MicroPython**).

Das Programm zu micro:Bingo, das du gerade blockbasiert erstellt hast, sieht so aus:

Lösung zu Experiment 2c

Man kann es in MicroPython wie folgt „übersetzen“:

Programmcode: micro:Bingo [MicroPython]

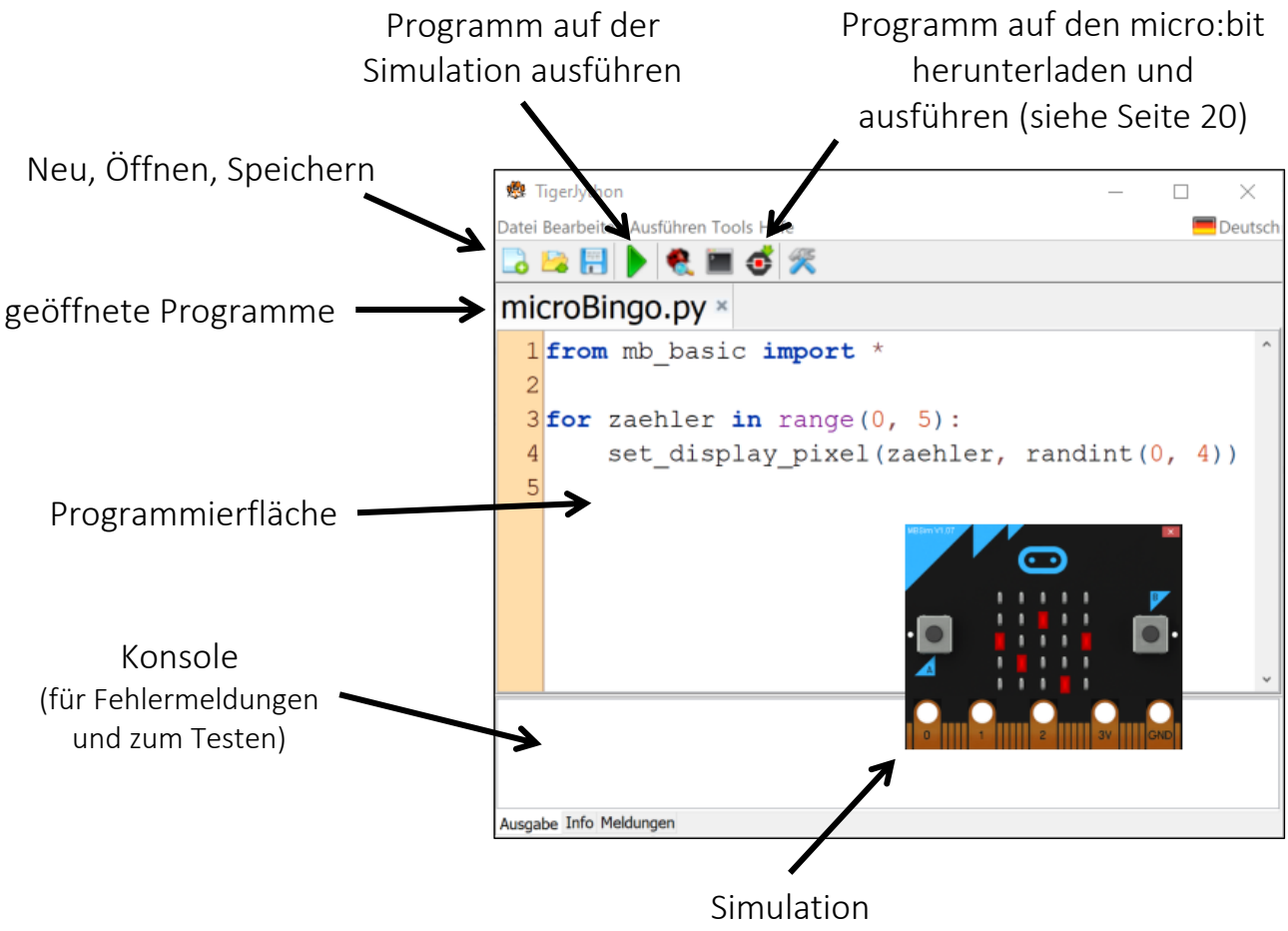
```
from mb_basic import *

for zaehler in range(0, 5):
    set_display_pixel(zaehler, randint(0, 4))
```

Um mit MicroPython zu programmieren, werden wir nun nicht mehr MakeCode verwenden, sondern die Programmierumgebung **TigerJython**.

[Bildquelle: www.tjgroup.ch]

Falls noch nicht geschehen, installiere dieses Programm bitte auf deinem Computer und starte es. Eine Anleitung dazu findest du auf Seite 59.

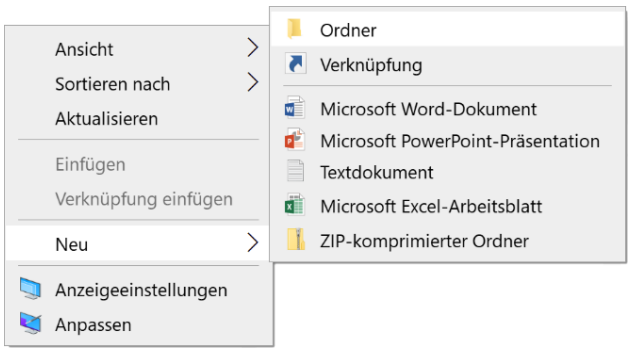
Die Entwicklungsumgebung TigerJython



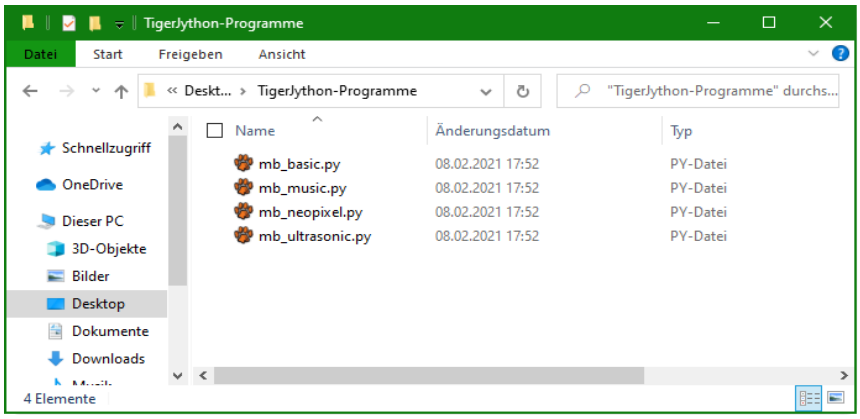
Vorbereitungen

Bevor du mit TigerJython programmieren kannst, befolge die folgenden Punkte:

1. **Lege dir einen neuen, leeren Ordner an**, am besten mit Rechtsklick auf dem Desktop oder unter „Dokumente“.
   Nenne den Ordner mit „*TigerJython-Programme*“.

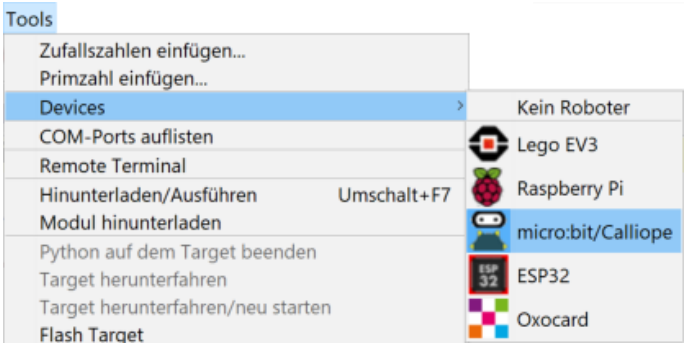


2. **Lade dir die sogenannten Bibliotheks-Dateien aus dem E-Learning (siehe Seite 5) herunter** und speichere sie in deinem Ordner als einzelne Dateien ab.



3. **Öffne TigerJython**.

4. Um TigerJython mitzuteilen, welches Gerät du programmieren möchtest, musst du einmal unter „Tools“ → „Devices“ **den Eintrag „micro:bit“ auswählen**.



5. Nun kannst du loslegen! Aber **Achtung**:

Hinweis

Speichere dein Programm immer zuerst in deinem Ordner „TigerJython-Programme“, bevor du das Programm ausführst. Andernfalls kann es zu Fehlern kommen.

Ein erstes Programm in der Simulation

Experiment 3a: micro:Bingo mit MicroPython

Starte TigerJython und übernimm den Programmcode für micro:Bingo (siehe Seite 17) in die Programmierfläche. Teste das Programm **in der Simulation**, indem du auf ▶ klickst.

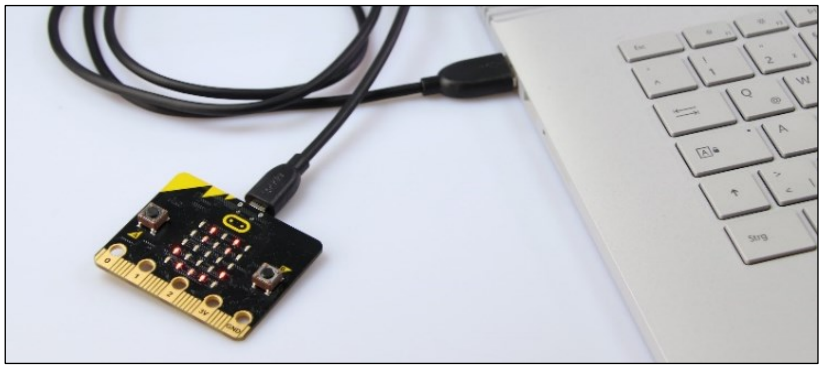
TigerJython-Programme an den (realen) micro:bit übertragen

Achtung!

Beachte auf Seite 4: „2. Stromversorgung“

Schritt 1

Schließe den micro:bit mit dem USB-Kabel an deinen PC an.

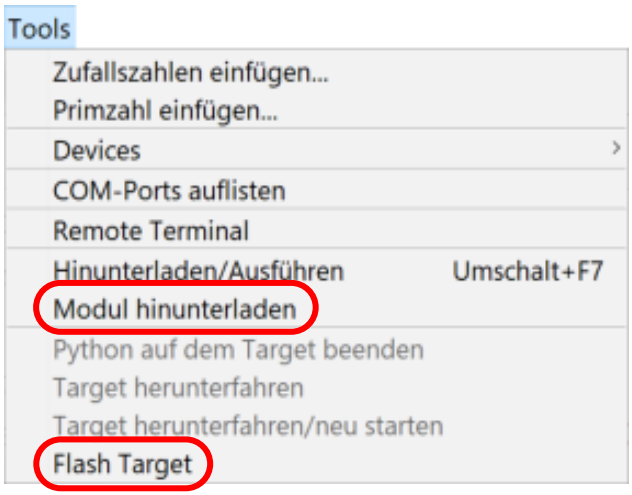


TigerJython erkennt automatisch das angeschlossene Gerät. **Achte darauf, dass das richtige Device (siehe Seite 19) ausgewählt ist.**

Schritt 2

(nur bei der ersten Verwendung mit TigerJython)

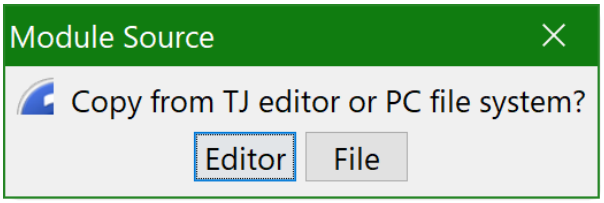
Falls du den micro:bit zum ersten Mal mit TigerJython verwendest, musst du unter „Tools“ auf „Flash Target“ klicken. Dien micro:bit wird nun vorbereitet. Du kannst den Fortschritt über die Konsole verfolgen.



Schritt 3

(nur bei der ersten Verwendung mit TigerJython)

Installiere anschließend die Bibliothek `mb_basic.py`. Öffne dazu die gleichnamige Datei aus deinem Ordner „TigerJython-Programme“, klicke auf „Tools“ → „Modul hinunterladen“ und wähle „Editor“ aus.

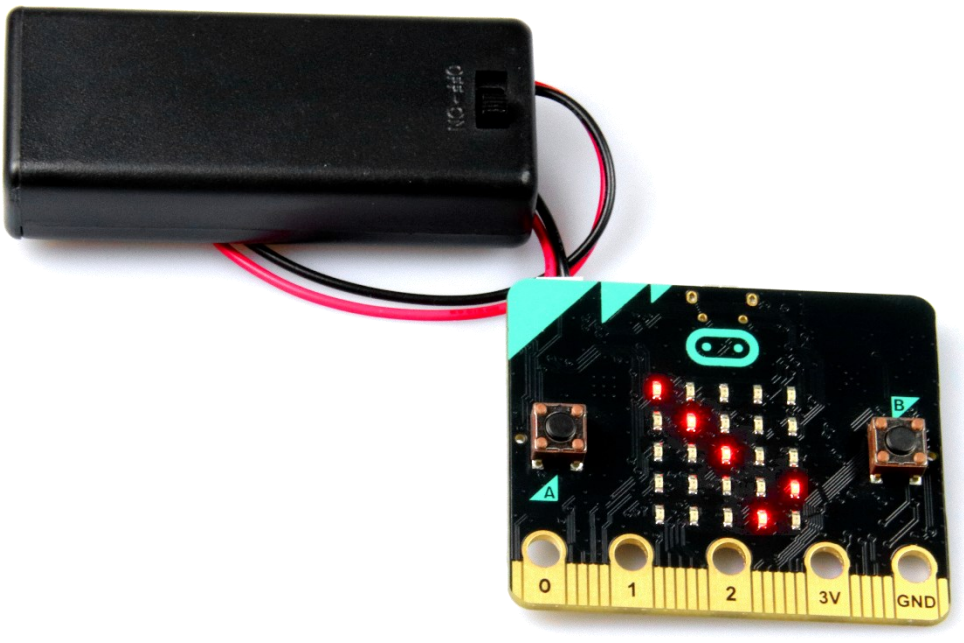


Schritt 4

Um dein Programm herunterzuladen, klicke auf den Button „Herunterladen und Ausführen“ in der Toolbar. Nach einem kurzen Moment (die LED auf der Rückseite des micro:bit binkt währenddessen) startet das Programm automatisch.

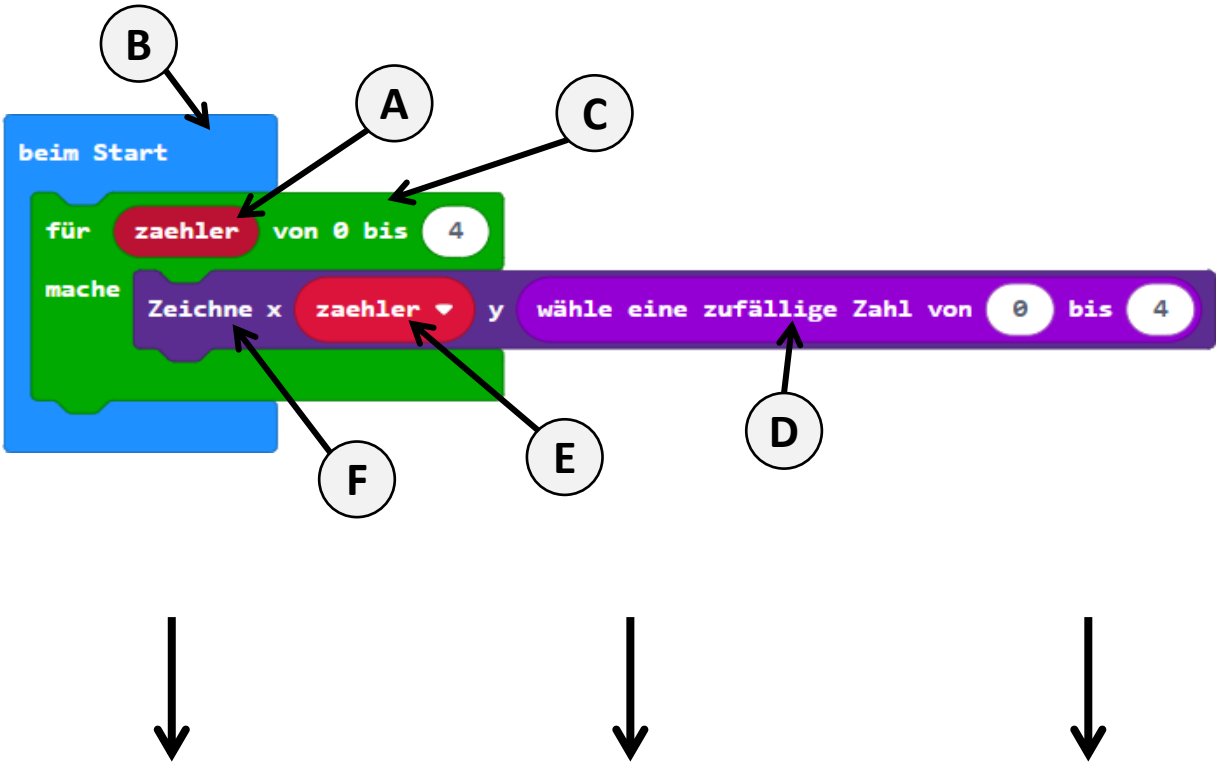
Experiment 3b: micro:Bingo goes real

Übertrage dein micro:Bingo auf deinen realen micro:bit und spiele es dort!



Vergleich: MakeCode-Blöcke vs. MicroPython

Vergleiche nun die beiden Programme (blockbasiert und textbasiert), indem du im textuellen Programmcode die passenden Stellen einkreist und mit den Buchstaben versiehst. Wo findet sich welche Struktur wieder? Als Beispiel haben wir A für dich markiert.



Programmcode: micro:Bingo [MicroPython]

```
from mb_basic import *

for zaehler in range(0, 5):

    set_display_pixel(zaehler, randint(0, 4))
```

Information

Zu Beginn eines jeden MicroPython-Programmes musst du „ankündigen“, welche Fähigkeiten dein Programm benutzen soll. Zum Beispiel muss TigerJython wissen, dass du im Programm den Befehl `set_display_pixel(...)` verwenden wirst.

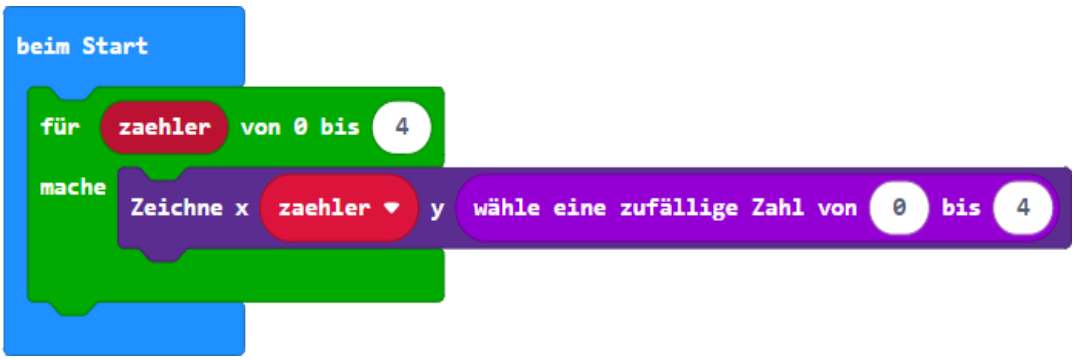
Wir haben dir alle nötigen Dinge in einer sogenannten **Bibliothek** vorbereitet. Diese musst du bei jedem Programm in der ersten Zeile wie folgt angeben:

```
from mb_basic import *
```

Achtung

Um die Bibliothek „`mb_basic`“ in TigerJython verwenden zu können, musst du sie vor der ersten Verwendung installieren. Befolge dazu die Anleitung auf Seite 21 (Schritt 3).

Zusammengefasst siehst du hier nochmal die beiden Programmvarianten, wobei wir MicroPython in den Farben der zugehörigen Blöcke markiert haben:



Programmcode: micro:Bingo [MicroPython]

```
from mb_basic import *

for zaehler in range(0, 5):
    set_display_pixel(zaehler, randint(0, 4))
```

Beachte beim Programmieren mit MicroPython

Schreibweise

Achte immer auf die korrekte Groß- und Kleinschreibung. MicroPython erkennt die Befehle sonst nicht.

Import

Vergiss nicht den Import zu Beginn deines Programmes:

```
from mb_basic import *
```

Später werden wir weitere Bibliotheken (`mb_neopixel`, `mb_music`,…) benötigen.

Hinweis

Zu viele Imports können den Speicher des micro:bit überlasten (`memory error`). Importiere deshalb immer nur die nötigsten Bibliotheken.

Code in, vor und nach der for-Schleife

Im MicroPython gibt es keine vorgefertigten Blöcke, die dir beim Programmieren helfen. Hier musst du selbst dafür sorgen, dass der Code an der richtigen Stelle steht! Probiere es selbst:

Experiment 4: Einrückungen

Ergänze dein micro:Bingo um den Befehl `print_slowly("fertig")`. Er gibt die Zeichenkette „`fertig`“ auf der Konsole aus. Was passiert, wenn du den Befehl

a) in der Zeile vor der for-Schleife platzierst?

b) in der Zeile nach „`set_display_pixel`“ platzierst mit Einrückung?

c) in der Zeile nach „`set_display_pixel`“ platzierst ohne Einrückung (gleich am Anfang der Zeile)?

Damit „`fertig`“ erst dann ausgegeben wird, wenn alle Pixel eingezeichnet sind, muss der Befehl wie in ___ beschrieben platziert werden.

Einrückungen

Achtung

Achte stehts auf die korrekte Einrückung deines Codes!

Hier findest du eine Übersicht für die Auswirkungen der Einrückung bei der for-Schleife:

Programmcode: Code vor der Schleife

[MicroPython]

```
from mb_basic import *

print_slowly("fertig")
for zaehler in range(0, 5):
    set_display_pixel(zaehler, randint(0, 4))
```

Hier wird „fertig“ angezeigt, **bevor** die LEDs aktiviert werden.

Nach jeder einzelnen LED wird **innerhalb der Schleife** einmal „fertig“ ausgegeben (also fünf Mal).

Programmcode: Code in der Schleife

[MicroPython]

```
from mb_basic import *

for zaehler in range(0, 5):
    set_display_pixel(zaehler, randint(0, 4))
    print_slowly("fertig")
```

Programmcode: Code nach der Schleife

[MicroPython]

```
from mb_basic import *

for zaehler in range(0, 5):
    set_display_pixel(zaehler, randint(0, 4))
print_slowly("fertig")
```

„fertig“ wird erst am Ende einmal **nach** der Abarbeitung der Schleife ausgegeben.

Zahlengrenzen

Experiment 5: Zahlengrenzen

Verändere die Zahlengrenzen in deinem Code (an den Stellen `range(0, 5)` und `randint(0, 4)`). Was stellst du fest? Wie ändert sich die Ausgabe auf der Simulation?

Du wirst merken: Anders als beim Block „**von 0 bis …**“ wird in MicroPython bei „`range(..., ...)`“ die Obergrenze nicht mitgezählt!

Information

Der Befehl `print_slowly(...)` wartet vor und nach der Ausgabe auf der Konsole einen kurzen Moment. Der Befehl ohne Warten lautet `print(...)`. Vielleicht hilft er dir in deinen folgenden Programmen bei der Fehlersuche.

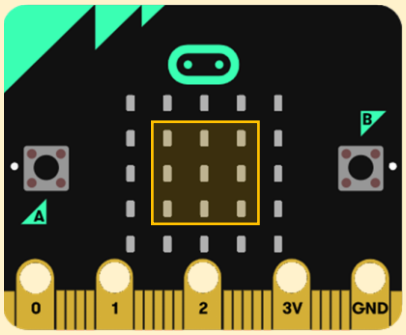
Mit deinem Wissen kannst du nun dein Programm weiter ausbauen:

Weiter gedacht…

Es ist sehr schwierig, fünf Punkte in einer Reihe zu erhalten. Überlege dir, wie du das Spiel leichter gestalten könntest!

Vielleicht müssen nicht alle Punkte zufällig festgelegt werden?

Eventuell gibt es auch eine kleinere Variante des Spiels, bei dem nur die inneren neun LEDs beteiligt sind…



Teile deine Programme mit den anderen Teilnehmern über das Forum auf dem E-Learning-Kurs! Siehe Seite 5: „Allgemeine Informationen“.

Externe Aktoren und Sensoren anschließen

Um externe Aktoren und Sensoren anzuschließen, können die Pins an der Unterseite des micro:bit verwendet werden.

Information

Strom fließt nur in einem geschlossenen Stromkreis.

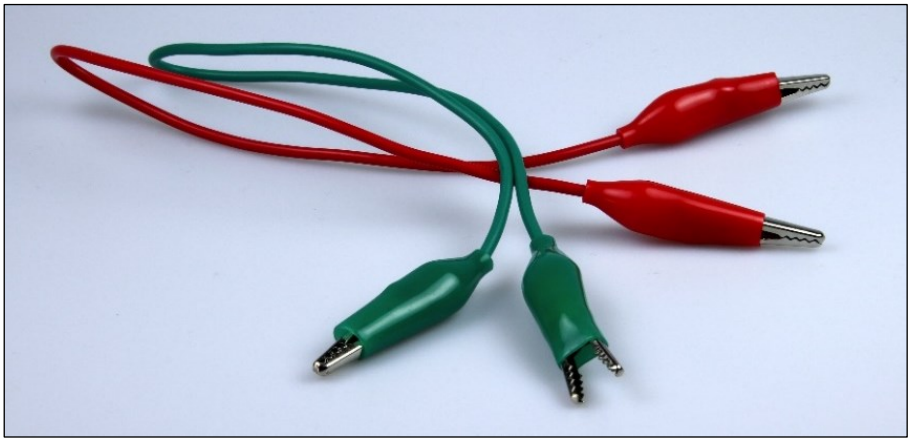
Die Pins 0, 1 und 2 sind programmierbar und geben Strom ab (Plus-Pol). Der Pin 3 V gibt dauerhaft Strom ab (Plus-Pol).

Der **Ground** (GND) ist der Minuspol.

Achtung!

Beachte auf Seite 4:
„9. Vermeide Kurzschlüsse“

Um zum Beispiel den Lautsprecher anzuschließen, können *Krokodil-Klemmen* verwendet werden:



Die Farbe der Krokodil-Klemmen spielt eigentlich keine Rolle. Es ist aber üblich, an den *Ground schwarze* und an den *Plus-Pol* (Pin 0, 1, 2 oder 3 V) *rote Kabel* anzuschließen.

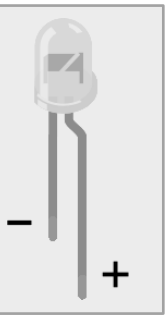
Eine externe LED programmieren

Mithilfe von Krokodil-Klemmen kann man zum Beispiel eine externe LED programmieren. Wir beginnen mit **einfarbigen** LEDs. Nimm hierfür eine LED **mit zwei Beinen** in der Farbe deiner Wahl aus deinem Experimentierset zur Hand.

Information

Eine LED hat immer ein kürzeres und ein längeres Beinchen. Das kürzere Beinchen wird immer näher zum Ground ausgerichtet.

Biege die beiden Beinchen bei der Verwendung mit Krokodil-Klemmen leicht auseinander, um keinen Kurzschluss zu verursachen! (siehe „9. Vermeide Kurzschlüsse“ auf Seite 4)



Eine LED kann über Pins gesteuert werden und benötigt ein *digitales* Signal (`0` oder `1`). Hierbei ist der folgende Befehl hilfreich:

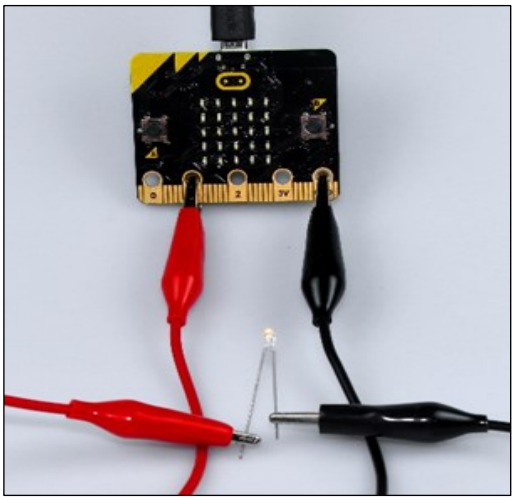
Programmcode: LED an- und ausschalten [MicroPython]

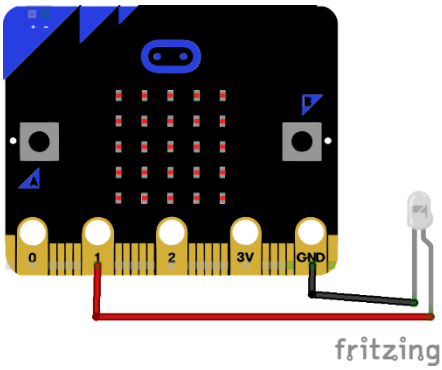
```
write_digital(pin_nr, value)
```

Verwende ihn, indem du `pin_nr` durch die Zahl des Pins und `value` durch `0` oder `1` ersetzt.

Experiment 6: Leuchtende LED

Schließe eine einfarbige LED mit Krokodil-Klemmen an Pin `1` an und programmiere sie so, dass sie beim Start des micro:bit eingeschalten wird.





Dauerhaft

Im nächsten Schritt soll die angeschlossene LED dauerhaft blinken.
Zur Erinnerung: In MakeCode konnte dies durch den Block „*dauerhaft*“ realisiert werden. Dies könnte zum Beispiel so aussehen:

Programmcode: dauerhaft [MicroPython]

Bei dem Block „dauerhaft“ handelt es sich um eine endlose Wiederholung. Um diese mit MicroPython umzusetzen, verwendest du das folgende Konstrukt:

```
while True:
    ...
```

Beachte, dass der Code, der sich innerhalb des „dauerhaft“-Blockes befand, wieder eingerückt sein muss.

Experiment 7a: Blinkende LED

Programmiere eine externe einfarbige LED, die dauerhaft immer wieder an- und ausgeschaltet wird.

Tipp: Ein Pausieren kann durch den Befehl `sleep(...)` realisiert werden. In den Klammern wir die Dauer der Pause in Millisekunden angegeben.

Weiter gedacht…

Warum ist es sinnvoll, den Befehl `sleep(...)` zweimal zu verwenden? Was passiert, wenn du ihn einmal weglässt? Probiere es auch und übertrage das Programm auch auf deinen realen micro:bit!

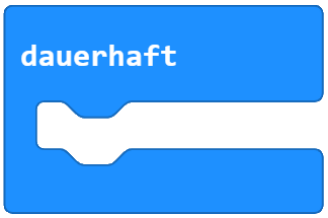
Das Konstrukt „dauerhaft“ ist sehr nützlich für Programme, die so lange laufen sollen, wie der micro:bit mit Strom versorgt wird. Es gibt jedoch eine Besonderheit!

Experiment 7b: Blinkende LED wird fertig

Benutze wieder den Befehl `print("fertig")`, um dein Programm aus dem letzten Experiment zu analysieren. Setze den Befehl an verschiedene Stellen und beobachte, was passiert.

Was stellst du fest, wenn der Befehl nach dem „dauerhaft“-Block (nicht eingerückt) steht?

`while True` erzeugt ein Konstrukt, das in der Informatik als **Endlosschleife** bezeichnet wird.



Kombiniere nun als nächstes die beiden Kontrollstrukturen, die du kennst: Endlosschleife und Zählschleife!

Experiment 8: Fading Light

Programmiere den folgenden Anwendungsfall:

Immer wieder soll die LED ganz schwach zu glimmen beginnen, dann immer heller werden, bis sie schließlich ihre maximale Helligkeit erreicht hat.

Anschließend soll sie wieder von vorne beginnen.

Um die Helligkeit einer LED zu steuern, muss ein *analoges* Signal verwendet werden. Benutze den folgenden Befehl:

Programmcode: LED Helligkeit steuern [MicroPython]

```
write_analog(pin_nr, value)
```

Hier musst du wieder `pin_nr` durch die Nummer des Pins ersetzen. `value` steuert die Helligkeit (also wie viel Strom durch die LED geschickt wird). Es kann Werte zwischen `0` und `1023` (inklusive) annehmen.

Lösung zu Experiment 7a und 7b

```
from mb_basic import *

while True:
    write_digital(1, 1)
    sleep(500)
    write_digital(1, 0)
    sleep(500)

# für Teil b:
# Die folgende Zeile wird wegen der
# Endlosschleife darüber nie erreicht:
print("fertig")
```

Information

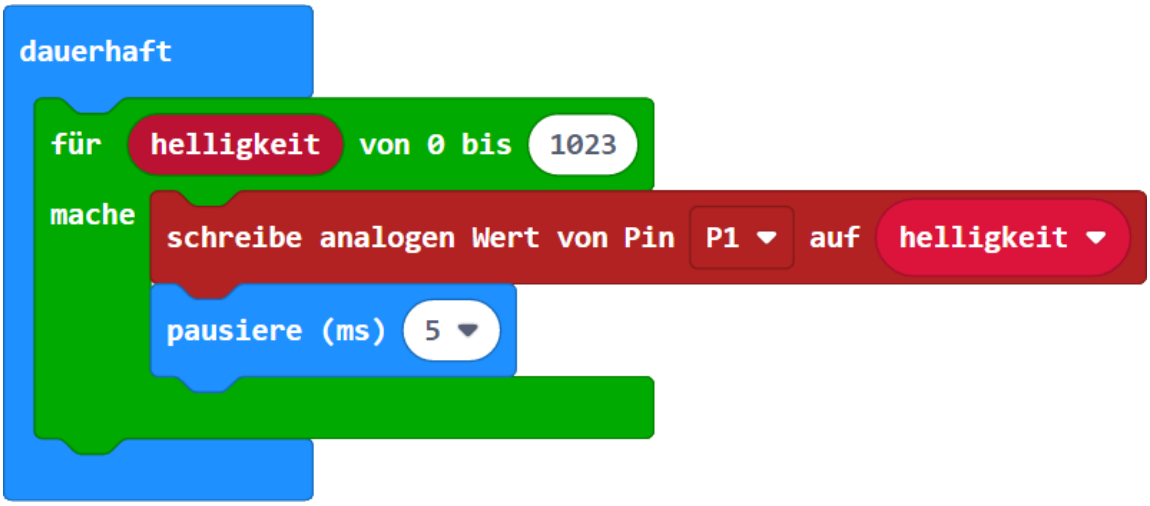
Zeilen im Programmcode, die das Zeichen # vorangestellt bekommen, werden als **Kommentare** bezeichnet.

Kommentare werden nicht von deinem micro:bit ausgeführt, sondern einfach ignoriert. Sie dienen dem Programmierer dazu, den Code zu beschreiben und erklären.

Lösung zu Experiment 8

```
from mb_basic import *

while True:
    for helligkeit in range(0, 1024):
        write_analog(1, helligkeit)
        sleep(5)
```

Dieser Code würde in einem blockbasierten Programm wie folgt aussehen:



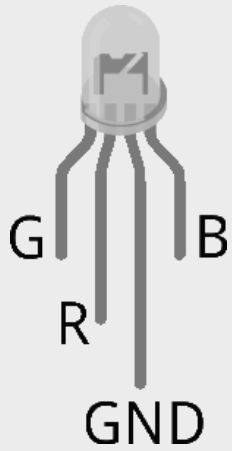
Weiter gedacht…

In der aktuellen Programmierung macht die Helligkeit einen Sprung von ganz hell auf ganz dunkel. Was müsstest du ergänzen, damit das Licht langsam wieder dunkler wird? Eine zweite for-Schleife kann dir dabei helfen.

RGB-LEDs

Neben den normalen einfarbigen LEDs gibt es auch welche, die mehrere Farben anzeigen können. Du findest solche LEDs ebenfalls in deinem Experimentierset. Sie haben *vier Beinchen*.

Information



Eine sogenannte **RGB-LED** kann drei verschiedene Grundfarben annehmen:

Rot,
Grün und
Blau.

Dementsprechend hat die LED ein Beinchen für jede Farbe und ein viertes für den Minuspol (Ground, **GND**). Dieses ist das längste und wird immer mit dem Ground verbunden.

Die Aufteilung der anderen Beinchen kannst du dem Bild entnehmen.

Achtung

Biege die Beinchen bei der Verwendung mit Krokodil-Klemmen immer weit genug auseinander. Sie oder die angeschlossenen Krokodilklemmen dürfen sich nicht berühren, um einen Kurzschluss zu vermeiden! Siehe „9. Vermeide Kurzschlüsse“ auf Seite 4.

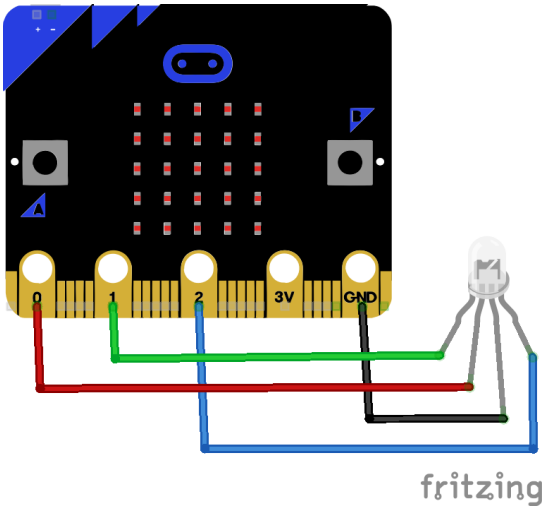
Ähnlich wie eine einfarbige LED können wir auch RGB-LEDs steuern. Je nach Helligkeit der einzelnen Grundfarben können beliebige Mischfarben mit dem Befehl `write_analog(pin_nr, value)` erzeugt werden.

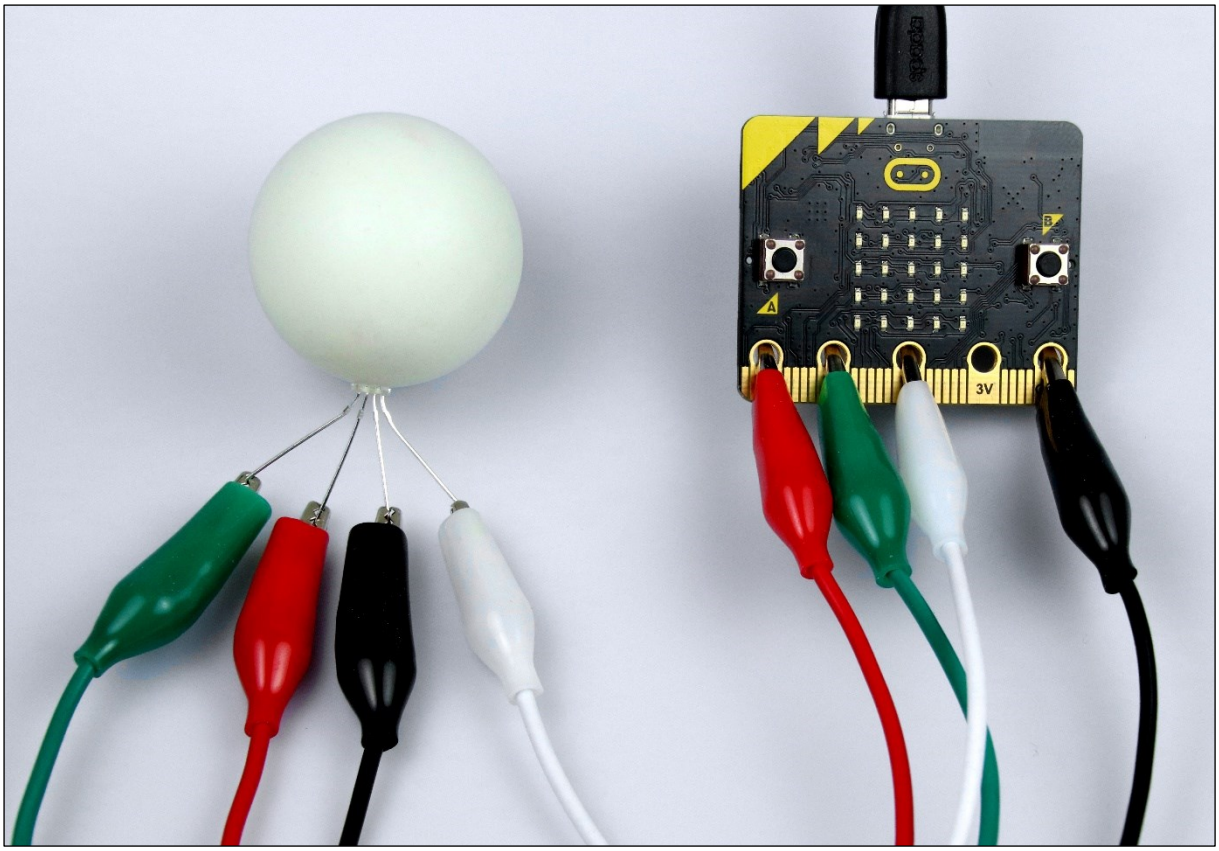
Experiment 9: Lieblingsfarbe

Schließe eine RGB-LED an deinen micro:bit mithilfe der Krokodilklemmen an. Verbinde Pin **0** mit dem Beinchen für Rot, Pin **1** mit Grün und Pin **2** mit Blau (siehe Aufbau unten).

Mische eine Farbe, bei der mindestens zwei der Grundfarben beteiligt sind.

Um die Mischfarben besser erkennen zu können, kannst du den Tischtennis-Ball in deinem Experimentierset über die LED stecken.





Wenn – Dann – Sonst

Lösung zu Experiment 9

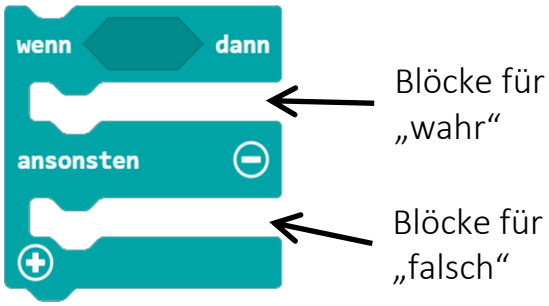
```
from mb_basic import *

# mithilfe der folgenden drei Aufrufe wird Türkis gemischt
write_analog(0, 100)
write_analog(1, 700)
write_analog(2, 300)
```

Von nun an soll eine bestimmte Farbe angezeigt werden, wenn eine Taste auf dem micro:bit gedrückt wurde. Zum Beispiel so:

Wenn die Taste A gedrückt ist,

dann wird die RGB-LED auf Grün geschaltet,

sonst auf Rot.



steht für eine *Bedingung*, die *wahr* (`True`) oder *falsch* (`False`) sein kann.

Programmcode: Bedingte Anweisung [MicroPython]

Falls Teile des Programmes von einer *Bedingung* abhängig sind, spricht man von einer *bedingten Anweisung*. Der Code hierfür sieht in MicroPython wie folgt aus:

```
if ...:
    # Code für den Fall „wahr“
else:
    # Code für den Fall „falsch“
```

Die Bedingung steht hier nach **`if`**. Je nachdem, ob sie wahr oder falsch ist, wird entweder der erste oder der zweite Teil ausgeführt und der Rest übersprungen.

Experiment 10a: Wunschlicht x2

Schreibe ein Programm, das immer dann die Farbe Grün anzeigt, wenn die Taste A gedrückt wird, sonst die Farbe Rot.

Tipp:
`is_button_a_pressed()` steht für die Aussage, ob die Taste A genau im Moment dieses Aufrufes gedrückt ist.

Programmcode: Verschachtelte bedingte Anweisungen

[MicroPython]

Du kannst auch mehrere bedingte Anweisungen hintereinanderschalten.

```
if Bedingung1:
    # Code für den Fall, dass Bedingung1 „wahr“
else:
    if Bedingung2:
        # Code für den Fall, dass Bedingung2 „wahr“
        # und Bedingung1 „falsch“
    else:
        # Code für den Fall, dass beide
        # Bedingungen „falsch“
```

Um weniger Einrückungen zu verwenden kann man auch folgendes schreiben:

```
if Bedingung1:
    # Code für den Fall, dass Bedingung1 „wahr“
elif Bedingung2:
    # Code für den Fall, dass Bedingung2 „wahr“
    # und Bedingung1 „falsch“
else:
    # Code für den Fall, dass beide
    # Bedingungen „falsch“
```

Experiment 10b: Wunschlicht x3

Erweitere dein Programm von oben dahingehend, dass eine dritte Farbe angezeigt wird, wenn die Taste B gedrückt wird.

Lösung zu Experiment 10b

```
from mb_basic import *

while True:
    if is_button_a_pressed():
        # Grün:
        write_analog(0, 0)
        write_analog(1, 1023)
        write_analog(2, 0)
    elif is_button_b_pressed():
        # Türkis:
        write_analog(0, 100)
        write_analog(1, 700)
        write_analog(2, 300)
    else:
        # Rot:
        write_analog(0, 1023)
        write_analog(1, 0)
        write_analog(2, 0)
```

Mehrere Bedingungen können mit dem Schlüsselwort **and** verknüpft werden:

Programmcode: Und [MicroPython]

```
is_button_a_pressed() and is_button_b_pressed()
```

Diese neue kombinierte Bedingung liefert den Wert `True`, wenn beide Bedingungen wahr sind, sonst `False`.

Weiter gedacht…

In welcher Reihenfolge müssen die Bedingungen überprüft werden, um unterschiedliche Farben für „Taste A gedrückt“, „Taste B gedrückt“, „Tasten A und B gedrückt“ und „nichts gedrückt“ anzuzeigen?

und, oder, nicht – ein Überblick

Erinnerst du dich an die Blöcke 

, 

und 

? Mit diesen kannst du Bedingungen verknüpfen.
Die zugehörigen Schlüsselwörter in MicroPython lauten **and**, **or** und **not**.

Experiment 11: und, oder, nicht

Erforsche die Auswirkungen von **and**, **or** und **not**.

Setze im folgenden Code-Schnipsel zuerst **and** und anschließend **or** an die Stelle des Platzhalters **X**. Trage in die Tabelle ein, ob die LED leuchtet.

```
if is_button_a_pressed() X is_button_b_pressed():
    # Lass die LED in einer Fabe leuchten.
else:
    # Schalte die LED aus.
```

Taste A gedrückt	Ja	Ja	Nein	Nein
Taste B gedrückt	Ja	Nein	Ja	Nein
and: LED leuchtet				
or: LED leuchtet				

Wie muss die Tabelle für folgenden Code-Schnipsel aussehen?

```
if not is_button_a_pressed():
    # Lass die LED in einer Fabe leuchten.
else:
    # Schalte die LED aus.
```

Taste A gedrückt	Ja	Nein
not: LED leuchtet		

Der Farbmixer

Bisher haben wir die Farbe der RGB-LED immer im Programm gemischt. Jetzt wollen wir die Farbe mithilfe der Taster auf dem micro:bit einstellen. Wir beginnen mit dem Rot-Anteil. Betrachte dafür das folgende Programm:

Programmcode: Rot selber einstellen [MicroPython]

```
from mb_basic import *

write_analog(0, 0) # schalte zu Beginn Rot (Pin 0) aus
write_analog(1, 0) # schalte zu Beginn Grün (Pin 1) aus
write_analog(2, 0) # schalte zu Beginn Blau (Pin 2) aus

while True:
    if was_button_a_pressed():
        # erhöhe die Helligkeit von Rot um 64
```

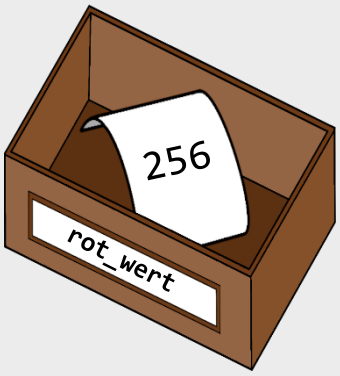
Tipp:
`was_button_a_pressed()` steht für die Aussage, ob die Taste A gedrückt und wieder losgelassen wurde.

Um den Helligkeitswert erhöhen zu können, muss das Programm wissen, welcher Wert aktuell verwendet wird. Dies kann man sich mit einer sogenannten **Variablen** merken.

Information

Eine **Variable** ist wie ein Behälter, in dem ein Wert liegen kann, z. B. wird in der Variable `rot_wert` die Zahl **256** gespeichert.

Bei der Verwendung der Variable wird der Wert aus dem Behälter abgerufen. **Wird der Variable ein neuer Wert zugewiesen, wird damit der alte überschrieben.**



Du kannst den Wert einer Variablen selbst festlegen: Man liest die folgende Code-Zeile als
„Die Variable `rot_wert` wird mit dem Wert `256` belegt“ oder
„`256` wird in `rot_wert` geschrieben“.

Programmcode: Variablenwert definieren [MicroPython]

```
rot_wert = 256
```

eindeutiger Bezeichner — Wertzuweisung — Wert

Experiment 12a: Rot erhöhen

Schreibe ein Programm, in dem man den Helligkeitswert von Rot über die Buttons selbst einstellen kann.

- Dafür brauchst du eine Variable `rot_wert`, in der gleich zu Beginn des Programmes 0 gespeichert wird.
- Im laufenden Betrieb soll immer wieder überprüft werden, ob die Taste A gedrückt wurde (`was_button_a_pressed()`).
- Ist dies der Fall, muss die Zahl aus der Variablen abgerufen, um `64` erhöht und wieder in `rot_wert` geschrieben werden:

  ```
  rot_wert = rot_wert + 64
  ```

  Danach muss Pin 0 aktualisiert werden.

Achtung: `write_analog(...)` kann auch mit Zahlen, die größer als `1023` sind, aufgerufen werden. Die Helligkeit beginnt dann wieder „von vorne“, d.h. `1024` erzeugt dieselbe Helligkeit wie der Wert `0`, `1025` wie `1` usw.

Mit diesem Programm lässt sich die Helligkeit von Rot einstellen. Wir haben aber noch andere Grundfarben:

Experiment 12b: Grün erhöhen

Ergänze dein Programm, sodass zusätzlich mit dem Button B der Helligkeitswert von Grün immer weiter um 64 erhöht werden kann.

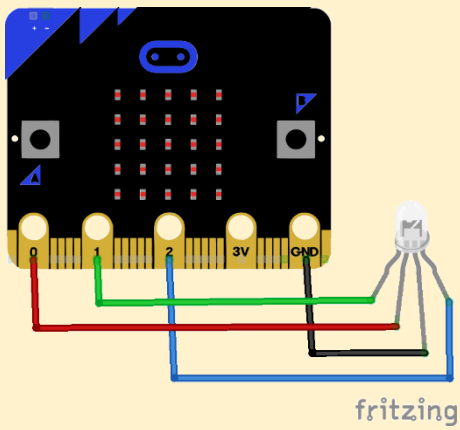
Lösung zu Experiment 12b

```
from mb_basic import *

rot_wert = 0
gruen_wert = 0

# schalte zu Beginn alle Farben aus:
write_analog(0, rot_wert)
write_analog(1, gruen_wert)
write_analog(2, 0)
# für Blau gibt es noch keine Variable,
# wir setzen die Helligkeit trotzdem "von Hand" auf 0.

while True:
    if was_button_a_pressed():
        rot_wert = rot_wert + 64  # erhöhe rot_wert
        write_analog(0, rot_wert) # setze die Helligkeit

    if was_button_b_pressed():
        gruen_wert = gruen_wert + 64 # erhöhe gruen_wert
        write_analog(1, gruen_wert)  # setze Helligkeit
```

Weiter gedacht…

Leider hat der micro:bit keine dritte Taste für Blau. Eventuell kannst du aber trotzdem ein Programm formulieren, das alle drei Farben ändern kann. Hier ein paar Ideen:

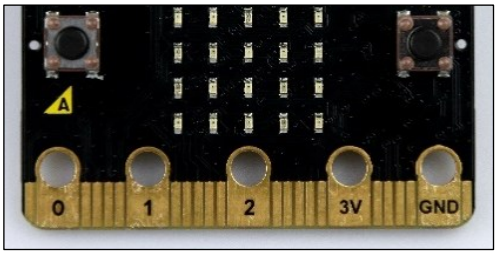
- Mit A wird die „aktuell bearbeitete“ Farbe ausgewählt, mit B wird der Wert erhöht.
- Die aktuell bearbeitete Farbe wird in einer extra Variable gemerkt. Sie sollte eventuell auch dem Nutzer auf der LED-Matrix angezeigt werden (`scroll_display_text(...)`).



- Je nach ausgewählter Farbe wird der entsprechende Helligkeitswert erhöht und am passenden Pin gesetzt.

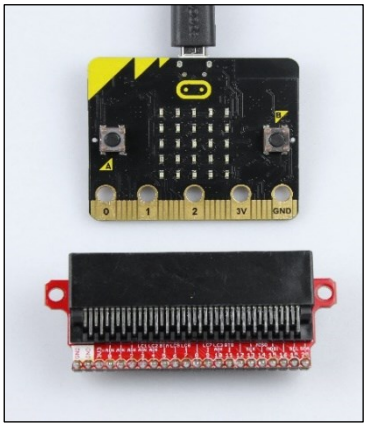
Ausbruch aus dem micro:bit

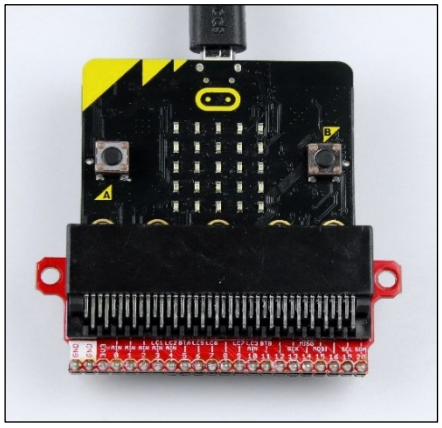
Der Breakout-Stecker



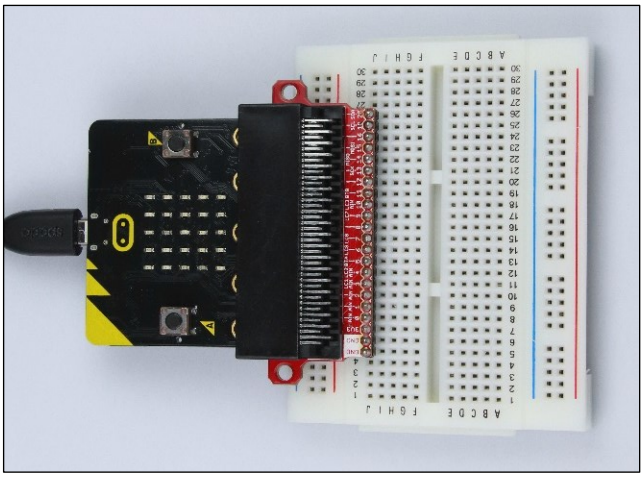
Wir wollen nun weitere RGB-LEDs einbauen. Hierfür reichen die drei programmierbaren Pins **0**, **1** und **2** offensichtlich nicht aus.

Der micro:bit besitzt noch weitere schmale Pins, die mithilfe des „Breakout“-Steckers benutzt werden können:





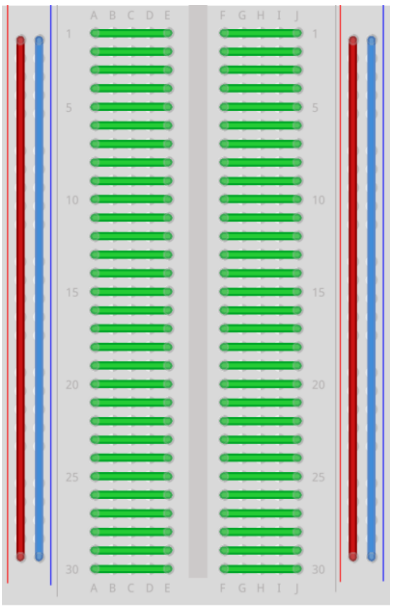
Mithilfe dieses Erweiterung-Steckers kann der Microbit auf ein Steckbrett, ein sogenanntes *Breadboard*, gesteckt werden:



Achtung

Beachte den Hinweis „Schaltungen stromfrei aufbauen“ auf Seite 4.

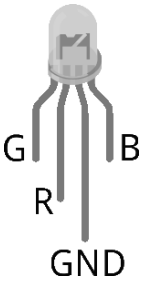
Schaltungen auf dem Breadboard aufbauen



In das Breadboard können elektronische Bauteile und Kabel einfach eingesteckt werden. Die Löcher auf dem Steckbrett sind untereinander so verbunden:

- Links und rechts außen sind die „Spalten“ durchverbunden (im Bild rechts rot und blau).
- Im Inneren Bereich sind jeweils die „Zeilen“ links bzw. rechts untereinander verbunden (im Bild grün).

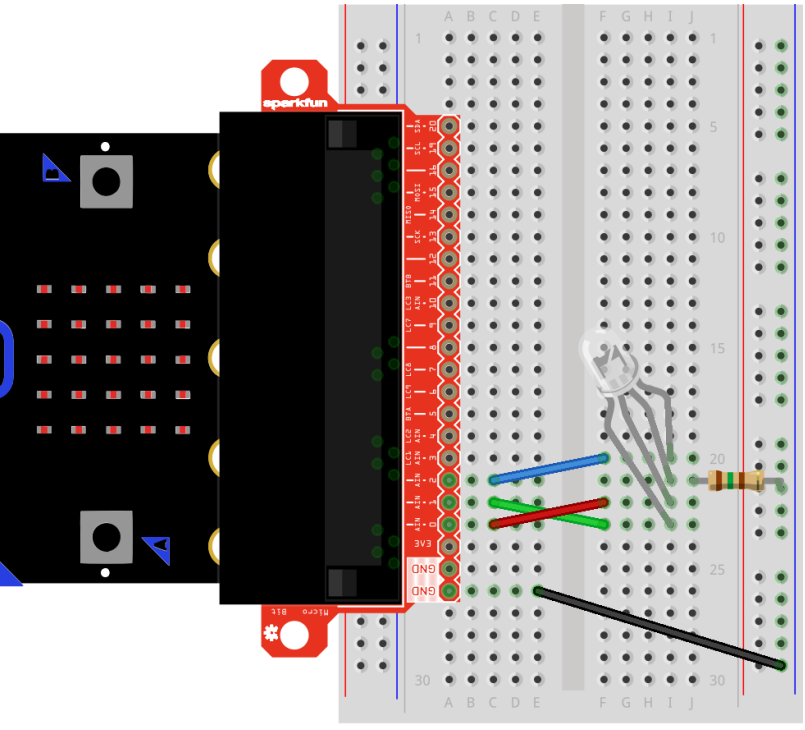
RGB-LED auf dem Breadboard

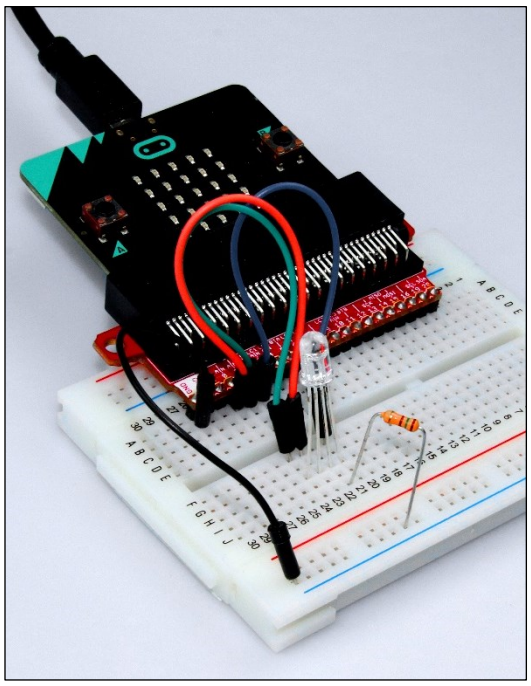


Stecke nun deinen micro:bit mithilfe des Breakout-Steckers auf dein Breadboard. Positioniere außerdem die RGB-LED auf dem Breadboard und verbinde sie wie abgebildet. Achte auf die Länge der Beinchen!

Der Widerstand (150 Ohm) ist notwendig, damit die LEDs mit der korrekten Spannung versorgt werden. Baue ihn entsprechend in deinen Aufbau ein.

Um Verbindungen auf dem Steckbrett herzustellen, verwende die dünnen sogenannten *Jumper-Kabel*.





Die Pins sind auf dem Breakout-Stecker durchnummeriert und lassen sich mit den bekannten Funktionen `write_digital(...)` und `write_analog(...)` steuern.

Experiment 13: Breakout mit einer RGB-LED

Probiere dein Programm aus dem vorherigen Experiment mit der neuen Schaltung aus. Am Programm selbst solltest du nichts ändern müssen, beachte aber den folgenden Hinweis.

Achtung

Die folgenden Pins hängen mit der LED-Matrix zusammen:

3 und 4, 6 und 7, 9 und 10

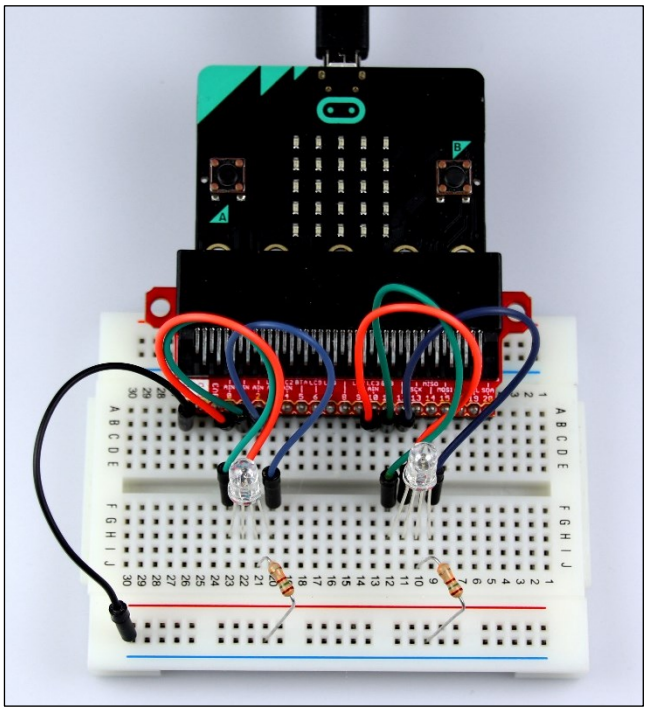
Du solltest deshalb, wenn du an einen dieser Pins etwas anschließt, immer zu Beginn deines Programmes den Aufruf `use_pins_with_breadboard()` stehen haben.

Nach diesem Aufruf ist die LED-Matrix des micro:bit für das komplette restliche Programm ausgeschalten.

Mehrere RGB-LEDs

Weiter gedacht…

Stecke eine zweite RGB-LED auf das Breadboard und verbinde es genauso wie die erste LED mit deinem micro:bit. Überlege dir weitere Programme, die du damit umsetzen kannst.

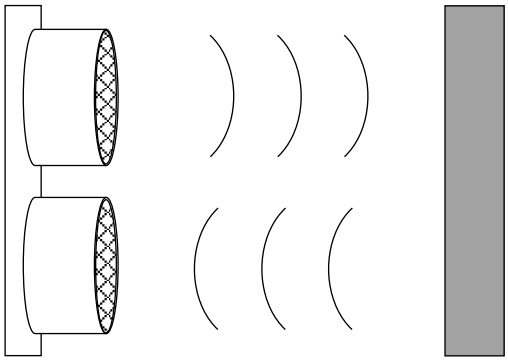


Ultraschall

In deinem Paket befindet sich auch ein Ultraschallsensor. Er sieht aus wie die Augen eines Roboters.

In Wirklichkeit sendet eines der „Augen“ Schallwellen aus, die für uns Menschen nicht hörbar sind. Das andere „Auge“ empfängt die Schallwellen, die von einem Objekt reflektiert werden.

Durch die Zeit, die dazwischen vergangen ist, kann berechnet werden, wie weit das Objekt entfernt ist.



Information

Der Ultraschallsensor hat vier Beinchen, die angeschlossen werden müssen.

- *GND* wird wieder mit dem Ground verbunden.
- *VCC* wird an den 3 V-Pin angeschlossen.
- Am *Trigger*-Pin wird ein digitales Signal angelegt, das den Sensor einen Ultraschallimpuls aussenden lässt.
- Über *Echo* kann die Entfernung zum Gegenstand empfangen werden.

Die Funktionen zur Bedienung des Ultraschallsensors erhältst du mit dem zusätzlichen Import:

```
from mb_ultrasonic import *
```

Programmcode: Verwendung des Ultraschall-Sensors

[MicroPython]

Zu Beginn des Programmes musst du dem micro:bit sagen, wo *Trigger* und *Echo* angeschlossen sind. Das passiert mit dem folgenden Befehl:

```
prepare_ultrasonic_sensor(trigger_pin_nr, echo_pin_nr)
```

Die (ungefähre) Entfernung in Zentimeter liefert dir die Funktion:

```
ultrasonic_distance_cm()
```

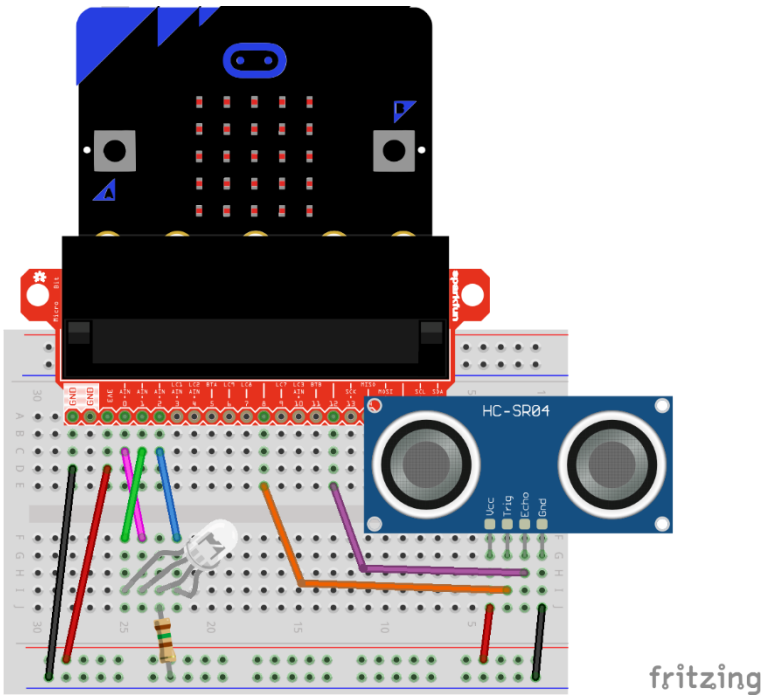
Experiment 14: Entfernungen messen

Programmiere den micro:bit so, dass die Entfernung zu Objekten auf der LED-Matrix angezeigt wird.

Wir wollen nun als nächstes den Rückfahrsensor eines Autos realisieren.

Experiment 15a: Rückfahrsensor mit Licht

Schreibe ein Programm, das die Farbe einer RGB-LED verändert, sobald der Abstand des Ultraschallsensors unter einen bestimmten Wert fällt. Vergiss nicht, das Licht wieder auszuschalten, wenn der Abstand groß genug ist.



Lösung zu Experiment 14

```
from mb_basic import *
from mb_ultrasonic import *

prepare_ultrasonic_sensor(8, 12)

while True:
    cm = ultrasonic_distance_cm()

    scroll_display_text(cm)
    print(cm) # zusätzliche Ausgabe auf der Konsole
    sleep(100)
```

Den Lautsprecher verwenden

Zusätzlich soll der Rückfahrsensor jetzt auch piepsen, je näher ein Objekt kommt. Hierfür brauchen wir den Lautsprecher aus dem Paket. Er kann über die beiden Kabel an das Breadboard angeschlossen werden. Verbinde den **grauen** Draht des Lautsprechers mit **GND**, den **roten** mit einem **beliebigen Pin** (`pin`).

Programmcode: Ton spielen [MicroPython]

```
play_tone(frequency, duration, pin)
```

In dieser Funktion kannst du angeben, welche Tonfrequenz (`frequency`) gespielt werden soll. Es sind Werte zwischen ca. `50` und `3900` möglich. Außerdem kannst du die Dauer des Tons über `duration` in Millisekunden angeben.

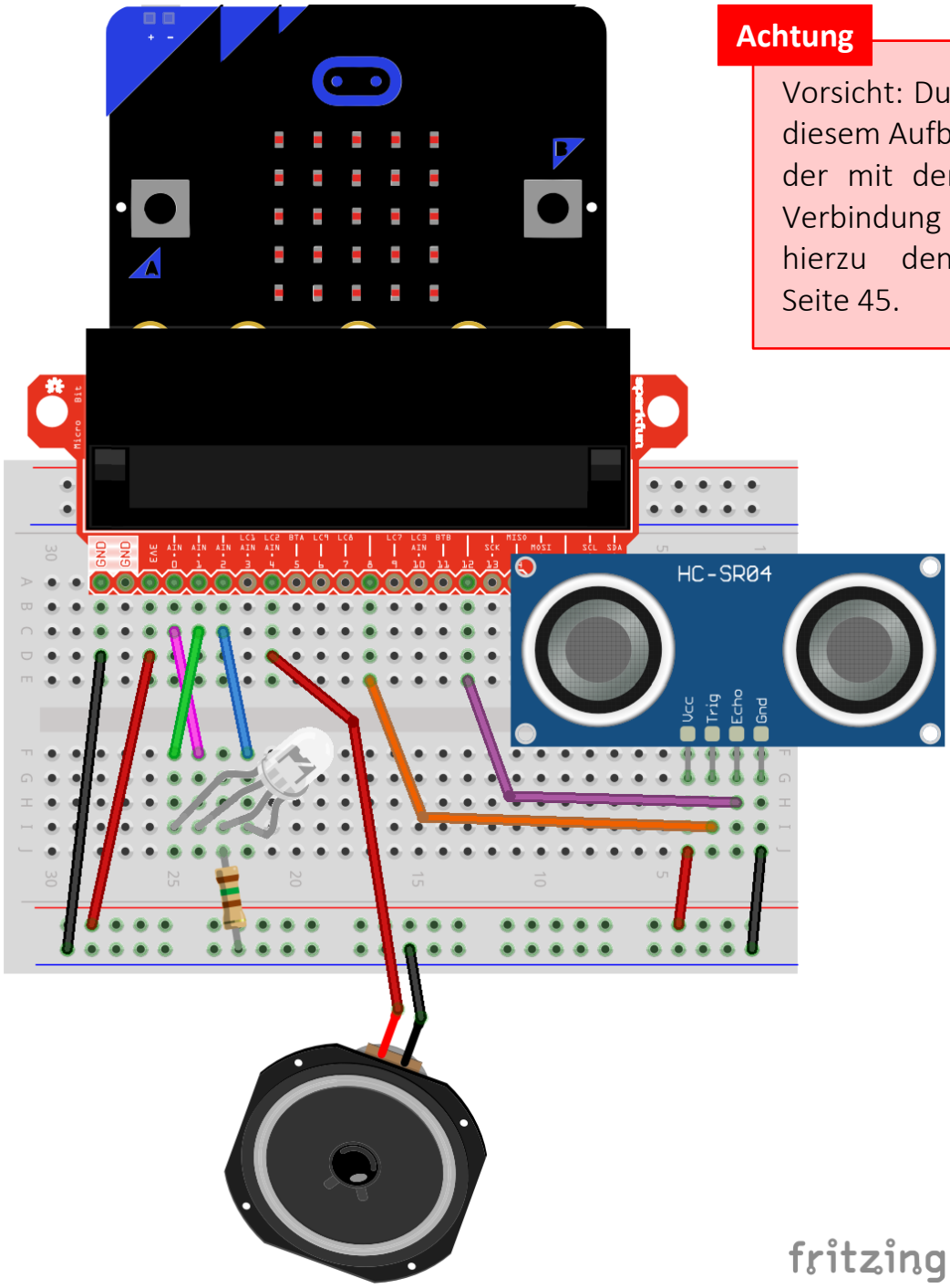
Information

Importiere die folgende Bibliothek, um Töne abzuspielen:

```
from mb_music import *
```

Experiment 15b: Rückfahrsensor mit Ton

Ergänze dein Programm aus dem vorigen Experiment um einen Piepston, der immer schneller wird, je näher ein Objekt kommt.



Achtung

Vorsicht: Du verwendest bei diesem Aufbau wieder Pin **4**, der mit der LED-Matrix in Verbindung steht. Beachte hierzu den Hinweis auf Seite 45.

Lösung zu Experiment 15b (mehr als in der Aufgabenstellung gefordert)

```
from mb_basic import *
from mb_ultrasonic import *
from mb_music import *

use_pins_with_breadboard()

# speichere die Pins der Einfacheit halber in Variablen
led_pin_r = 0
led_pin_g = 1
led_pin_b = 2

speaker_pin = 4

prepare_ultrasonic_sensor(8, 12)

while True:
    cm = ultrasonic_distance_cm()

    if cm < 15:
        write_analog(led_pin_r, 1023)
        write_analog(led_pin_g, 0)
        write_analog(led_pin_b, 0)
        play_tone(750, 100, speaker_pin)
    elif cm < 30:
        write_analog(led_pin_r, 1023)
        write_analog(led_pin_g, 511)
        write_analog(led_pin_b, 0)
        play_tone(750, 500, speaker_pin)
    else:
        write_analog(led_pin_r, 0)
        write_analog(led_pin_g, 0)
        write_analog(led_pin_b, 0)

    sleep(100)
```

Immer wieder dasselbe?

Folgender Code ist eine weitere Lösung zu Experiment 15b. Vergleiche die beiden Programme. Was hat sich verändert? Warum funktioniert es immer noch? Welchen Vorteil hat der untenstehende Code?

Programmcode [MicroPython]

```
from mb_basic import *
from mb_ultrasonic import *
from mb_music import *

use_pins_with_breadboard()

# speichere die Pins der Einfacheit halber in Variablen
led_pin_r = 0
led_pin_g = 1
led_pin_b = 2

speaker_pin = 4

prepare_ultrasonic_sensor(8, 12)

def setze_farbe(r_wert, g_wert, b_wert):
    write_analog(led_pin_r, r_wert)
    write_analog(led_pin_g, g_wert)
    write_analog(led_pin_b, b_wert)


while True:
    cm = ultrasonic_distance_cm()

    if cm < 15:
        setze_farbe(1023, 0, 0)
        play_tone(750, 100, speaker_pin)
    elif cm < 30:
        setze_farbe(1023, 511, 0)
        play_tone(750, 500, speaker_pin)
    else:
        setze_farbe(0, 0, 0)

    sleep(100)
```

Man kann wiederkehrende Anweisungen in einer sogenannten **Funktion** bündeln: Eine Funktion kann mehrere Code-Bausteine enthalten und wird meistens am Anfang des Programmes (nach den Variablen) definiert.

Programmcode: Eine Funktion definieren [MicroPython]

Name

drei sog. Parameter

```
def setze_farbe(r_wert, g_wert, b_wert):
    write_analog(led_pin_r, r_wert)
    write_analog(led_pin_g, g_wert)
    write_analog(led_pin_b, b_wert)
```

Verwendung eines Parameters innerhalb der Funktion

Eine so definierte Funktion kann nun immer wieder verwendet werden. Dabei wird immer ein Wert für jeden Parameter übergeben:

Programmcode: Eine Funktion verwenden [MicroPython]

Statt

```
write_analog(led_pin_r, 1023)
write_analog(led_pin_g, 511)
write_analog(led_pin_b, 0)
```

verwenden wir nun

```
setze_farbe(1023, 511, 0)
```

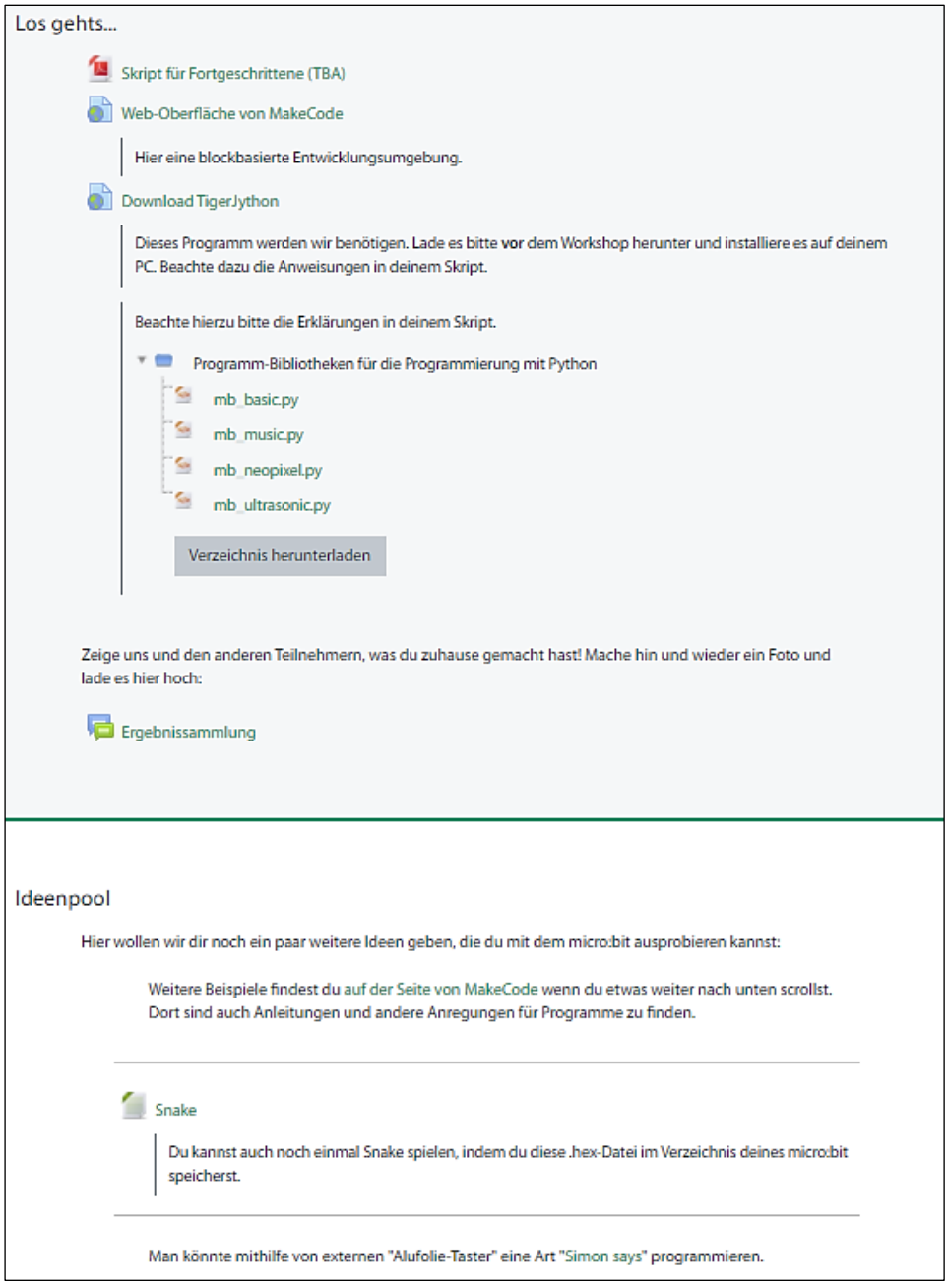
Weiter gedacht…

Findest du innerhalb eines deiner vorherigen Experimente wiederkehrende Code-Zeilen, die du in Funktionen bündeln kannst? Versuche deinen Code zu optimieren!

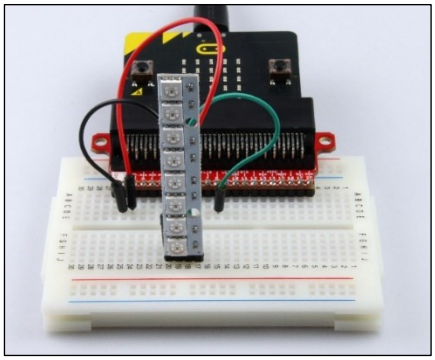
Herzlichen Glückwunsch!

Du bist nun in der Lage, einen Mikrocontroller mit MicroPython textbasiert zu programmieren! Jetzt bist du bereit, ein paar komplexere Beispiele umzusetzen. Auf den nächsten Seiten haben wir dir noch ein paar Infos zum RGB-Stick und RGB-Schlauch zusammengefasst.

Links und Anweisungen zu weiteren Ideen findest du auch im E-Learning-Kurs (siehe Seite 5). Schau einfach mal vorbei!



Anhang: RGB-LED-Stick und -Schlauch

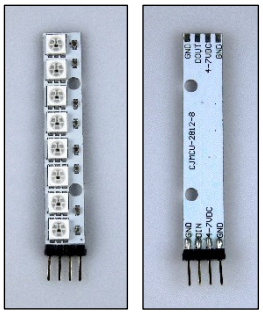


Betrachten wir einen weiteren externen Aktor: Einen RGB-LED-Stick.

Information

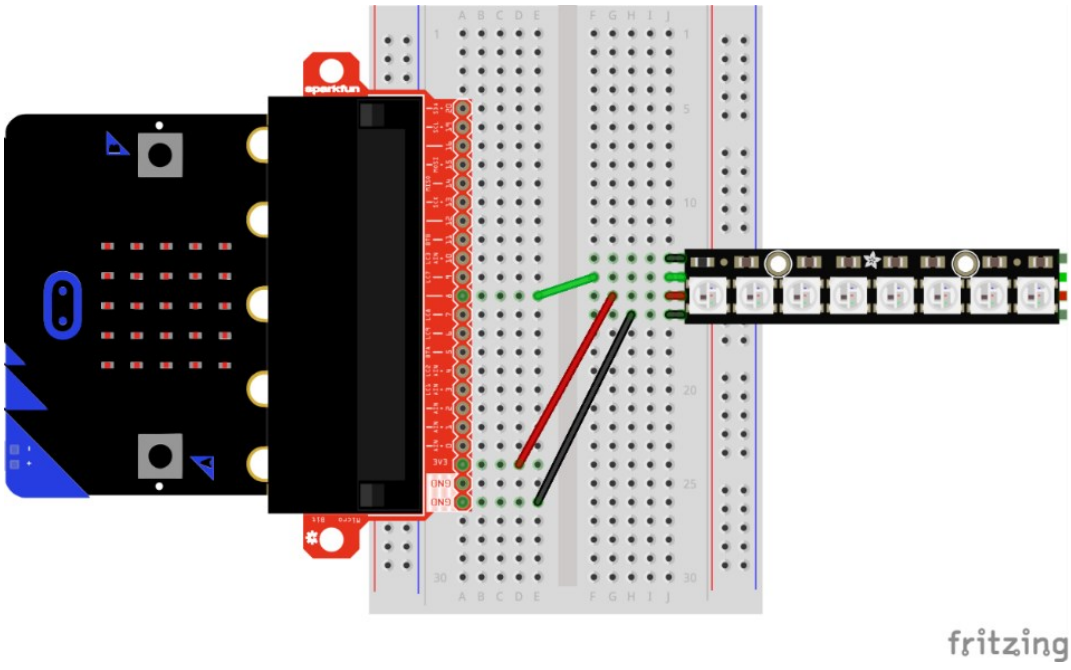
Bei einem RGB-Stick kann jede einzelne LED eine beliebige Farbe, gemischt aus Rot, Grün und Blau, annehmen.

Der LED-Stick hat vier Anschlussmöglichkeiten, die auf der Rückseite benannt sind: 2x **GND**, **VDC** und **DIN**.



Verbinde einen der beiden Grounds mit dem Ground (Minuspol) und VDC mit dem 3V3 (Pluspol). Bei DIN wird eine digitale Eingabe erwartet. Diesen Anschluss musst du mit dem Pin verbinden, der im Startblock benannt wurde (z. B. Pin 8).

Möglicher Schaltplan:



Die folgenden Funktionen finden sich in der Bibliothek `mb_neopixel`.

Um einen LED-Stick mit `n` LEDs zu programmieren, musst du die Verwendung am Anfang deines Programmes ankündigen:

Programmcode: LED-Stick vorbereiten [MicroPython]

```
prepare_neopixel(din_pin_nr, n)
```

Anschließend können die einzelnen Pixel mithilfe von RGB-Mischungen beliebig eingefärbt werden. Benutze dafür den folgenden Aufruf:

Programmcode: LED-Stick einfärben [MicroPython]

```
neopixel_set_pixel(nr, r_wert, g_wert, b_wert)
```

Erinnere dich bei `nr`, dass der Informatiker immer bei `0` zu Zählen beginnt. `r_wert`, `g_wert` und `b_wert` sind Zahlen zwischen `0` und `255`.

Um die Änderungen auf dem Stick sichtbar zu machen, ist der Aufruf der folgenden Funktion erforderlich:

Programmcode: LED-Stick Änderungen anzeigen [MicroPython]

```
neopixel_show()
```

Experiment 16 (Zusatz): LED-Stick

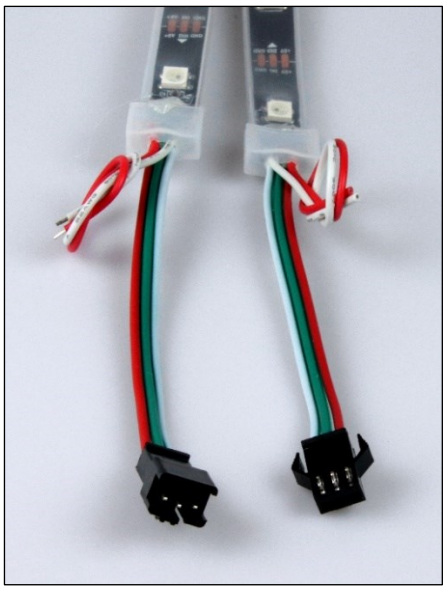
a) Färbe alle 8 LEDs in derselben Farbe.

b) Lass einen andersfarbigen Punkt von unten nach oben wandern.

c) Baue eine Ampel, die die obersten drei Pixel für rot, gelb und grün verwendet.
Tipp: Eine normale Ampel schaltet rot – rotgelb – grün – gelb – rot.

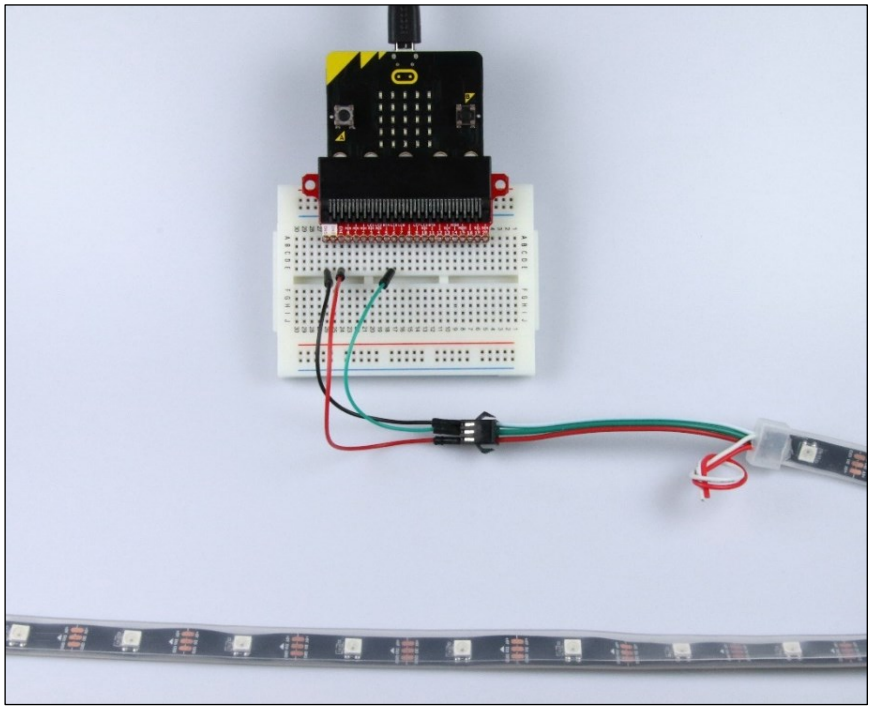
d) Überlege dir weitere Anwendungen für den LED-Stick und probiere sie aus.

LED-Schlauch



Genauso wie einen LED-Stick kann man einen LED-Schlauch mit 15 LEDs anschließen und programmieren. Er hat zwei verschiedene Anschlussseiten.

Benutze zum Anschließen Jumper-Kabel, die du in die rechte Anschlussseite im Bild oben steckst. Hier wird der Ground ausnahmsweise durch ein weißes Kabel dargestellt. Die beiden zusätzlichen Kabel werden nicht benötigt.



Experiment 17 (Zusatz): LED-Schlauch

Überlege dir Anwendungen für den LED-Schlauch und setze sie um.

Poste davon Fotos im E-Learning-Kurs (siehe Seite 5).

Anhang: Installation von TigerJython

Befolge diese Schritte, um das Programm TigerJython auf deinem PC zu installieren.

Schritt 1

Öffne folgende Internetseite mit einem Webbrowser:

www.tjgroup.ch/?site=download

Lade dort die Installationsdatei herunter, die zu deinem Betriebssystem passt.

Schritt 2

Führe die heruntergeladene Datei aus. Du findest sie in dem Ordner, den du beim Download angegeben hast oder unter „Downloads“. Die Software wird installiert.

Schritt 3 (nur für das Betriebssystem MacOS)

Solltest du das Betriebssystem MacOS benutzen, musst Du vor dem allerersten Start der Anwendung eine spezielle Datei ausführen, um deinen Rechner auf die Benutzung von TigerJython vorzubereiten.

Navigiere zum Installationsverzeichnis von TigerJython. In den meisten Fällen sollte das der Ordner „Programme“ bzw. „Applications“ sein.

Dort findest du eine Datei mit dem Namen „unlock“. Klicke mit der rechten Maustaste auf diese Datei und wähle „Öffnen mit“ → „Terminal“.

Schritt 4

Starte das Programm TigerJython.

(Navigiere zum Installationsverzeichnis der Anwendung und öffne TigerJython mit einem Doppelklick auf die Datei „TigerJython.exe“ oder „TigerJython.app“.)

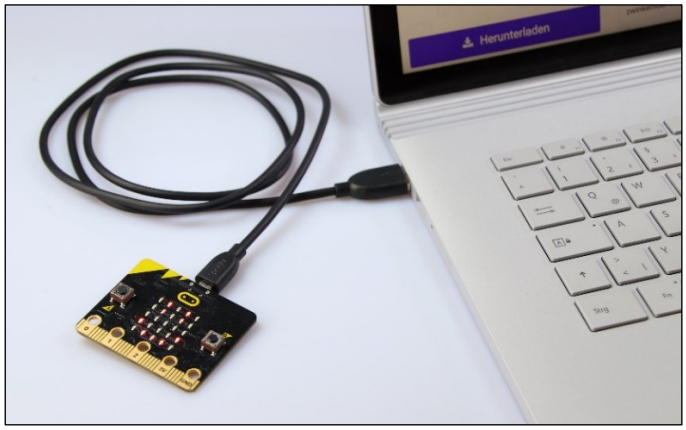
Anhang:
Programme von MakeCode an den micro:bit übertragen

Achtung!

Beachte auf Seite 4:
„2. Stromversorgung“

Schritt 1

Schließe den micro:bit mit dem USB-Kabel an deinen PC an.

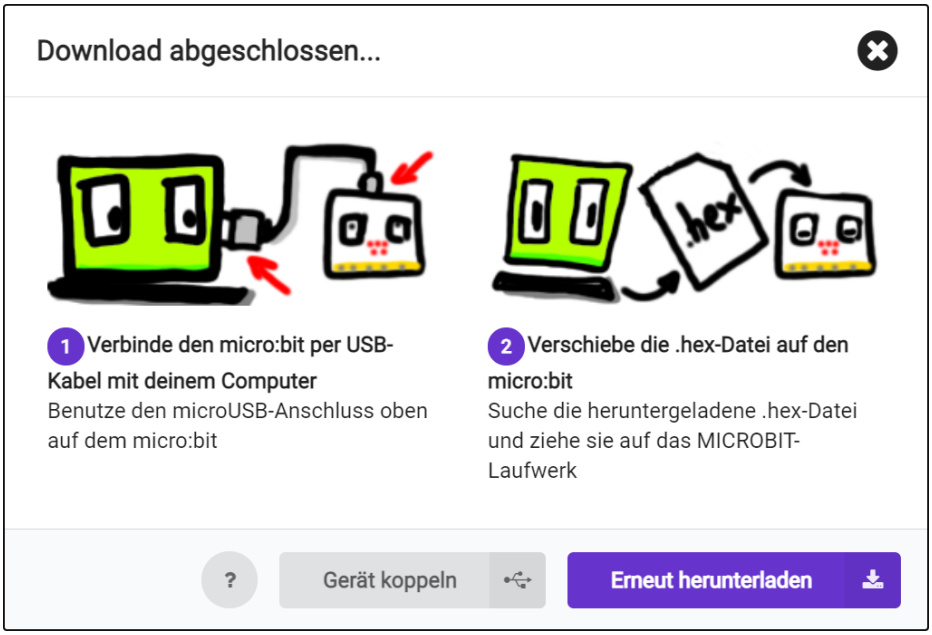


Schritt 2

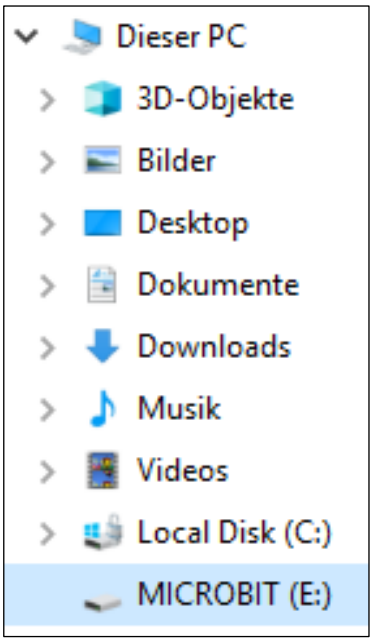
Klicke auf den Button 

.

Befolge die Anleitung zum Speichern der Programmdatei auf das micro:bit-Laufwerk:



Sollte sich ein weiteres Popup öffnen, wähle das Laufwerk des micro:bit aus und speichere die Datei dort.



Microsoft Windows

Mac OS

Hinweis

Manchmal wird die Datei auch direkt automatisch im „Download“-Ordner gespeichert. Verschiebe diese dann einfach auf das micro:bit-Laufwerk.

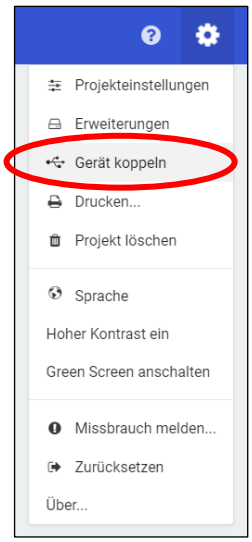
Falls du Google Chrome benutzt, kannst du das Gerät auch koppeln. Beachte hier die Anleitung auf Seite 62.

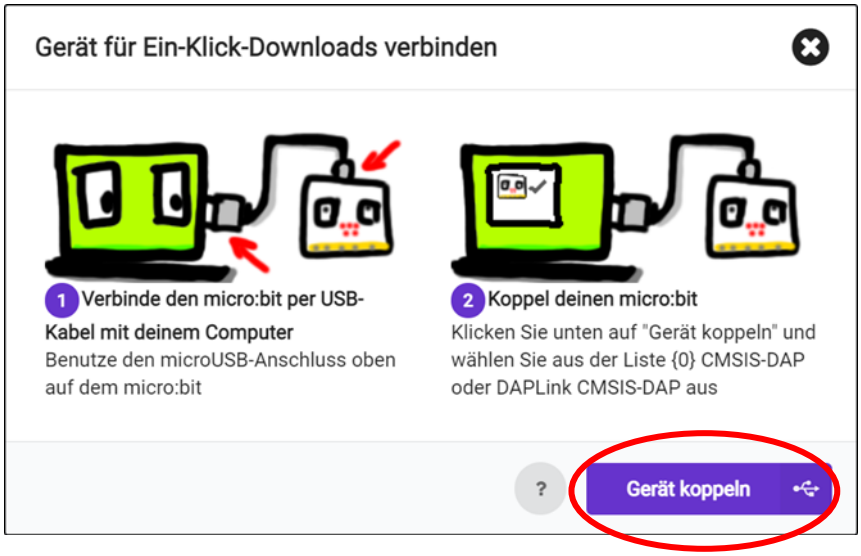
Während des Upload-Vorganges blinkt die gelbe LED auf der Rückseite des micro:bit. Der Upload-Vorgang sollte ein paar Sekunden dauern. Anschließend startet dein Programm automatisch.

Anhang: micro:bit mit Google Chrome koppeln

Schritt 1

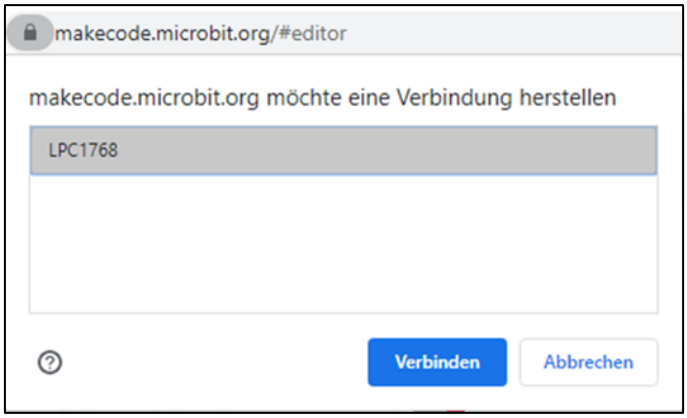
Wähle unter den **Einstellungen** die Option „**Gerät koppeln**“. Es öffnet sich ein Fenster. Klicke dort erneut auf „**Gerät koppeln**“:





Schritt 2

Wähle deinen micro:bit aus der Liste aus (es sollte nur einer verfügbar sein) und klicke auf „Verbinden“:



Dein micro:bit ist nun mit deinem Browser gekoppelt. Um ein Programm auf den Mikrocontroller zu übertragen, klicke auf Herunterladen. Das Programm wird dann nach kurzer Zeit starten.

Gefördert durch BayernMINT kompetent. vernetzt. erfolgreich

Dieses Skript entstand im Rahmen des
TAO-Schülerforschungszentrum Oberfranken
TechnologieAllianzOberfranken
tao-oberfranken.de/sfz

Digitales Lehren und Lernen &
Didaktik der Informatik
Universität Bayreuth
dlldi.uni-bayreuth.de