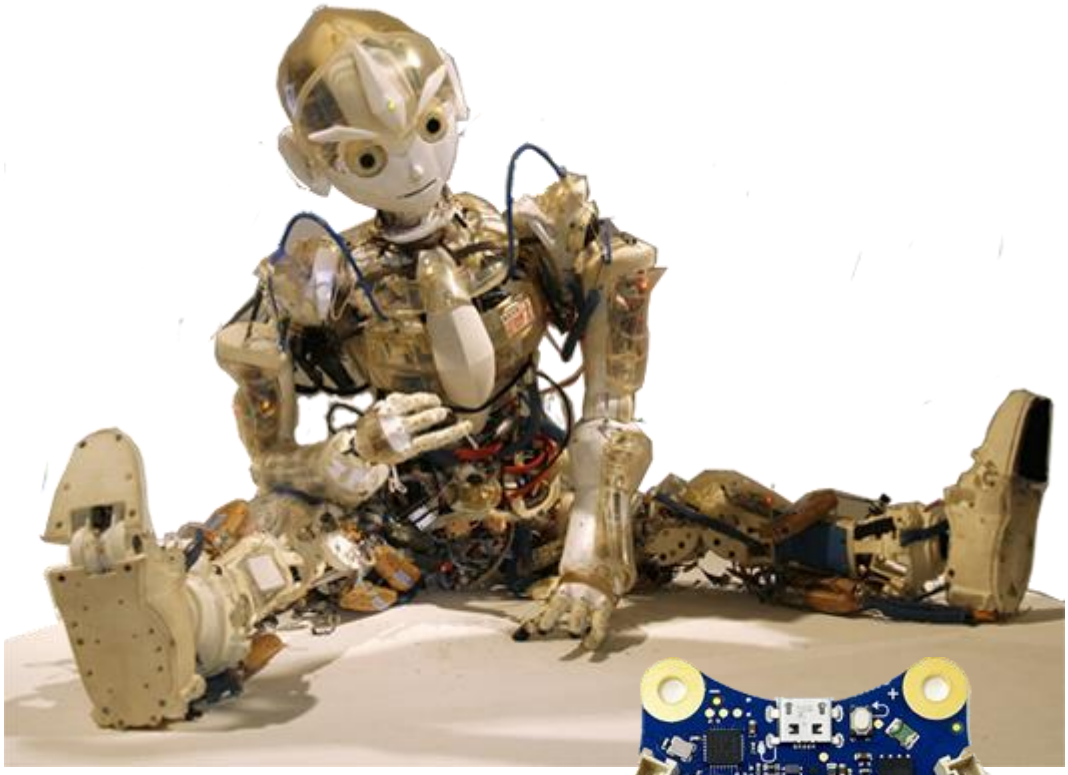
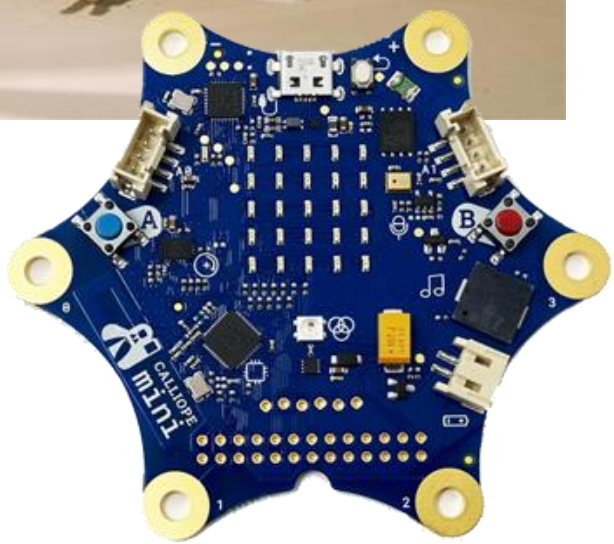
TigerJython4Kids

www.tigerjython4kids.ch

Teil 2: ROBOTIK





mit TigerJython und Calliope

www.tigerjython4kids.ch

INHALT

ROBOTIK MIT CALLIOPE

ANHANG:

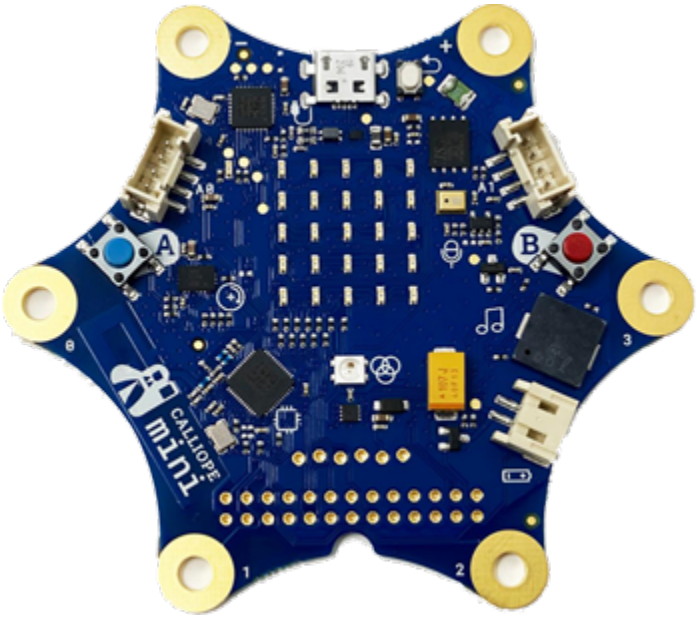
Version 1.0, März 2024

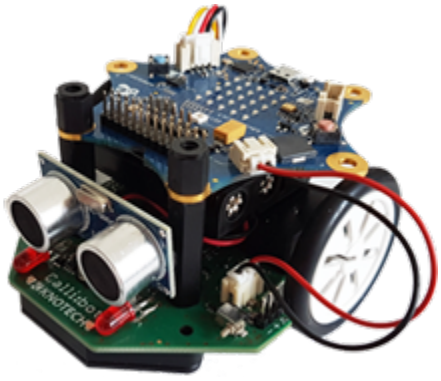
Kontakt: help@tigerjython.ch

CALLIOPE MINI

WAS IST EIN CALLIOPE ?

Der Calliope ist ein programmierbarer Computer im Taschenformat, der für Bildungszwecke entwickelt wurde. Er besteht aus einer ca. 7 x 7 cm grossen, sternförmigen Platine mit einem 32-Bit Mikrokontroller, Flash Speicher, Bluetooth, 25 roten Leds, mit welchen du einfachen Bilder und Nachrichten anzeigen kannst, einer Farb- LED, zwei Tasten (Buttons), Lautsprecher und Mikrofon, einem kombinierten Lage-, Kompass und Gyrosensor, sowie einer H-Brücke zum Anschluss von Motoren.





Weitere Sensoren. kannst du über gut zugängliche Pins und zwei I^2C-Schnittstellen anschliessen.

Zusammen mit dem kompakten und günstigen Calli:bot-Fahrwerk von Knotech, wird der Calliope zu einem fahrenden Roboter, mit dem man weitere wichtige Robotik-Konzepte aufzeigen kann. Musterbeispiele und Aufgaben für den fahrenden Roboter findest du im Lernprogramm Callibot.

Das neueste Callipe Modell **Calliope mini 3** verfügt zusätzlich über zwei weitere FarbLEDs und über eine USB3-Schnittstelle. Die Programme können daher schneller auf den Calliope heruntergeladen werden. Alle Programme in diesem Lehrgang können auch mit dem Calliope mini 3 ausgeführt werden.

Die Programme werden mit TigerJython auf einem Computer entwickelt und über ein USB-Kabel auf den Calliope hinunter geladen. Die Bedienung der Entwicklungsumgebug ist einfach. Eine kurze Anleitung dazu findest du unter dem Menüpunkt "Loslegen". Wir gehen davon aus, dass du aus dem Kapitel Turtlegrafik die grundlegenden Programmstrukturen in Python bereits kennst. Wenn dies nicht der Fall ist, empfehlen wir dir, zuerst das Kapitel "Python Crash Course" durchzuarbeiten.

SIMULATIONSMODUS

Falls du keinen Calliope hast oder wenn du dein Programm zuerst auf dem Computer testen möchtest, so kannst du den **Simulationsmodus** verwenden, bei dem dir allerdings nur ein Teil der Hardware-Optionen zur Verfügung steht.

1. LOSLEGEN

DU LERNST HIER...

wie du mit Python ein Programm entwickeln und auf dem Calliope ausführen kannst.

PROGRAMMENTWICKLUNG

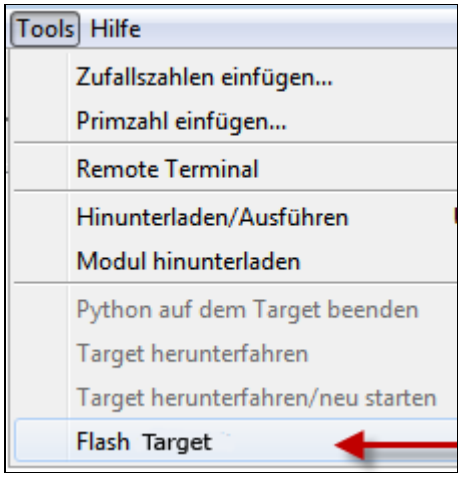
Ein Programm, das auf dem Calliope laufen soll, kannst du mit TigerJython auf irgendeinem Computer entwickeln. Nachdem du das Programm im Editor geschrieben hast, lädst du es über ein USB-Kabel auf den Calliope hinunter und es wird dort mit ***MicroPython*** (einer reduzierten Python-Version) ausgeführt. Dazu muss auf dem Calliope eine ***Firmware*** installiert sein.

USB-VERBINDUNG

Schliesse den Calliope über ein USB-Kabel am Computer an. Du siehst ein zusätzliches externes USB-Gerät (unter Windows mit einem Laufwerk-Buchstaben).

FIRMWARE INSTALLIEREN

Vor der ersten Verwendung musst du die Firmware auf den Calliope hinunterladen. Mit TigerJython ist dies sehr einfach: Wähle unter *Einstellungen/Bibliotheken* die Option *micro:bit/ Calliope*. Diese Einstellung bleibt gespeichert und ist auch für das Hinunterladen der Programme erforderlich. Wähle im Menü unter *Tools* die Option ***Flash Target*** .



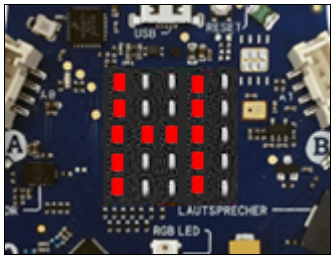
Warte bis im Ausgabefenster die Meldung: "Successfully transferred file to Calliope" erscheint.

MUSTERBEISPIEL

Zum Einstieg schreibst du nur einen kurzen Text auf dem Display aus. Tippe im Editor das unten stehende Programm ein oder klicke auf *In Zwischenablage kopieren* und füge es mit Ctrl+V im Editorfensterein.

```
from calliope_mini import *

display.scroll("HELLO PYTHON!")
```

► **In Zwischenablage kopieren**



Kontrolliere, ob der Calliope am Computer angeschlossen ist und klicke auf die Schaltfläche *Hinunterladen/Ausführen*.

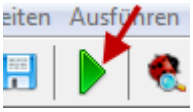
Neben dem TigerJython-Fenster erscheint ein zweites Terminal-Fenster. Hier werden Mitteilungen und Fehlermeldungen angezeigt.

Wenn HALLO PYTHON als Scrolltext angezeigt wird, funktioniert dein Calliope einwandfrei.

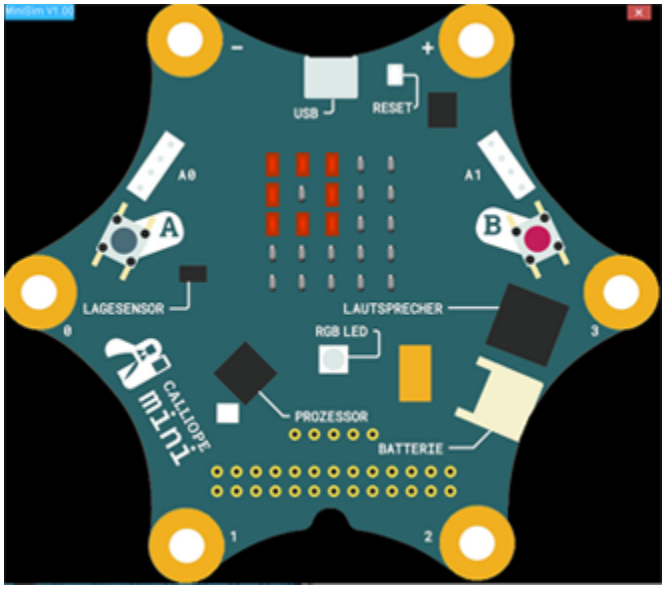
Der Programmcode ist leicht zu verstehen: In der ersten Programmzeile wird das Modul *calliope_mini* importiert und die Objekte und Funktionen aus dieser Bibliothek bereit gestellt, beispielsweise das Objekt *display*, mit dem du auf die LEDs zugreifst. Mit dem Befehl *display.scroll("HELLO PYTHON!")* kannst du kurze Mitteilungen als Lauftext anzeigen. Beachte, dass der Text in einfachen oder doppelten Anführungszeichnen stehen muss!

SIMULATIONSMODUS

Um ein Programm im Simulationsmodus auszuführen, klickst du den **grünen Startbutton**.



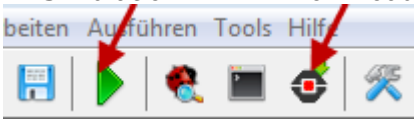
Im Simulationsmodus werden LEDs, Buttons, Pins und alle Befehle von *cpglow* unterstützt. Es erscheint ein Fenster, das du mit gedrückter Maustaste verschieben kannst. Beim Start des nächsten Programms wird das vorhergehende Fenster automatisch geschlossen.



MERKE DIR...

Du schreibst ein Programm für den Calliope im TigerJython-Editor. Um das Programm auf dem Calliope auszuführen, klickst du auf die Schaltfläche *Hinunterladen/Ausführen*. Die Programmausführung im Simulationsmodus wird mit Klick auf den grünen Pfeil gestartet.

Simulation Download auf Calliope



Auf dem Calliope ist jeweils das zuletzt hinuntergeladene Programm gespeichert. Die

Programmausführung startet automatisch beim nächsten Anschluss einer Stromversorgung. Sollte einmal das gespeicherte Programm wegen eines Programmierfehlers den Calliope blockieren, so musst du den Calliope neu flashen.

ZUM SELBST LÖSEN

1. Schreibe verschiedene Meldungen als Scrolltext aus (auch kleine Buchstaben sind erlaubt).

2. Schreibe ein Mitteilung aus, die sich endlos wiederholt. Verwende dazu in scroll() den zusätzlichen Parameter loop = True.

   Das Programm läuft endlos. Du kannst es allerdings im Terminalfenster mit Ctrl+C abbrechen (und mit Ctrl+D wieder starten).

ZUSATZBEMERKUNGEN

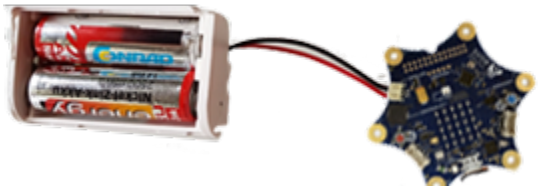
VERWENDUNG DES ONLINE-EDITORS

Die meisten Beispiele und Aufgaben kannst du auch im Browser mit TigerJython für den Web: https://webtigerpython.ethz.ch ausühren (Anleitung siehe unter https://python-online.ch/calliope).

EXTERNE STROMVERSORGUNG

Dein Programm bleibt so lange auf dem Calliope gespeichert, bis du es mit einem neuen Programm überscheibst (oder der Calliope geflasht wird). Du kannst also den Calliope beim Computer ausstecken und an eine andere Spannungsquelle, beispielsweise an ein USB-Ladegerät oder eine PowerBank, anschliessen.

Am zusätzlichen Batteriestecker kannst du auch einen Batteriehalter mit zwei 1.5V-Batterien oder Nickel-Zink-Akkus anschliessen (Achte sorgfältig auf die richtige Polarität!)



Sofort startet das zuletzt gespeicherte Programm wieder.

Um ein Programm mehrmals auszuführen, kannst du auch den Reset-Button klicken, der sich neben der USB-Buchse befindet, statt die Spannungsversorgung zu unterbrechen.

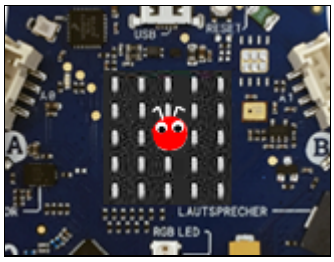
DU LERNST HIER...

das Wichtigste zum Programmieren mit Python an einfachen Beispielen mit einen ***Leuchtkäfer*** (***Glowbug***), den du mit den Befehlen *forward(), left(), right()* und *back()* auf dem Calliope-Display bewegen kannst. Falls du dich bereits im Kapitel *Turtlegrafik* in Python eingearbeitet oder äquivalente Grundkenntnisse von Python hast, kannst du dieses Kapitel überspringen und direkt zum nächsten Menüpunkt übergehen.

PROGRAMMSTRUKTUR *SEQUENZ*

Ein Computerprogramm besteht aus einer Folge von Programmzeilen, die der Reihe nach (als ***Sequenz***) abgearbeitet werden. Damit du die Käferbefehle verwenden kannst, musst du mit *from cpglow import* * das Modul *cpglow* importieren.

Mit dem Befehl makeGlow() erzeugst du einen sichtbaren Käfer in der Mitte des Displays. Es ist eine einzelne leuchtende LED. Bei einer Bewegung des Käfers werden LEDs an den besuchten Positionen eingeschaltet, er hinterlässt also sozusagen eine Spur.



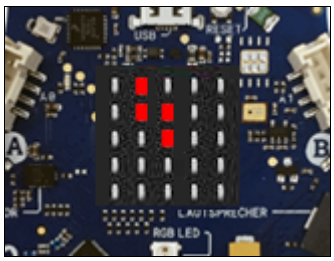
Deine Befehle werden nach dem Herunterladen des Programms auf den Calliope ausgeführt.

Die Befehle werden grundsätzlich Englisch geschrieben und enden immer mit einer Parameterklammer. Diese kann weitere Angaben für den Befehl enthalten. Die Gross-/ Kleinschreibung musst du exakt einhalten.

Mit forward() bewegt sich der Käfer einen Schritt vorwärts in der aktuellen Bewegungsrichtung. Bei Programmstart zeigt die Bewegungsrichtung nach oben. Mit left(90) dreht der Käfer um 90 Grad nach links und mit right(90) um 90° nach rechts. Dies macht sich aber erst beim nächsten *forward()*-Befehl bemerkbar.

Mit deinem Programm zeichnet der Käfer die daneben abgebildete Spur:

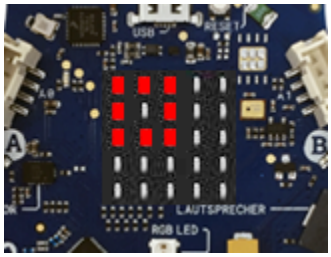
```
from cpglow import *

makeGlow()

forward()
left(90)
forward()
right(90)
forward()
```



Du kannst das Programm eintippen oder aus der Vorlage kopieren. Dazu klickst du auf *In Zwischenablage kopieren* und fügst es mit *Ctrl+V* in das TigerJython-Fenster ein.

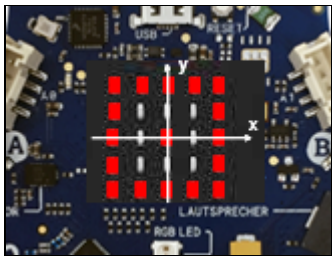
Um das Programm auf den Calliope herunterzuladen und dort auszuführen, klickst du dann in der Taskleiste auf den schwarzen Button *(Hinunterladen/ Ausführen*).

WIEDERHOLUNG MIT *REPEAT*



Um ein Quadrat zu durchlaufen, muss der Käfer vier Mal die Befehle *forward()* und *left(90)* ausführen. Du ersparst dir viel Schreibarbeit, wenn du eine Wiederholschleife verwendest. Du schreibst einfach repeat 4: (der Doppelpunkt ist wichtig). Die Befehle. die wiederholt werden sollen, müssen alle **gleichweit einrückt** sein. Du verwendest dazu vier Leerschläge oder die Tabulator-Taste.

```
from cpglow import *

makeGlow()
repeat 4:
    forward()
    forward()
    left(90)
```



Du willst ein möglichst grosses Quadrat zeichnen. Dazu setzt du mit setPos(-2, -2) den *Käfer* in die linke untere Ecke. Mit ***setPos(x, y)*** wird also der Käfer an eine beliebige neue Position gesetzt. Damit er sichtbar ist, musst du für x und y -2, -1, 0, 1 oder 2 wählen.

Um eine Quadratseite zu zeichnen, bewegst du den Käfer 4 mal vorwärts. Auch für diese 4 Vorwärtsschritte kannst du eine *repeat*-Schleife verwenden. Dein Programm hat also zwei ineinander geschachtelte repeat-Schleifen.

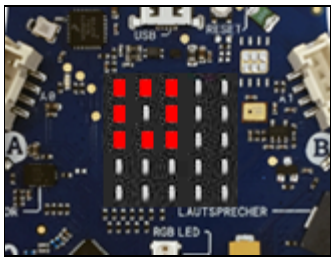
Achte auf eine korrekte Einrückung!

```
from cpglow import *

makeGlow()

setPos(-2, -2)
repeat 4:
    repeat 4:
        forward()
    right(90)
```

FUNKTIONEN (BENANNTE PROGRAMMBLÖCKE)

Mit benannten Programmblocks, in Python ***Funktionen*** genannt, machst du deine Programme übersichtlicher. Zudem kannst du Funktionen mehrmals **aufrufen**. Die Verwendung von Funktionen ist von grosser Wichtigkeit, denn du vermeidest dadurch, dass du gleichen Code mehrmals im Programm hinschreiben musst (Codeduplikation) und du kannst Probleme in kleinere Probleme zerlegen.

In deinem Beispiel definierst du eine Funktion *square()*, die den Käfer auf einem Quadrat bewegt.

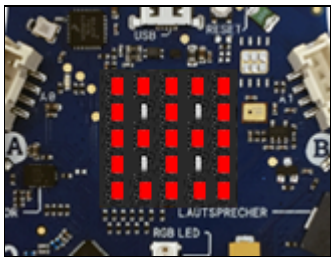


Die Funktionsdefinition beginnt mit dem Keyword ***def***. Dann folgen der Funktionsname, eine Parameterklammer und ein Doppelpunkt. . Die Befehle im Funktionskörper müssen eingerückt sein. Beachte, dass die Befehle erst beim Funktionsaufruf ausgeführt werden. Die Funktionsdefinitionen stehen jeweils im oberen Teil des Programms.

Wähle als Funktionsnamen einen Bezeichner, dass etwas darüber aussagt, was die Funktion macht. Du darfst keine Spezialzeichen (Leerschläge, Umlaute, Akzente, Satzzeichen, usw.) verwenden. Beginne den Namen immer mit einem kleinen Buchstaben. Im folgenden Programm definierst du eine Funktion *square()*, die ein Quadrat zeichnet, und führst sie dann aus.

```
from cpglow import *

makeGlow()

def square():
    repeat 4:
        forward()
        forward()
        left(90)

square()
```

Mit der Definition von *square()* hast du sozusagen einen neuen Befehl eingeführt, den du nun beliebig oft verwenden kannst.. Um das nebenstehende Bild zu erzeugen rufst du die Funktion *square()* 4-mal auf.

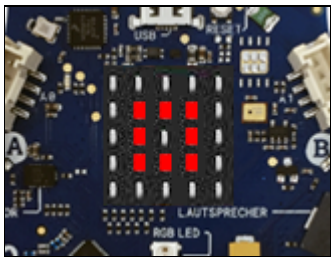


Mit dem Befehl setSpeed(80) läuft den Käfer schneller. (Der Standardwert ist 50 und es sind Werte im Bereich 0 bis 100 erlaubt.)

```
from cpglow import *

makeGlow()

def square():
    repeat 4:
        forward()
        forward()
        left(90)

setSpeed(80)
repeat 4:
    square()
    right(90)
```

VARIABLEN

In der Informatik sind Variablen Platzhalter für Werte, die sich im Laufe der Programmausführung ändern. v ist die Geschwindigkeit des Käfers. Mit v = 30 setzst du v auf den Anfangswert 30. Man nennt diesen Befehl eine **Zuweisung**. Dieser wird nach jedem durchgelaufenen Quadrat um 20 vergrössert.



Dazu verwendest du den Befehl v = v + 20, der auf den ersten Blick wie eine unsinnige Gleichung aussieht, aber dem Computer sagt, er soll den bestehenden Wert von v holen und 20 dazu zählen, und nachher das Resultat wieder in v abspeichern. Der Befehl showTrace(False) bewirkt, dass der Käfer keine Spuren hinterlässt.

```
from cpglow import *

makeGlow()

def square():
    repeat 4:
        forward()
        forward()
        right(90)

v = 30
showTrace(False)
setPos(-1, -1)

repeat 4:
    setSpeed(v)
    square()
    v = v + 20
```

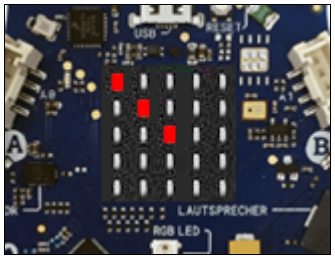
WIEDERHOLUNG MIT WHILE

Die *while-Schleife* ist eine der wichtigsten Programmstrukturen überhaupt. Sie kann allgemein für jede Art von Wiederholungen verwendet werden und kommt in praktisch allen Programmiersprachen vor. Eine *while-Schleife* wird mit dem Schlüsselwort while eingeleitet gefolgt von einer Bedingung und einem Doppelpunkt. So lange die Bedingung erfüllt ist, werden die Befehle im nachfolgenden Programmblock wiederholt. Umgangssprachlich würde man dies wie folgt ausdrücken:

Solange die *Bedingung wahr,* führe aus ...

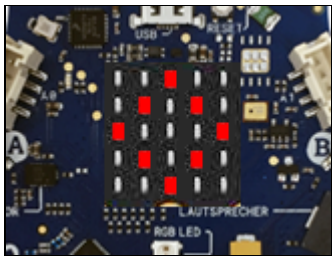
In der Bedingung werden in der Regel die Vergleichsoperatoren < (kleiner), <= (kleiner-gleich), > (grösser) >= (grösser-gleich), == (gleich), != (verschieden) verwendet. Beachte die Verdoppelung des Gleichheitszeichens beim Test auf Gleichheit (damit es nicht mit einer Zuweisung verwechselt wird).

In deinem Beispiel bewegt sich der Käfer zuerst 2 Schritte vorwärts, dann 2 Schritte rückwärts, dreht um 45°, danach wieder 2 Schritte vorwärts usw., so lange, bis er wieder gegen Norden schaut. Den Gesamtdrehwinkel speicherst du in der Variablen a. Du beginnst mit a = 0 und zählst nach jedem Schleifendurchgang 45 dazu. So lange die Bedingung a <= 360 stimmt, werden die eingerückten Anweisungen ausgeführt.



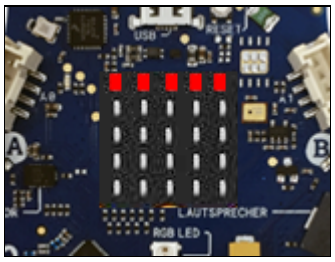

```
from cpglow import *

makeGlow()

setSpeed(90)
showTrace(False)
a = 0
while a <= 360:
    forward()
    forward()
    back()
    back()
    left(45)
    a = a + 45
```

In der Robotik wird häufig eine sogenannte "endlose while-Schleife" verwendet. Diese wird mit while True: eingeleitet. Da *True* immer wahr ist, wird die Schleife so lange ausgeführt, bis du die Programmausführung im nebenstehenden Konsolenfenster mit Ctrl+C beendest oder die Spannungsversorgung des Calliope unterbrichst. clear() löscht alle LEDs.




```
from cpglow  import *

makeGlow()

def square():
    repeat 4:
        forward()
        forward()
        left(90)

clear()
setSpeed(90)
showTrace(False)
setPos(2, 0)
left(45)
while True:
    square()
```

WIEDERHOLUNG MIT FOR IN RANGE



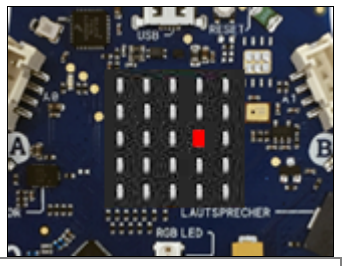
Oft brauchst du in einer Wiederholschleife eine ganzzahlige Variable, die bei jedem Durchgang um eins grösser wird. Du kannst dies zwar mit einer *while*-Schleife lösen, einfacher geht es aber mit einer *for*-Schleife, bei welcher der Schleifenzähler automatisch verändert wird.
for i in range(n) durchläuft Zahlen *i* von 0 bis *n-1*
for i in range(a, b) durchläuft Zahlen i von a bis b-1
Beachte, dass der Endwert n bzw. b nie enthalten ist.

Mit for x in range(-2, 3) durchläuft also x die Werte -2, -1, 0, 1, 2. Dabei werden die LEDs in der obersten Reihe nacheinander eingeschaltet. Mit sleep(500) kannst du die Programmausführung 500 Millisekunden anhalten, damit du den Ablauf besser beobachten kannst.

```
from cpglow import *

makeGlow()

clear()
for x in range(-2, 3):
    setPos(x, 2)
    sleep(500)
```

IF - ELSE - STRUKTUR (SELEKTION)

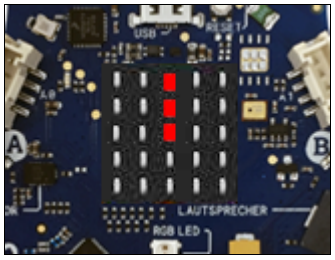
Mit der Selektion kannst du bewirken, dass bestimmte Programmblöcke nur unter gewissen Bedingungen ausgeführt werden. Die Selektion wird mit dem Schlüsselwort ***if*** eingeleitet, gefolgt von einer Bedingung. Die Anweisungen im ***if*** werden nur dann ausgeführt, wenn die Bedingung wahr ist, sonst werden die Anweisungen nach ***else*** ausgeführt. In der ***if-***Bedingung werden üblicherweise die Vergleichsoperatoren >, >= , < , <= , == , != verwendet. Die Anweisungen im *if-* bzw. *else*-Block müssen eingerückt sein. Der *else*-Teil kann auch wegfallen.



In diesem Beispiel bewegt sich der Käfer nach rechts. Wenn er am rechten Rand angekommen ist, springt er wieder zum linken Rand und wiederholt diese Bewegung von links nach rechts endlos. Du bewegst also den Käfer mit *forward()* nach rechts und überprüfst nach jedem Schritt seine x-Koordinate. Wenn sie 3 ist, so wird der Käfer wieder auf den linken Rand versetzt (x = -2).

```
from cpglow import *
makeGlow()
right(90)
x = 0
showTrace(False)
while True:
    forward()
    x = x + 1
    if x == 3:
        setPos(-2, 0)
        x = -2
```

Im nächsten Programm soll der Käfer zufällig zwei Schritte vorwärts oder zwei Schritte rückwärts laufen, dann kurz anhalten, die Spur löschen und an die Anfangsposition zurückkehren. Diese Bewegung soll er 10 Mal wiederholen.



Du verwendest Zufallszahlen. Die Funktion *randint(a, b)* aus dem Modul *random* erzeugt zufällig eine ganze Zahl zwischen a und bis und mit b.

Daher liefert die Funktion randint(0, 1) zufällig eine 0 oder eine 1. Ist der Wert 1, so läuft der Käfer 2 Schritte vorwärts, sonst (wenn die Zahl 0 ist) 2 Schritte rückwärts.

```
from cpglow import *
from random import randint

makeGlow()
setSpeed(80)
repeat 10:
    r = randint(0, 1)
    if r == 1:
        forward()
        forward()
    else:
        back()
        back()
    sleep(300)
    clear()
    setPos(0, 0)
```

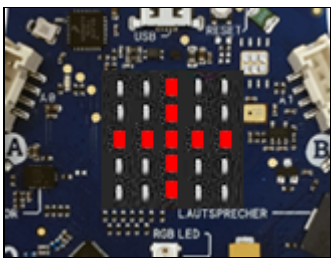
MERKE DIR...

Die grundlegenden Programmstrukturen sind Sequenz, Wiederholung und Selektion. Bei einer **Sequenz** werden die Befehle der Reihe nach abgearbeitet. Mit einer **Wiederholung** werden Programmblocks mehrmals ausgeführt. Im TigerJython kannst du dazu die ***repeat-***; **while**- oder ***for***-Schleifen verwenden. Die **Selektion** mit ***if*** und ***else*** bewirkt, dass bestimmte Anweisungen nur unter gewissen Bedingungen ausgeführt werden. Der *else*-Teil kann auch entfallen.

Funktionen sind wichtig für die **strukturierte Programmierung**. Sie vermeiden die Codeduplikation und dienen dazu, Probleme in kleinere Teilprobleme zu zerlegen. Man spricht auch von **Modularisierung**.

ZUM SELBST LÖSEN

1. Schreibe ein Programm mit einer Wiederholstruktur *repeat*, so dass der Leuchtkäfer ein Pluszeichen zeichnet.



2. LED-DISPLAY

DU LERNST HIER...

wie man mit dem LED-Display Textmeldungen und einfache Bilder anzeigen kann.

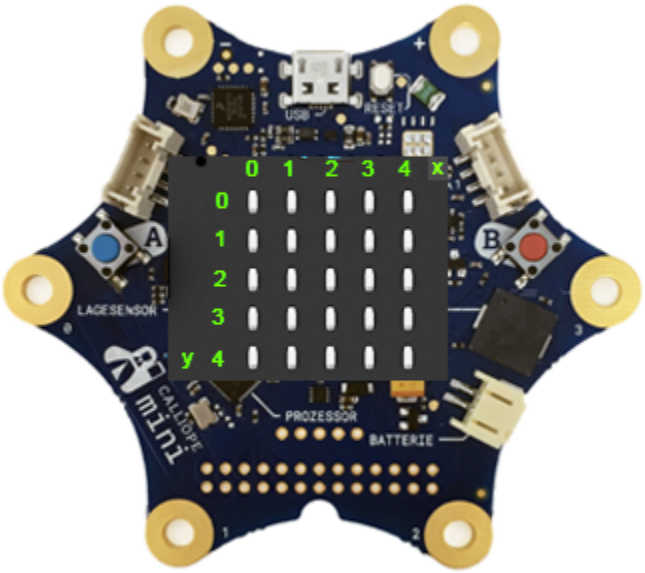
25 LEDs

Die 25 LEDs des Displays sind in einer 5x5 Matrix angeordnet und über ihre x- und y-Position einzeln ansprechbar. Mit

display.set_pixel(x, y, n)

leuchtet die LED an der Position x, y mit der Helligkeit n, wobei n eine Zahl von 0 bis 9 ist. Für n = 9 ist die LED am hellsten, für n = 0 ist sie ausgeschaltet.

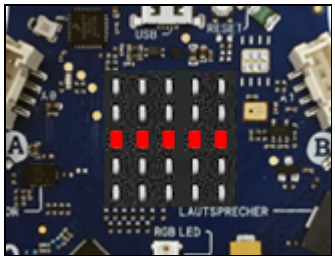
Du kannst alle LEDs mit ***display.clear()*** ausschalten.



MUSTERBEISPIELE

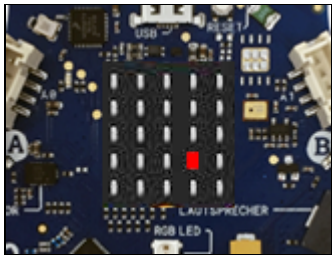
LEDs ein- und ausschalten

Dein Programm soll die LEDs in der mittleren Reihe der Reihe nach ein- und ausschalten. Der Befehl *sleep(400)* hält das Programm 400 Millisekunden an. Dadurch bleibt die gerade leuchtende LED während dieser Zeit eingeschaltet.




```
from calliope_mini import *

for x in range(5):
    display.set_pixel(x, 2, 9)
    sleep(400)
    display.set_pixel(x, 2, 0)
```

Damit du mit Zufallszahlen programmieren lernst, soll in diesem Beispiel wiederholt eine zufällig ausgewählte LED ein- und ausgeschaltet werden. In der Funktion *randomLed()* werden zuerst alle LEDs gelöscht,und nachher zwei Zahlen zwischen 0 und 4 gewählt und die LED mit diesen Koordinaten mit maximaler Helligkeit eingeschaltet.

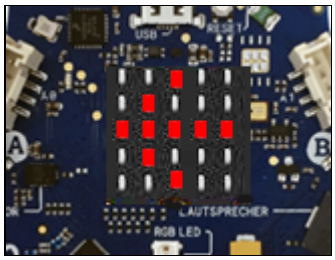


Im Hauptprogramm wird die Funktion *randomLed()* in einer Endlosschleife alle 200 Millisekunden aufgerufen. Das Programm läuft also so lange, bis du ein neues Programm hinunterlädst, den Resetknopf (Button neben dem USB-Anschluss) drückst oder die Stromversorgung unterbrichst.

```
from calliope_mini import *
from random import randint

def randomLed():
    display.clear()
    x = randint(0, 4)
    y = randint(0, 4)
    display.set_pixel(x, y, 9)

while True:
    randomLed()
    sleep(200)
```

Images anzeigen

Im Modul calliope_mini sind mehrere 5x5 Pixelbilder gespeichert, die du auf dem Display anzeigen kannst. Die Bildnamen siehst du in der Dokumentation.



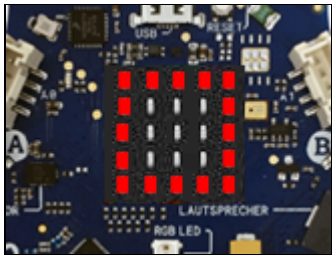
Mit dem Befehl ***display.show()*** kannst du diese Bilder anzeigen. In diesem Beispiel werden nacheinander Pfeile angezeigt, die nach Norden, Osten, Süden und Westen zeigen.

Jedes Bild wird 1 Sekunde lang angezeigt. Mit einer for-Schleife wiederholst du den Vorgang dreimal.

```
from calliope_mini import *

for i in range(3):
    display.show(Image.ARROW_N)
    sleep(1000)
    display.show(Image.ARROW_E)
    sleep(1000)
    display.show(Image.ARROW_S)
    sleep(1000)
    display.show(Image.ARROW_W)
    sleep(1000)
```

Eigene Images erstellen

Du kannst auch eigene Images erstellen, indem du als Parameter von Image einen String angibst, der aus 5 Blöcken mit je 5 Zahlen 0 bis 9 besteht. Die Blöcke entsprechen den einzelnen Zeilen. Die Zahlen legen die Helligkeit im Bereich 0 (dunkel) bis 9 (ganz hell) fest. Das folgende Programm zeigt ein Quadrat mit den Randpixeln in voller Helligkeit.



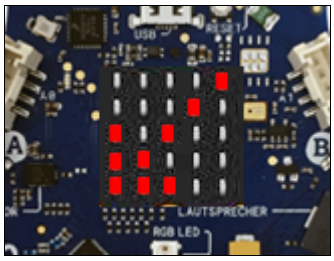
```
from calliope_mini import *

img = Image('99999:90009:90009:90009:99999:')
display.show(img)
```

MERKE DIR...

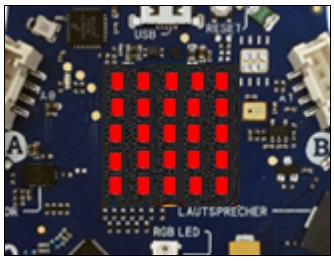
Zur Ansteuerung des LED-Displays verwendest du das Objekt *display* mit den Funktionen *set_pixel(), show(), scroll()* und *clear().* Du orientierst dich am besten im Menü Menü von unter *Hilfe* | *APLU-Dokumentation* | *Calliope* | *MicroPython API* über die erlaubten Parameter.

ZUM SELBST LÖSEN

1. Schreibe ein Programm, welches nacheinander vier diagonale Pfeile darstellt:
   Image.ARROW_NW
   Image_ARROW_NE
   Image.ARROW_SW
   Image.ARROW_SE.



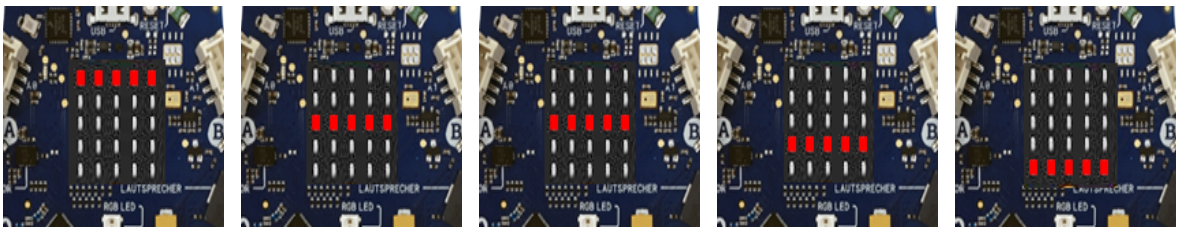
2. Mit einer doppelten for-Schleife



```
for y in range(5):
    for x in range(5):
        display.set_pixel(x, y, 9)
        sleep(400)
```

   kannst du alle LEDs der Reihe nach einschalten. Die LEDs sollen danach alle gleichzeitig mit dem Befehl *clear()* ausgeschaltet werden und zwar 1000 Millisekunden, nachdem die letzte LED eingeschaltet wurde.

3. Schreibe ein Programm, welches zuerst die oberste LED-Reihe, dann die zweite, dritte usw. einschaltet. Lasse den Vorgang in einer Endlosschleife ablaufen.



3. LAGESENSOR

DU LERNST HIER...

wie du mit dem Lagesensor Lageänderungen und Bewegungen des Calliope erfassen kannst.

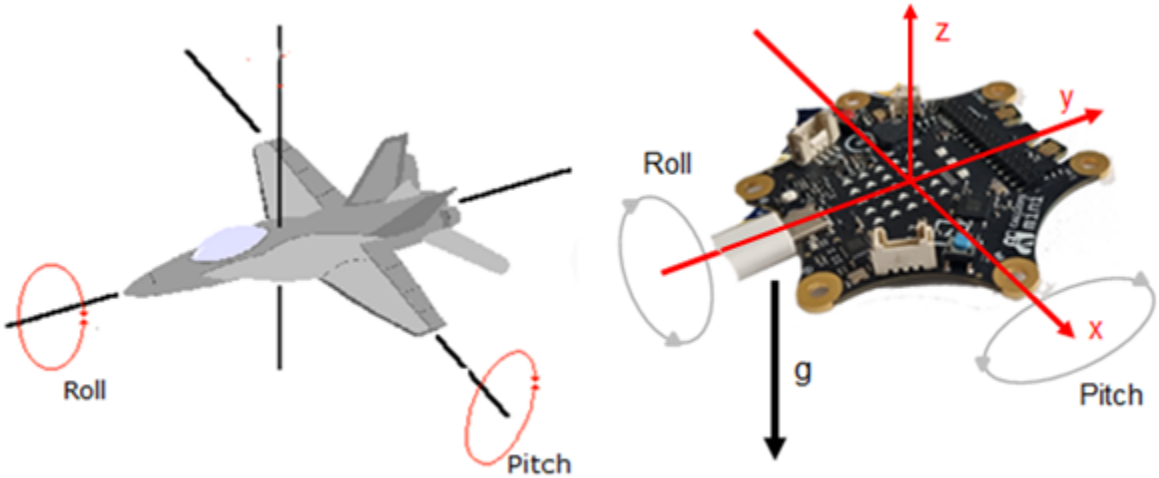
SENSORWERTE

Der Lagesensor (Beschleunigungssensor) misst sowohl die konstante Erdbeschleunigung von rund 10 m/s^2, die vertikal nach unten zeigt, als auch die Beschleunigungen, die durch Bewegungen zustande kommen.

Der Lagesensor ist auf dem Board gut sichtbar. Ähnliche Sensoren sind auch in den meisten Smartphones eingebaut.

Der Sensor misst die x-, y, und z-Komponente der Beschleunigung, aus welchen die Vorwärtsneigung (Pitch) und Seitwärtsneigung (Roll) berechnet werden kann.

Es ist ähnlich wie bei der Lageanzeige von Flugzeugen, die mit einem künstlichen Horizont angezeigt werden. Wenn man das Board nach vorne oder nach hinten neigt, ändert der Pitch-Winkel, bei der Seitwärtsneigung ändert der Roll-Winkel.



Im folgendem Programm verwendest du die Befehle *accelerometer.get_x()*, *accelerometer.get_y()* oder *accelerometer.get_z()* , die Werte im Bereich von ungefähr -2000 bis 2000 liefern (entspricht den Beschleunigungen -20 m/s^2 bis 20 m/s^2). Mit *acceloremeter.get_values()* erhältst du alle drei Werte in einem Tupel zurück.

Wenn die z-Achse senkrecht nach unten zeigt, muss der Wert der Erdbeschleunigung, also 9.81 m/s^2 gemessen werden. Der Sensor gibt bei az umgefähr 981 zurück.

```
from calliope_mini import *
```

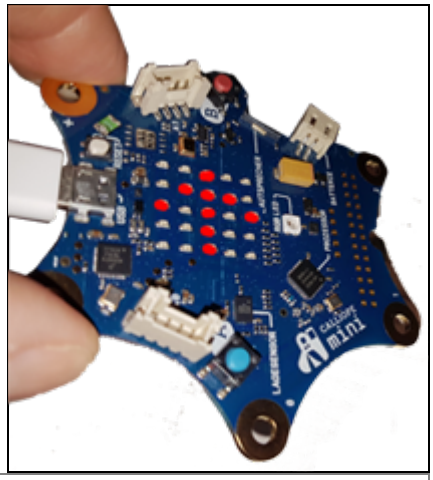
```
while True:
    ax = accelerometer.get_x()
    ay = accelerometer.get_y()
    az = accelerometer.get_z()
    a = accelerometer.get_values()
    print(ax, ay, az, a)
    sleep(500)
```

MUSTERBEISPIELE

Sensorwerte abfragen

Starte das unten stehende Programm, kippe den Calliope um den USB-Anschluss links und rechts und beobachte dabei die Sensorwerte, die mit *print(acc)* im Terminalfenster angezeigt werden.

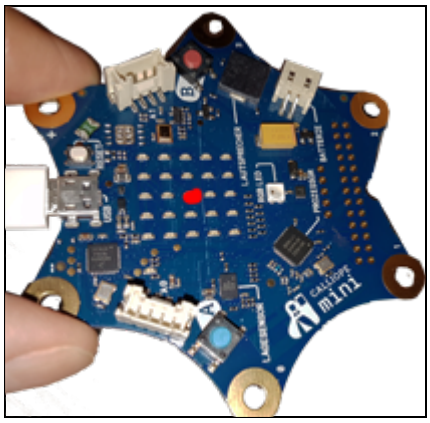
Auf dem Display wird bei einem positiven Wert der Pfeil ARROW_E und bei einem negativen Wert der Pfeil ARROW_W angezeigt.



```
from calliope_mini import *

while True:
    acc = accelerometer.get_x()
    print(acc)
    if acc > 0:
        display.show(Image.ARROW_E)
    else:
        display.show(Image.ARROW_W)
    sleep(100)
```

Wasserwaage-Libelle

Der Calliope soll wie eine Wasserwaage-Libelle funktionieren. Dabei verwendest du die Tatsache, dass die x- und y-Komponenten der Beschleunigungen ungefähr 0 sind, wenn sich das Board in waagrechter Lage befindet.

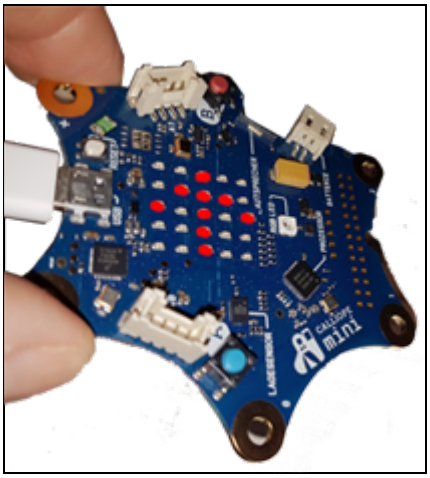


Es wird nur ein Pixel angezeigt, das sich je nach Neigung nach rechts, links, oben oder unten verschiebt. Ziel ist es, die Libelle so auszurichten, dass die mittlere LED leuchtet.

Im Programm verwendest du vier Bedingungen, um den x- und y-Wert des Pixels zu bestimmen und löschst dann das Pixel bevor du das neue anzündest.

```
from calliope_mini import *

x = 1
y = 1

while True:
    accX = accelerometer.get_x()
    accY = accelerometer.get_y()
    if accX > 100 and y < 4:
        y += 1
    elif accX < -100 and y > 0:
        y -= 1
    elif accY > 100 and x < 4:
        x += 1
    elif accY < -100 and x > 0:
        x -= 1
    display.clear()
    display.set_pixel(x, y, 9)
    sleep(100)
```

Pitch und Roll



Im Modul *cputils* befinden sich zwei Funktionen, die aus der Beschleunigunskomponenten Pitch bzw. Roll berechnen.

Du verwendest in deinem Programm die Funktion *getRoll()*, die den Rollwinkel in Grad zurückgibt und machst die Rollposition auf dem Display sichtbar. Drehst du den Calliope nach rechts, so erscheint auf dem Display ein Rechtspfeil, drehst du ihn nach links, erscheint ein Linkspfeil.

```
from calliope_mini import *
from math import*

def getRoll(a):
    anorm = sqrt(a[0] * a[0] + a[1] * a[1] + a[2] * a[2])
    roll = asin(a[0] / anorm)
    return int(degrees(roll))

while True:
    a = accelerometer.get_values()
    roll = getRoll(a)
    print(roll)
    if roll >= 0:
        display.show(Image.ARROW_N)
    if roll <= 0:
        display.show(Image.ARROW_S)
    sleep(0.3)
```

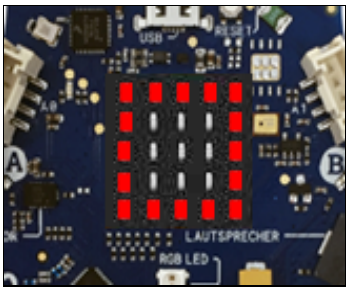
MERKE DIR...

Mit dem Beschleunigungssensor kannst du die Bewegungen und die Lage des Calliope erfassen. Meist werden die Sensorwerte in einer Messschleife regelmässig alle 10 - 100 Millisekunden abgefragt. Man nennt dies auch *Pollen der Sensorwerte*.

ZUM SELBST LÖSEN

1. Ergänze das erste Beispiel so, dass auch die Drehung des Calliope in der y-Richtung gemessen und mit den passenden Pfeilen Image.ARROW_N, bzw.Image.ARROW_S angezeigt wird. Beachte, dass die Werte nicht eindeutig < 0 bzw. < 0 ausfallen. Eine Möglichkeit ist es., die passenden Pfeile anzuzeigen, wenn der Sensorwert < 100 bzw. < -100 ist. Dann kannst du den Calliope eideutig in einer Richtung kippen.

2. Durch Kippen des Calliope sollen alle Pixel am Rand einzeln eingeschaltet werden.



3. Du willst das "Schütteln" des Calliope messen. Dazu holst du mit *a = accelerometer.get_values()* die Beschleunigungskomponenten, betrachtest aber nur den Betrag b der x- und y-Komponenten, der sich wie folgt berechnet

   $b = \sqrt{a[0] * a[0] + a[1] * a[1]}$

   a) Schreibe im Terminalfenster den Betrag in einer Endlosschleife 50x pro Sekunde aus.

   b) Führe nach Start des Programm die Messungen nur während 1 Sekunde aus und schreibe das Maximum von b aus.

   c) Schreibe das Maximum als Integer auf dem Display aus (als Scrolltext).
   Anleitung: Um aus b einen Integer zu machen, verwendest du *n = int(b)*. Für den Display musst du daraus mit *s = str(n)* einen String (Text) machen.

   d) Mache daraus ein "Schüttelspiel", bei dem derjenige gewinnt, der am stärksten schüttelt. Die Messung soll aber erst beim Erscheinen das Zeichens > nach einer zufälligen Zeit von 3-6 Sekunden beginnen und man darf vorher nicht bereits schütteln.

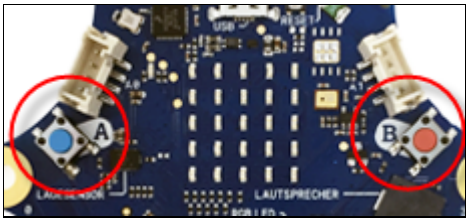
4. BUTTONS

DU LERNST HIER...

wie man die beiden Tastenschalter (Buttons) verwendet, um interaktive Programme zu entwickeln.

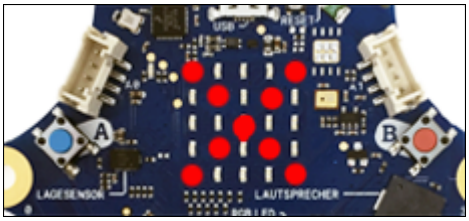
AUF DRÜCKEN EINES BUTTONS REAGIEREN

Der Calliope verfügt über zwei programmierbare Buttons. Der blaue wird mit A, der rote mit *B* bezeichnet und mit den Objektvariablen button_a bzw. button_b angesprochen.



Die Funktion ***button_a.is_pressed()*** gibt *True* zurück, wenn der linke Button im Moment des Aufrufs gedrückt ist (analog *button_b_is pressed()*).

Dein Programm wartet in einer Endlosschleife, bis du den Button A drückst. Dann werden die im Kreuz angeordneten LEDs eingeschaltet und zwar so lange du den Button gedrückt hältst.




```
from calliope_mini import *

while True:
    if button_a.is_pressed():
        display.show(Image.NO)
    else:
        display.clear()
    sleep(10)
```

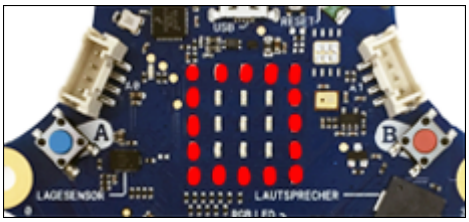
<

Das überflüssig erscheinende *sleep(10)* ist wichtig, damit du nicht unnötig viele Rechnerressourcen verschwendest, wenn das Programm nichts anderes machen muss, als zu überprüfen, ob der Button gedrückt ist. In der Fachsprache sagt man auch, dass der Zustand des Buttons in der Endlosschleife "*gepollt*" wird.

AUF KLICKEN EINES BUTTONS REAGIEREN

So wie du es von den Mausklicks kennst, möchtest du hier mit einem Button-Klick (nicht mit gedrückt halten) eine Aktion auslösen. Dazu verwendest du die Funktion ***button_a.was_pressed()***. Diese liefert True, wenn du irgendeinmal den Button kurz gedrückt hast.

In deinem Beispiel wird beim Klick auf den Button A das Image SQUARE und beim klicken des Buttons B das Image NO angezeigt. Im Unterschied zum ersten Beispiel, musst du die Buttons nicht gedrückt halten.




```
from calliope_mini import *

while True:
    if button_a.was_pressed():
        display.show(Image.SQUARE)
    if button_b.was_pressed():
        display.show(Image.NO)
    sleep(10)
```

So wie du es von den Mausklicks kennst, kannst du auch hier mit einem Button-Klick ein laufendes Programm unterbrechen und eine andere Aktion ausführen. In einer endlosen while-Schleife lässt du die mittlere LED mit einer Periode von 200 ms blinken. Mit einem Klick auf den Button A wird das Blinken unterbrochen und während 1000 ms ein Quadrat anzeigen. Dann wird das Programm mit Blinken wieder fortgesetzt.

Du verwendest auch hier die Funktion ***button_a.was_pressed()***. Der Klick wird als **Event** aufgefasst, der vom System auch dann registriert wird, wenn dein Programm gerade etwas anderes macht.

```
from calliope_mini import *

def blink(x, y):
    display.set_pixel(x, y, 9)
    sleep(500)
    display.set_pixel(x, y, 0)
    sleep(500)

while True:
    if button_a.was_pressed():
        display.show(Image.SQUARE)
        sleep(1000)
        display.clear()
    blink(2, 2)
    sleep(10)
```

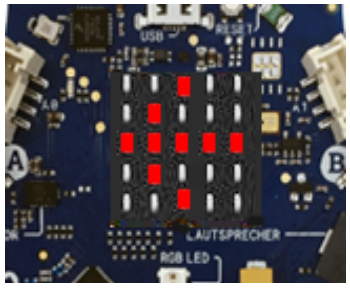
MERKE DIR...

Du kannst interaktive Programme entwickeln, die auf einen gedrückt gehaltenen Button oder auf einen Button-Klick reagieren. Mit der Funktion ***is_pressed()*** muss der Button gedrückt sein, damit sie True zurückgibt, mit der Funktion ***was_pressed()*** wird True zurückgegeben, wenn seit dem dem Start des Programms oder seit dem letzten Buttonklick irgendwann mal geklickt wurde.

ZUM SELBST LÖSEN

1. Programmiere die Buttons wie folgt:

   Wenn du den Button A drückst, so erscheint ein Pfeil, der nach links zeigt (ARROW_W), wenn du den rechten Button drückst, so erscheint ein Pfeil nach rechts (ARROW_E).

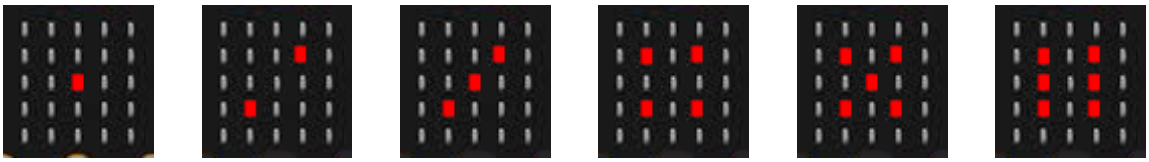


2. Programmiere einen Klickzähler: Bei jedem Klick auf den Button A wird die Anzahl n, die zu Beginn 0 ist, um eins vergrössert und zur Kontrolle im Terminalfenster ausgeschrieben. Beim Klick auf den Button B soll die Totalzahl als Lauftext auf dem Display erscheinen.

   Bemerkung Der Befehl *scroll()* kann nur Texte anzeigen. Mit *str(n)* kannst du eine Zahl in einen Text umwandeln.

3. Du simulierst mit dem Calliope das Werfen eines Würfels. Bei jedem Klicken auf den Button A wird auf dem Display das Bild einer zufälligen Würfelseite dargestellt.

   Anmerkung:
   Am einfachsten erstellst du die Bilder als Images z. B. die Vier mit
   img = Image('00000:09090:00000:09090:00000:')



5. SOUND

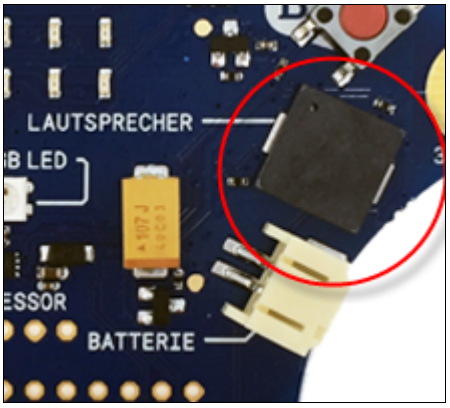
DU LERNST HIER...

wie du mit dem Calliope Töne, Tonfolgen und kurze Melodien abzuspielen kannst.

MUSTERBEISPIELE

Der Calliope verfügt über einen eingebauten Lautsprecher, der auf dem Bord gut sichtbar ist.

Um Töne und Melodien abzuspielen, importierst du das Modul *music*.



Eine Tonfolge abspielen
Mit dem Befehl pitch(f, 500) wird ein Ton mit der Frequenz f während 500 Millisekunden abgespielt. Um mehrere Töne nacheinander abzuspielen, gibst du die zugehörigen Frequenzen in einer Liste an und durchläufst diese mit einer *for*-Schleife. Eine Tabelle mit Tönen und ihren Frequenzen findest du im Overlay-Fenster: Ein Ton mit der Frequenz 0 ist gleichbedeutend mit einer Pause.

```
from music import *

song = [262, 294, 330, 349, 392, 392, 392, 0, 440, 440, 440, 440, 392]
for f in song:
   pitch(f, 500)
```

Eingebaute Melodien abspielen
Einige Melodien sind bereits eingebaut. Ihre Namen findest du hier oder in der Dokumentation. Du kannst das Abspielen der Melodien mit der Verwendung der Buttons kombinieren. Wenn du den Button A drückst, wird die Melodie JUMP_UP abgespielt, beim Drücken des Buttons B die Melodie JUMP_DOWN.

```
from calliope_mini import *
from music import *
while True:
    if button_a.was_pressed():
        play(JUMP_UP)
    if button_b.was_pressed():
        play(JUMP_DOWN)
    sleep(10)
```

Was passiert, wenn du die beiden Button miteinander gedrückt hältst?

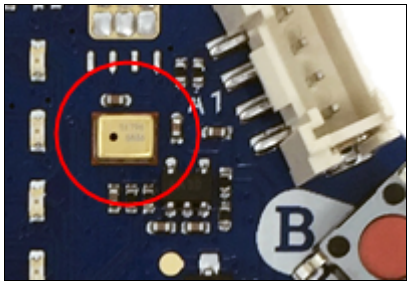
Ein akustisches Lagemessgerät

Messgeräte mit akustischen Ausgaben sind sehr beliebt. Du erzeugst hier ein Tonsignal, dessen Höhe von der Seitwärtsneigung des Calliope abhängt. Ausgehend von einer Anfangsfrequenz f0 = 1000, berechnest du die steigenden bzw. fallenden Tonfrequenzen für Halbtöne der wohltemperierten Stimmung und spielst jeden Ton 100 Millisekunden lang ab. Mit dem Button B kannst du das Tonsignal ausschalten.

```
from music import *
from calliope_mini import *

def beep(n):
    freq = int(f0 * r**n)
    pitch(int(r**n * f0), 100)
    sleep(50)

f0 = 1000
r = 2**(1/12)
while not button_b.was_pressed():
    n = int(accelerometer.get_x() / 100)
    beep(n)
    sleep(10)
```

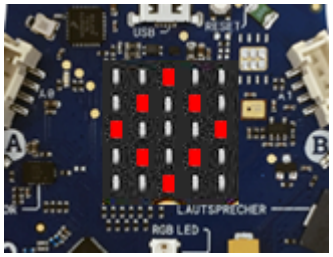
Mikrofon

Der Calliope besitzt ein Mikrofon, welches Geräusche detektieren kann. Das Mikrofon ist nur 3x4 mm gross. Die Funktion
isClicked(level , rearm_time
gibt True zurück ,falls der Schallpegel den vorgegebenen Pegel (level = 1...500) übersteigt. Die Detektion wird danach während der *rearm_time* unterdrückt



.

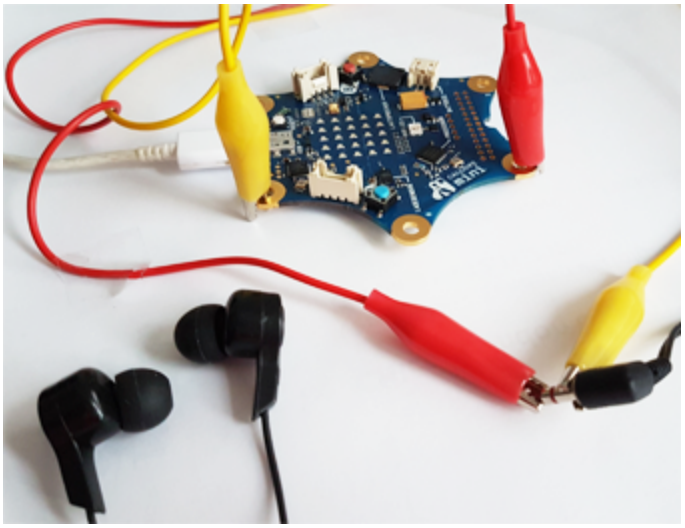
In deinem Programm wird der Schallpegel auf 20 gesetzt. Der Mikrofon reagiert bereits auf schwache Geräusche (Pfeifen, Klatschen, auf Tischklopfen usw.). Du musst aber warten bis das Programm gestartet ist und im Terminalfenster "starting" ausgeschrieben ist. Falls ein Geräusch detektiert wird, leuchten auf dem Display während 500 Millisekunden einige LEDs.



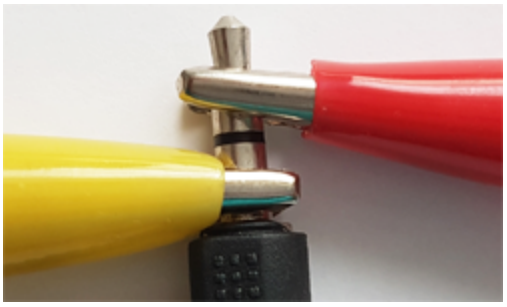

```
from calliope_mini import *
from cpmike import *

print("starting")
while True:
    if isClicked(level = 20, rearm_time = 800):
        display.show(Image.DIAMOND)
        sleep(500)
        display.clear()
```

Kopfhörer oder Lautsprecher anschliessen

Anstelle des eingebauten Lautsprechers, der die Töne nur schlecht wiedergibt, kannst du einem externen Kopfhörer oder aktiven Lautsprecher (mit einem eingebauten Verstärker) an den Calliope anschliessen. Dazu brauchst du zwei Kabel mit Krokodilklemmen. Mit dem ersten Kabel (hier gelb) verbindest du den Pin **GND** mit dem hinteren Teil des Kopfhörersteckers. Mit dem zweiten Kabel (hier rot) verbindest du den **Pin1** mit dem vorderen Teil des Kopfhörersteckers.



Um den Sound über den Pin1 auszugeben, musst du das Modul calliope_mini importieren und in der Funktion *pitch()* oder *play()* den Parameter pin auf pin1 setzen.



Damit du den Sound in den beiden Hörmuscheln hörst, musst du versuchen, die rote Krokodilklemme so anzubringen, dass sie die beiden vorderen Teile des Steckers berührt.

```
from music import *
from calliope_mini import *

print("starting")
play(ENTERTAINER, pin = pin1)
```

MERKE DIR...

Den Kopfhörer musst du an die Ausgänge, die mit GND und P1 bezeichnet sind, anschliessen. Einen Ton mit der Frequenz f spielst du mit dem Befehl *pitch(f, time)* ab. Mit *play(song)* kannst du ganze Melodien, entweder als Tonfolge oder in musikalischer Notation abspielen.

Das eingebaute Mikrofon detektiert die Umgebungsgeräusche.

ZUM SELBST LÖSEN

1. a) Schreibe ein Programm, welches die folgende Tonfolge abspielt:

   song = [262, 294, 330, 262, 262, 294, 330, 262, 330, 349, 392, 330, 349, 392]

   Du kannst selbstverständlich auch eigene Tonfolgen komponieren.

   b) Spiele die Tonfolge schneller bzw. langsamer ab.

2. Du kannst eine Melodie wiederholt abspielen lassen, indem du *play()* mit zusätzlichen Parametern verwendest, beispielsweise für die Melodie BIRTHDAY

   play(BIRTHDAY, loop = True)

3. play(BIRTHDAY, wait = False, loop = True)

   Der Parameter *wait* = *False* bewirkt, dass das Abspielen im Hintergrund abläuft und das Programm weiter läuft. Du kannst also während des Abspielens z. Bsp. LEDs einschalten oder Buttons verwenden.

   Schreibe ein Programm, dass das Geburtstagslied solange endlos abspielt, bis du den Button A klickst. (Anmerkung: Du kannst eine Soundausgabe mit dem Befehl *stop()* beenden.)

4. Das Mikrofon deines Calliope reagiert wie folgt auf Geräusche: Jedesmal, wenn du auf den Tisch klopfst, wird die Melodie JUMP_UP abgespielt.

ZUSATZSTOFF

Melodien in musikalischer Notation abspielen

Die Notation hält sich an folgende Regeln:

- Die zwölf Halbtöne einer Oktave haben mit Kreuz geschrieben die Bezeichnungen C, C#, D, D#, E, F, F#, G, G#, A, A#, H (auch mit Kleinbuchstaben). Für die Pause verwendet man die Note "R"
- Will man die Oktave wechseln, so schreibt man eine Oktavezahl hinter die Note, also C2 für die zweite Oktave (zu Beginn ist man in der 4. Oktave). Diese Oktave gilt dann für alle folgenden Noten, bis wieder eine Oktavezahl angegeben wird
- Hinter der Note kann ein Doppelpunkt mit einer Angabe der Dauer (in Ticks) stehen. Alle folgenden Noten werden dann mit dieser Dauer abgespielt, bis wieder eine neue Angabe der Dauer folgt.

Um eine Melodie zu komponieren, schreibst du die Noten in eine Liste, beispielsweise im folgenden Programm für einen Dur- und nachfolgenden Moll-Akkord.

```
from music import *

melody = ['e:4', 'f#', 'g#', 'r:2', 'e', 'f#', 'g:4']
play(melody)
```

Morse

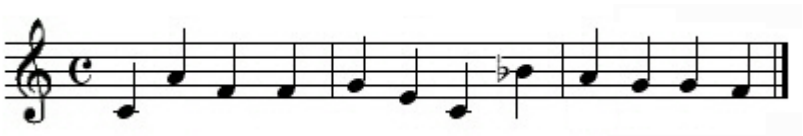
Der Morsecode wird zur Übermittlung von Nachrichten verwendet. Buchstaben, Satzzeichen und Zahlen sind als kurzes Signal, langes Signal und Pause codiert. Mit dem eingebauten Lautsprecher kannst du sehr einfach Morse-Nachrichten erzeugen. Es ist zweckmässig, den Code in einem Python-Dictionary abzulegen.

```
from calliope_mini import *
from music import *

dt = 100 # adapt to your speed
```

```
morse = {
'a':'.-'   , 'b':'-...' , 'c':'-.-.' , 'd':'-..'  , 'e':'.'    ,
'f':'..-.' , 'g':'--.'  , 'h':'....' , 'i':'..'   , 'j':'.---' ,
'k':'-.-'  , 'l':'.-..' , 'm':'--'   , 'n':'-.'   , 'o':'---'  ,
'p':'.--.' , 'q':'--.-' , 'r':'.-.'  , 's':'...'  , 't':'-'    ,
'u':'..-'  , 'v':'...-' , 'w':'.--'  , 'x':'-..-' , 'y':'-.--' ,
'z':'--..' , '1':'.----', '2':'..---', '3':'...--', '4':'....-',
'5':'.....', '6':'-....', '7':'--...', '8':'---..', '9':'----.',
'0':'-----', '-':'-....-', '?':'..--..', ',':'--..--', ':':'---...',
'=':'-...-'}

def dot():
    pitch(1000, dt)
    sleep(dt)

def dash():
    pitch(1000, 3 * dt)
    sleep(dt)

def transmit(text):
    for c in text:
        if c == " ":
            sleep(4 * dt)
        else:
            c = c.lower()
            if c in morse:
                k = morse[c]
                for x in k:
                    if x == '.':
                        dot()
                    else:
                        dash()
            sleep(2 * dt)

transmit("komme morgen abend")
```

ZUM SELBST LÖSEN

5. Schreibe die folgende Melodie in der musikalischen Notation und spiele sie ab.



6. Schreibe ein Programm, welches eine Geheimnachricht im Morsecode so langsam abspielt, dass dein(e) Kamerad(in) sie entziffern kann.

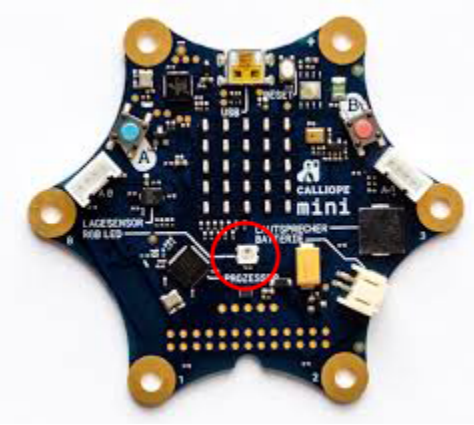
6. Farb-LED

DU LERNST HIER...

wie Farben aus einer roten, einer grünen und einer blauen Komponente zusammengesetzt werden und du die Farb-LED mit der gewünschten Farbe leuchten lässt.

RGB-LED

Um Farbbilder mit dem Computer zu speichern und zu verarbeiten, müssen die Farben als Zahlen definiert werden. Es gibt dazu mehrere Möglichkeiten. Die bekannteste ist das RGB-Farbmodell, wo die Intensitäten der drei Farbkomponenten Rot, Grün und Blau als Zahlen zwischen 0 und 255 angegeben werden.
Mit der Farb-LED des Calliope kannst du mit den Farben im RGB-Modell experimentieren.



Der Befehl
set_colors(r, g, b)
wo r, g, und b Zahlen zwischen 0 und 255 sind, legt die rote, grüne und blaue Farbkomponente fest.

Der **Calliope_mini 3** verfügt über drei RGB-LEDs. Diese werden als Neopixel angesprochen:

```
LEDs = NeoPixel(pin_RGB, 3)
LEDs[0] = (r, g ,b)
LEDs[1] = (r, g ,b)
LEDs[1] = (r, g ,b)
LEDs.show()
```

MUSTERBEISPIELE

Farb-LED in verschiedenen Farben leuchten lassen

Die Farb-LED leuchtet der Reihe nach in den Farben rot, grün, blau, gelb, magenta und cyan jeweils während 1000 Millisekunden und schaltet dann ab.

```
from calliope_mini import *

led.set_colors(255, 0, 0)
sleep(1000)
```

```
led.set_colors(0, 255, 0)
sleep(1000)
led.set_colors(0, 0, 255)
sleep(1000)
led.set_colors(255, 255, 0)
sleep(1000)
led.set_colors(255, 0, 255)
sleep(1000)
led.set_colors(0, 255, 255)
sleep(1000)
led.set_colors(0, 0, 0)
```

Für Calliope_mini 3 musst du das Programm entspechen anpassen. Hier leuchtet das erste LED rot, das zweite grün und das dritte hellblau.

```
from calliopemini import *
from neopixel import *

LEDs = NeoPixel(pin_RGB, 3)
LEDs[0] = (255,0,0)
LEDs[1] = (0,255,0)
LEDs[2] = (0,255,255)
LEDs.show()
```

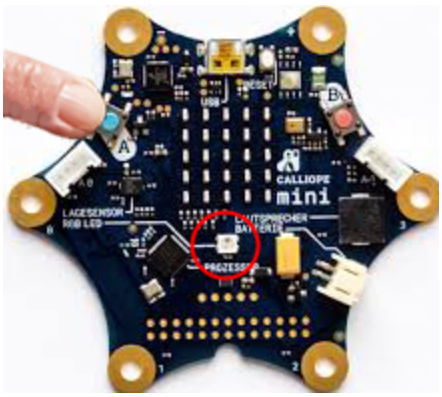
Wenn für die Farbkomponente die maximal mögliche Zahl 255 gewählt wird, leuchten die LEDs sehr hell. Angenehmer ist es, wenn du für r, g, b kleinere Zahlen wählst:

```
LEDs[0] = (100,0,0)
LEDs[1] = (0,100,0)
LEDs[2] = (0,100,100)
```

Farb-LED in zufällig gewählten Farben leuchten lassen

Hier soll die Farb-LED in zufällig gewählten Farben jeweils 500 ms lang leuchten. Die Farbkomponenten r, g, b bestimmst du in der Funktion randomColor() mit drei Zufallszahlen im Bereich 0 ... ... 100.

Durch Drücken des Buttons A beendest du das Farbspiel.



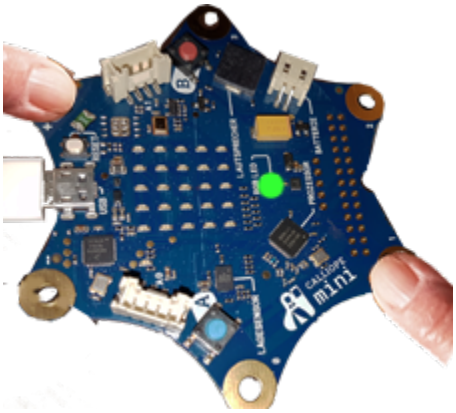

```
from calliope_mini import *
from random import randint

def randomColor():
    r = randint(0, 100)
    g = randint(0, 100)
    b = randint(0, 100)
    led.set_colors(r, g, b)
```

```
while not button_a.was_pressed():
     randomColor()
     sleep(500)
led.set_color(0, 0, 0)
```

Farb-LED für eine Spannungsanzeige verwenden
Der Calliope kann auch für physikalische Experimente verwendet werden, beispielsweise um Spannungen eines Sensors im Bereich 0...3.3V zu messen.

Berührst du gleichzeitig den Anschluss + (3.3V) und den Pin 1, so ergibt sich ein Stromkreis und die Spannung am Pin 1 steigt an.
Der Befehl pin1.read_analog() gibt die Spannung als Zahl zwischen 0 und 1023 zurück.

In deinem Programm leuchtet die LED ohne Stromkreis blau, beim Berühren des Pin 1 grün und beim Berühren des Pin 2 rot.



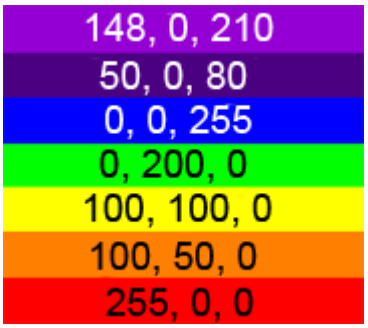
```
from calliope_mini import *

while True:
    if pin1.read_analog() > 400:
        led.set_colors(0, 255, 0)
    elif pin2.read_analog() > 400:
        led.set_colors(255, 0, 0)
    else:
        led.set_colors(0, 0, 255)
    sleep(100)
```

MERKE DIR...

Die Farbe der Farb-LED legst du durch die rote, grüne und blaue Farbkomponente (eine Zahl zwischen 0 und 255) fest.

ZUM SELBST LÖSEN

1. Schreibe ein Programm, mit dem die Farb-LED nacheinander in Regenbogenfarben leuchtet. Wiederhole diese Farbfolge 3 mal. Verwende die nebenstehende Zuordnung der Farben.



2. Wenn du den Button A drückst, soll die Farb-LED in der Farbe magenta, beim Drücken des Buttons B mit der Farbe *cyan* leuchten. Damit die LED nicht zu hell leuchtet, wählst du für die Farbkomponenten höchsten einen Wert von 100.

3. Wenn du mit den Fingerspitzen Pin + und Pin 1 berührst, blinkt die Farb-LED rot, wenn du Pin + und Pin 2 berührst, so blinkt sie blau, bei der Berührung des Pin + und Pin 3 blinkt sie grün.

4. In einer endlosen while-Schleife soll die Farb-LED mit weisser Farbe das Signal SOS senden, wobei sie bei S dreimal kurz (200 ms) und bei O dreimal lange (600 ms) leuchten soll. Nach jedem SOS Signal folgt eine Pause.

7. BLUETOOTH-KOMMUNIKATION

DU LERNST HIER...

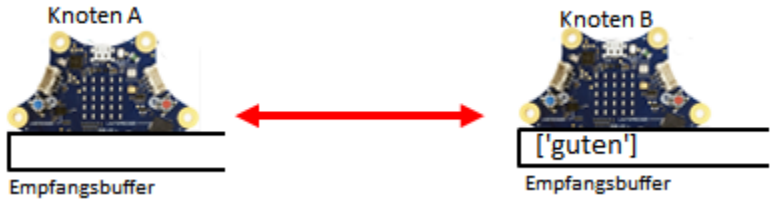
wie zwei Computer über Bluetooth miteinander Informationen austauschen.

DRAHTLOSE KOMMUNIKATION

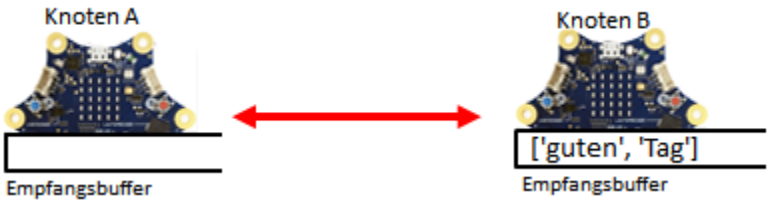
Bei der Kommunikation zwischen zwei Calliope (auch Knoten genannt) ist jeder der Knoten gleichzeitig ein Sender und ein Empfänger. Zuerst schalten beide Knoten mit radio.on() das Sende-/Empfangsgerät ein. Dabei wird eine drahtlose Bluetooth-Verbindung erstellt. Nachfolgend kann jeder der Knoten mit radio.send(msg) eine Message (als String) an den anderen Knoten senden, die dort zuerst in einen Empfangsbuffer wie in einer Warteschlange gespeichert wird. Mit radio.receive() holst du die "älteste" Message, die dann im Buffer gelöscht wird.

- Knoten A und Knoten B haben *radio.on()* aufgerufen:

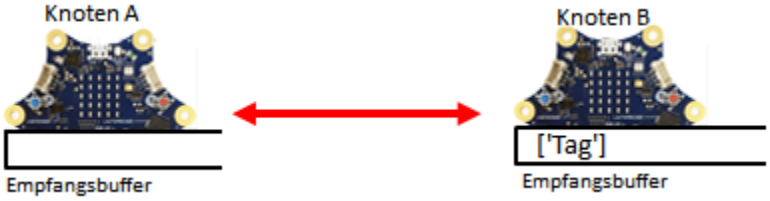
- Knoten A hat *send('guten')* aufgerufen:



- Knoten A hat *send('Tag')* aufgerufen:

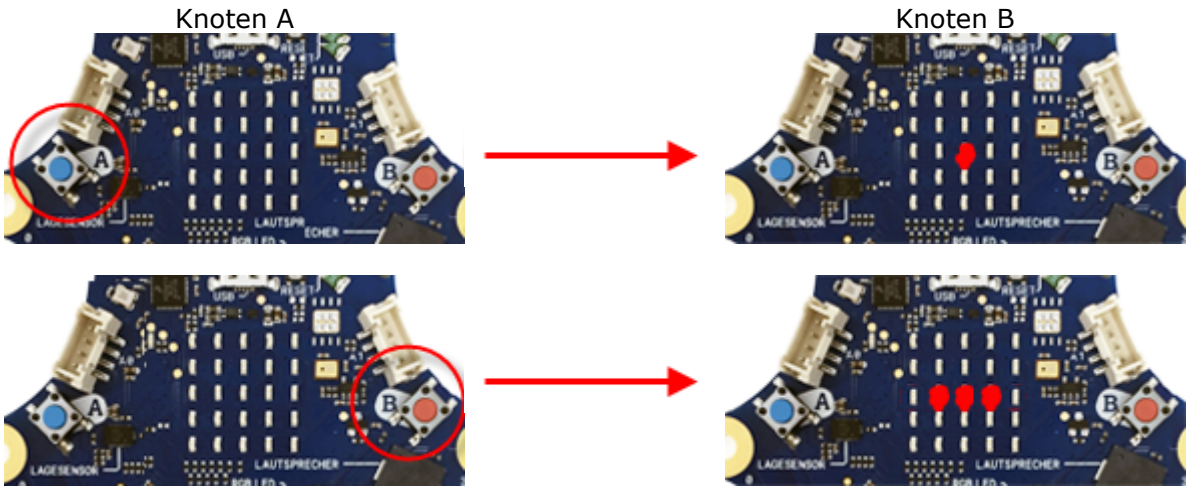


- Knoten B hat *msg = receive()* aufgerufen. msg enthält 'guten':



MUSTERBEISPIELE

Du erstellst mit einer anderen Person eine drahtlose Kommunikation, um Informationen mit Punkten und Strichen (wie im Morsecode) auszutauschen. Wenn man auf dem einen Calliope den Button A klickt, so leuchtet beim anderen die mittlere LED des Displays während ½ s auf (wie ein Punkt), klickt man auf Button B so leuchten drei LEDs auf (wie ein Strich).



Der zentrale Teil des Programms besteht aus einer Endlosschleife, in der zuerst das Senden und dann das Empfangen geregelt wird. Wenn A geklickt wird, überträgst du die Message "p", (kurz für "point"), wenn B geklickt wird, überträgst du "d" (kurz für "dash").

Nachfolgend holst du mit *receive()* den ältesten Eintrag in der Warteschlage. (Falls diese leer ist, kriegst du den speziellen Wert *None*.) Je nachdem, ob du "p" oder "d" erhalten hast, führst du die entsprechende Aktion aus.

```
import radio
from calliope_mini import *

point = Image('00000:00000:00900:00000:00000')
dash = Image('00000:00000:09990:00000:00000')

def draw(img):
    display.show(img)
    sleep(500)
    display.clear()
    sleep(500)

radio.on()
while True:
    if button_a.was_pressed():
        radio.send("p")
    elif button_b.was_pressed():
        radio.send("d")
    rec = radio.receive()
    if rec != None:
        if rec == "p":
            draw(point)
        elif rec == "d":
            draw(dash)
    sleep(10)
```

► In Zwischenablage kopieren

Echte Morsestationen arbeiten fast immer mit kurzen und langen Tönen. Dein Programm soll beim Empfänger den Ton einschalten, solange der Button A gedrückt ist. Dazu sendest du dem Empfänger die Message "d" (für "down"), wenn der Button gedrückt wird. Wenn du den Button loslässt, so sendest du "u" (für "up"). Dazu musst du eine boolesche Variable *down* einführen, mit der du dir merkst, ob der Button gedrückt oder losgelassen ist.

```
import radio
from calliope_mini import *
import music

radio.on()
down = False
while True:
    if button_a.is_pressed() and not down:
        down = True
        radio.send("d")
    elif not button_a.is_pressed() and down:
        down = False
        radio.send("u")
    rec = radio.receive()
    if rec != None:
        if rec == "d":
            music.pitch(500, 1000000, wait = False)
        elif rec == "u":
            music.stop()
    sleep(10)
```

► In Zwischenablage kopieren

MERKE DIR...

dass der Sender mit *send(msg)* eine Message sendet, die dann im Empfangsbuffer der Reihe nach gespeichert wird, bis der Empfänger die "älteste" mit *msg = receive()* im Buffer abholt.

ZUM SELBST LÖSEN

1. Orientiere dich über den Morsecode und versuche mit dem Programm im ersten Musterbeispiel einige kurzen Meldungen auszutauschen.

2. a) Schreibe ein Programm, das die aktuelle Neigung (in einer Richtung) des Calliope einem Empfängerknoten mitteilt, der den Wert im Terminal ausschreibt.

   b) Schreibe den Wert beim Empfänger auf dem Display aus (skaliert auf eine Zahl zwischen 0 und 9)

8. MAGNETFELD- UND GYROSENSOR

DU LERNST HIER...

wie du mit dem Calliope das Magnetfeld bestimmen kannst und wie ein Gyrosensor funktioniert.

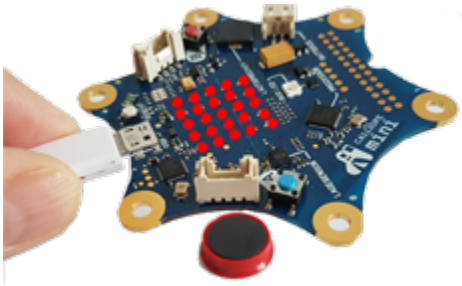
SENSORWERTE ANZEIGEN

Der Magnetfeld- und der Gyrosensor sind hardwaremässig im Lagesensor-Chip integriert. Der Magnetfeldsensor gibt die aktuellen x-, y-, und z-Komponenten der Magnetfeldes an der Stelle des Sensors zurück. Mit get_x(), get_y() und get_z() kannst du die einzelnen Komponenten abfragen. Die Werte liegen zwischen -32768 und 32767.

```
from calliope_mini import *

while True:
    bx = magnetometer.get_x()
    by = magnetometer.get_y()
    bz = magnetometer.get_z()
    print(bx, by, bz)
    sleep(500)
```

MUSTERBEISPIELE

Minen suchen
Lege einen kleinen Magneten, wie du ihn für Memoboards finden kannst, unter einen Kartondeckel. Den Magneten kannst du wie eine Mine auffassen, die man mit dem Calliope als Minensucher auffinden muss. In deinem Programm leuchten alle LEDs umso heller, je näher das Board zur Mine kommt.



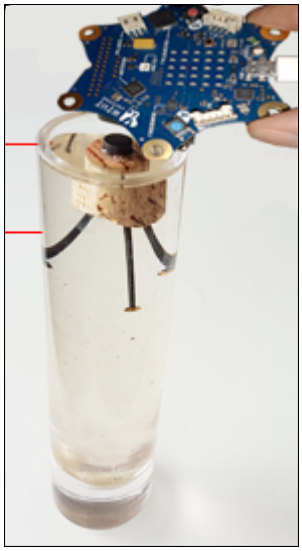
Du verwendest den Befehl *magnetometer.get_z(),* der die Stärke des Magnetfeldes in der Vertikalen zurückgibt. Dann skalierst ihn so, dass du einen Helligkeitswert zwischen 0 und 9 erhältst. In der Funktion light(v) schaltest du alle LEDs miteinander an.

```
from calliope_mini import *

def light(n):
    for i in range(5):
        for k in range(5):
            display.set_pixel(i, k, n)

while True:
    bz = magnetometer.get_z()
    v = abs(bz // 3300)
```

```
    n = min(9, v)
    light(n)
    sleep(20)
```

Wasserstand messen

Deine Testanlage besteht aus einer Blumenvase und einem Magneten, der auf einem schwimmenden Korkzapfen befestigt ist. Mit drei Nägeln kannst du den Zapfen so stabilisieren, dass sich der Magnet bei änderndem Wasserstand immer in der gleichen vertikalen Linie bewegt.



Aus der Messung der z-Komponente des Magnetfeldes erhältst du eine Information über die Höhe der Wassersäule. Du teilst sie in 3 Wasserstandsbereiche "zu hoch", "ok" und "lzu tief" .

Am besten stellst du dir vor, dass sich das Gefäss in drei "Zuständen" befindet, die du mit der Variablen *state* erfasst, welche die Werte "high", "ok" und "low" annehmen kann. Aus dem skalierten Wert des Magnetfelds triffst du den Entscheid, ob sich der Zustand ändert.

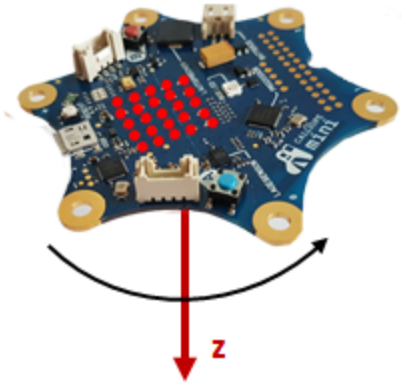
Mit Sound kannst du auch einen akustischen Alarm realisieren.

```
from calliope_mini import *
import music

state = "ok"

while True:
    z = magnetometer.get_z()
    #print(z)
    if z >= 700 and state != "high":
        state = "high"
        print("->high")
    elif z >= 500 and z < 700 and state != "ok":
        state = "ok"
        print("->ok")
    elif z < 500 and state != "low":
        state = "low"
        print("->low")
    if state == "high":
        music.pitch(2000, 100)
    elif state == "low":
        music.pitch(500, 100)
    sleep(100)
```

Beachte, dass die z-Komponente des Magnetfeldes abhängig vom Magnet, den du in diesem Experiment verwendest. Aktiviere zu Beginn die Zeile *print(z)* und ändere, falls nötig, die Grenwerte.

Gyrometer verwenden

Der Gyrosensor misst die Rotationsgeschwindigkeit (Winkel pro Zeiteinheit, auch Winkelgeschwindigkeit genannt), im Gegensatz zum Lagesensor bleiben die Werte in der Nähe von Null, wenn sich der Calliope nicht mehr bewegt. Mit *gyrometer.get_x()*, *gyrometer.get_y()* und *gyrometer.get_z()* erhält man Werte im Bereich -32768 und 32767. Analog wie beim Minensuchen, werden alle LEDs heller, wenn die Rotationsgeschwindigkeit um die z-Achse grösser wird.




```
from calliope_mini import *

def light(n):
    for i in range(5):
        for k in range(5):
            display.set_pixel(i, k, n)

while True:
    bz = gyrometer.get_z()
    v = abs(bz // 3300)
    v = min(9, v)
    light(v)
    sleep(20)
```

MERKE DIR...

Der Magnetfeldsensor kann zum berührungslosen Erfassen der Position eines Magneten oder einer Magnetspule verwendet werden. Mit dem Gyrosensor wird die Rotationsgeschwindigkeit bestimmt.

ZUM SELBST LÖSEN

1. Berührungslose Umdrehungszähler (z.B. bei Rädern) verwenden oft einen Magneten, der bei jeder Umdrehung einmal vor einem Magnetfeldsensor vorbei läuft. Befestige ein Magnetchen (z.B. von einem Memoboards oder einem Magnetschnäpper) an einem Velorad und schreibe ein Programm, dass im Terminalfenster die Anzahl Umdrehungen ausschreibt. Wie lässt sich damit ein Kilometerzähler bauen?

9. ALARMANLAGEN

DU LERNST HIER...

wie du den Calliope einsetzen kannst, um Systeme zu überwachen und gegebenenfalls Alarm auszulösen.

WASSERSTAND ÜBERWACHEN

Deine kleine Pflanze musst du selten giessen, aber ganz vergessen darfst du es nicht. Mit deinem Calliope kannst du den Wasserstand kontrollieren und wenn er zu niedrig ist, Alarm auslösen.

Du verwendest dabei die Eigenschaft von Wasser, Strom zu leiten, Wenn du also von der 3V (+) Spannungsversorgung einen Stromkreis zum Pin1 erstellst, der durch das Wasser führt, so wird am Pin1 eine höhere Spannung zu messen sein, als wenn der Stromkreis offen ist. Du brauchst für den Aufbau nur zwei Kabel mit Krokodilklemmen und zwei Draht- oder sonstige Metallstücke, die du als Sonden in den Blumentopf steckst.

Das eine Kabel verbindest du mit Pin1, das andere mit 3V.

Für die Programmentwicklung kannst du einen beliebigen Wasserbehälter verwenden.

Mit dem Befehl *pin1.read_analog()*

misst du die Spannung an P1 und schreibst den Werte ins Terminalfenster. Wie du siehst, ist der Unterschied der Messwerte gross, je nachdem, ob die Sonden im Wasser sind oder nicht.

```
from calliope_mini import *

while True:
    v = pin1.read_analog()
    print(v)
    sleep(500)
```

Als Eigenarbeit ergänzst du die Alarmanalage mit einer optischen oder akustischen Warnung.

MERKE DIR...

Einfache Alarmanlagen kannst du leicht mit dem Calliope als eingebettetes System aufbauen, wenn du deine handwerklichen Fähigkeiten mit etwas Elektronik- und Programmierkenntnissen kombinierst.

ZUM SELBST LÖSEN

1. Ergänze deine Anlage mit einer optischen Alarmanzeige auf dem LED-Display. Beim normalen Wasserstand soll das Bild *Image.HAPPY*, beim niedrigen Wasserstand soll das Bild *Image.SAD* eingezeigt werden.

2. Ergänze die Alarmanlage mit einem akustischen Signal, der mit dem eingebauten Lautsprecher beim niedrigen Wasserstand ausgelöst wird.

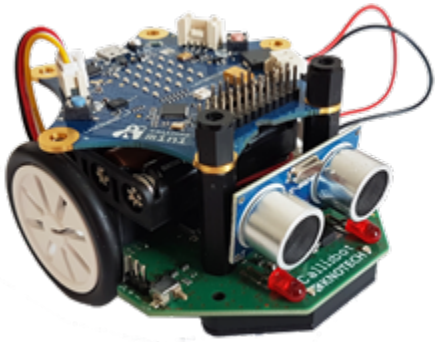
3. Baue eine Alarmanlage auf, welche überwacht, ob eine Tür geöffnet wird. Du kannst dabei den Lagesensor verwenden.

10. FAHRENDE ROBOTER

DU LERNST HIER...

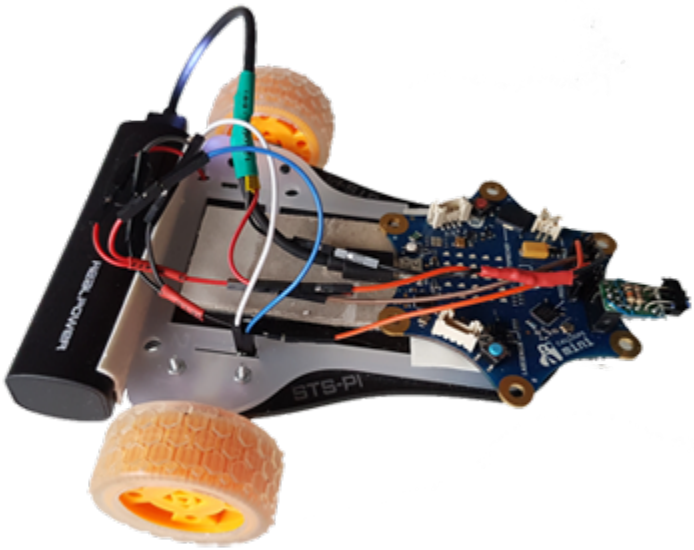
wie du einen fahrenden Roboter zusammenbauen und programmieren kannst.

ROBOTER "CALLI:BOT"



Am einfachsten kannst du einen fahrenden Roboter mit dem Bausatz von www.knotech.de zusammenbauen. Der Bausatz enthält alle notwendigen Komponenten und der Zusammenbau beansprucht nur wenige Minuten. Anwendungsbeispiele und Aufgaben für diesen Roboter findest du unter dem Menüpunkt Callibot.

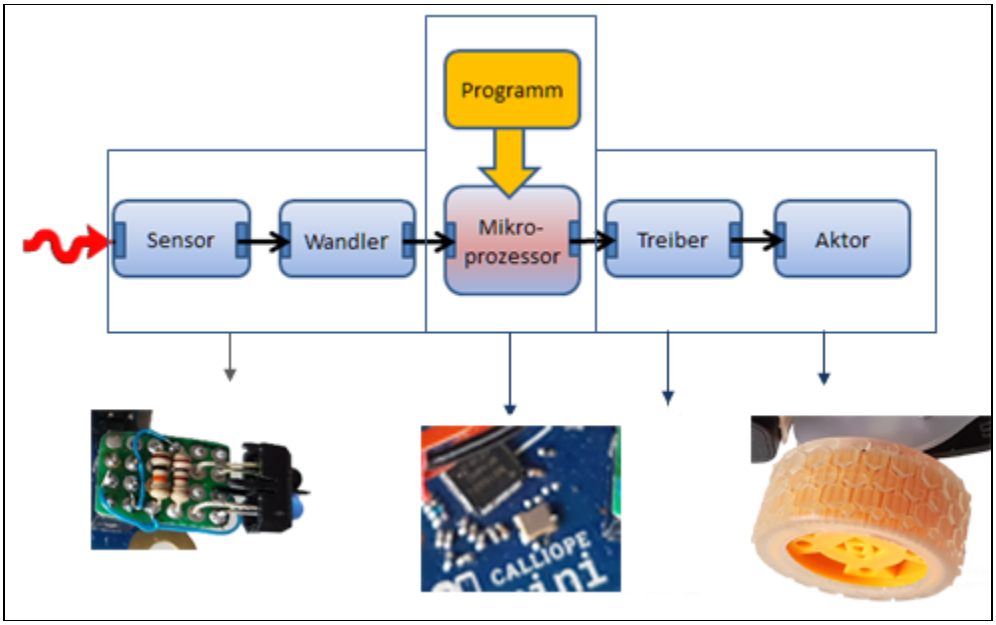
ROBOTER "CALLIOPY"



Falls du an der Elektronik interessiert bist, kannst du mit wenig Aufwand selbst einen fahrenden Roboter mit Motoren und Lichtsensoren zusammenbauen.

Die wichtigste Komponente des Roboters ist der Calliope. Dieser übernimmt die Funktion eines Gehirns, das auf Grund der Sensordaten, z.B. Hell-Dunkel-Werte von Lichtsensoren, Aktoren (Motoren, Display) betätigt.

Das folgende Blockbild für eine automatisierte Maschine ist darum sehr allgemein gültig und du erkennst diese Komponenten leicht bei deinem Calliopy-Roboter.

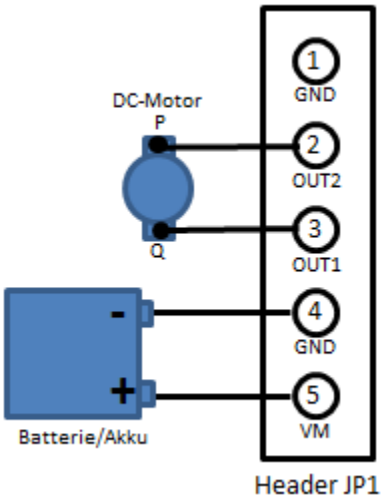


Da der Calliope in der Maschine integriert ist, nennt man sie auch ein "eingebettetes System" (embedded system).

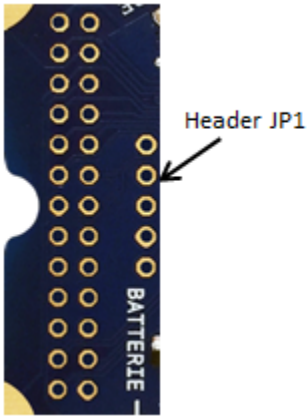
ANSTEUERUNG VON DC-MOTOREN

Ein DC-Motor (DC = Direct Current = Gleichstrom) besitzt zwei Anschlüsse P und Q. Legt man eine Gleichspannung mit dem Pluspol an P und dem Minuspol an Q, so dreht er in der einen Richtung, und zwar umso schneller, je höher die Spannung ist. Kehrt man die Polarität um, so dreht er in der anderen Richtung.

Zur Ansteuerung eines Motors (oder anderer Verbraucher mit bis zu maximal 1 A Strom) hat der Calliope einen Stromverstärker (H-Brücke DRV8887), dessen zwei Ausgänge mit Pin 2 und Pin 3 des Headers JP1 verbunden sind. Pin 5 ist der Eingang einer externen Spannung bis maximal 10 V zur Versorgung der des Motors und Pin 1 und Pin 4 sind Masseanschlüsse (GND = Ground). Einen einzelnen Motor hängst du wie folgt an:

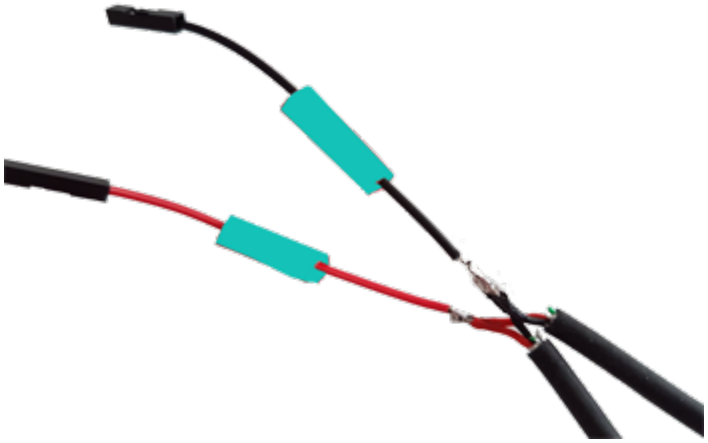


Header JP1




Eingelötete Stiftleisten

Als Batterie/Akku kannst du eine PowerBank verwenden, die eine Spannung von 5V abgibt. Dazu ein kurzes USB-Kabel auftrennen und die 5V separat herausführen.



Das so prepariertes Micro-USB-Kobel kanst du auch für die Stromversorgung des Calliope verwenden.

(Falls du einen ganz kleinen Motor hast, der bei 3.3V nicht mehr als 20 - 30 mA zieht, kann die externe Spannungsquelle entfallen, da der Calliope dann automatisch die interne Spannungsversorgung verwendet.)

Der Stromverstärker wird mit den Prozessorpins 29 und 30 angesteuert, wobei du mit dem Pin 28 den Verstärker einschaltest. Dies ergibt folgende Schaltungslogik für die 0/1-Pegel der Prozessorpins.

Pin 28	Pin 29	Pin 30	P	Q	Motor
0	x	x	kein Strom	kein Strom	Stop
1	1	0	VM	GND	Vorwärts
1	0	1	GND	VM	Rückwärts

x: 0 oder 1 hat keine Auswirkung, vorwärts und rückwärts hängen vom Einbau des Motors ab.

Mit dem folgenden Programm dreht der Motor je 5 Sekunden vorwärts und rückwärts bevor er stoppt.

Programm: [► Online-Editor]

```
from calliope_mini import *

def move():
    pin28.write_digital(1)
    pin29.write_digital(1)
    pin30.write_digital(0)

def rewind():
    pin28.write_digital(1)
    pin29.write_digital(0)
    pin30.write_digital(1)

def stop():
    pin28.write_digital(0)

move()
sleep(5000)
```

```
rewind()
sleep(5000)
stop()
```

Es ist erstaunlich, dass du mit dieser Schaltung sogar die Rotationsgeschwindigkeit des Motors wählen kannst. Wenn du nämlich den Pin 28 nur während einer bestimmten Zeit einschaltest und dann wieder ausschaltest, so dreht der Motor langsamer. Dazu verwendest du den Befehl pin28.write_analog(v) mit einem Wert v = 0..1023.

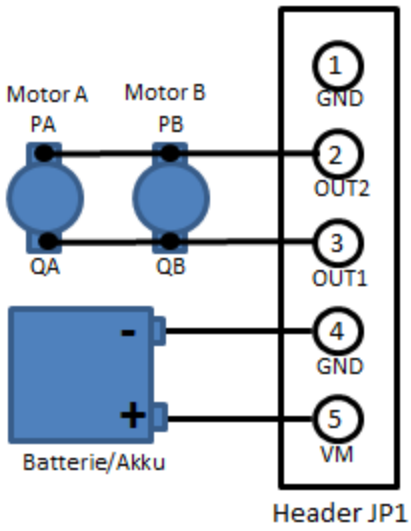
Im folgenden Programm dreht der Motor mit wählbarer Geschwindigkeit v (im Bereich 0.. 1023).

Programm: [► Online-Editor]

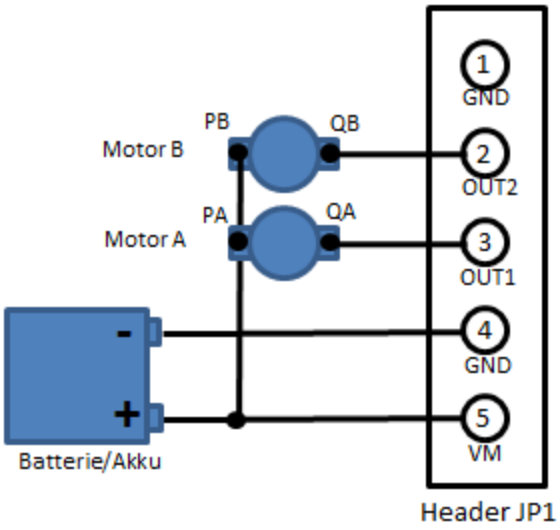
```
from calliope_mini import *

def move():
    pin28.write_analog(v)
    pin29.write_digital(1)
    pin30.write_digital(0)

def revind():
    pin28.write_analog(v)
    pin29.write_digital(0)
    pin30.write_digital(1)

def stop():
    pin28.write_digital(0)

v = 500
move()
sleep(5000)
rewind()
sleep(5000)
stop()
```

VERWENDUNG VON DC-MOTOREN

Leider besitzt der Calliope nur eine einzige H-Brücke. Willst du zwei Motoren verwenden, wie dies bei einem üblichen Rover der Fall ist, so musst du Kompromisse eingehen. Du kannst die Motoren entweder gleichzeitig vorwärts bzw. rückwärts laufen lassen, indem du sie parallel schaltest:



Wenn du den einen Motor zwischen OUT1 und VM und den anderen zwischen OUT2 und VM hängst, kannst du die beiden Motoren unabhängig voneinander vorwärts schalten. Wenn der Pegel von OUT1 auf GND liegt, läuft Motor A vorwärts, wenn OUT2 auf GND liegt, läuft Motor B vorwärts.



Es ergibt sich also folgende Schaltungslogik:

Pin 28	Pin 29	Pin 30	PA	PB	Motor A	Motor B
0	x	x	kein Strom	kein Strom	Stop	Stop
1	1	0	VM	GND	Stop	Vorwärts
1	0	1	GND	VM	Vorwärts	Stop
1	1	1	GND	GND	Vorwärts	Vorwärts

Sind die beiden Motoren in einem 2 rädrigen Rover eingebaut, so bewegt sich dieser mit folgendem Programm zuerst vorwärts, dann dreht er auf die eine Seite und dann auf die andere Seite. Schliesslich bewegt er sich wieder vorwärts und stoppt.

Programm: [► Online-Editor]

```
from calliope_mini import *

def forward():
    pin28.write_analog(v)
```

```
    pin29.write_digital(1)
    pin30.write_digital(1)

def left():
    pin28.write_analog(v)
    pin29.write_digital(0)
    pin30.write_digital(1)

def right():
    pin28.write_analog(v)
    pin29.write_digital(1)
    pin30.write_digital(0)

def stop():
    pin28.write_digital(0)

v = 500
forward()
sleep(3000)
left()
sleep(1000)
right()
sleep(1000)
forward()
sleep(3000)
stop()
```

Um die Programme zu vereinfachen, enthält die Python-Distribution von Calliope das Modul *cprover* mit den oben definierten Funktionen *forward(), right(), left(), stop()* und *setSpeed(v)* (v im Bereich 0..100).

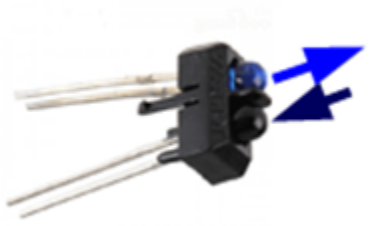
MERKE DIR...

DC-Motoren drehen je nach Polarität der Spannung vor- oder rückwärts. Der Calliope hat einen Stromverstärker, der für den Betrieb eines einzelnen Motors vorgesehen ist. Werden zwei Motoren benötigt, so kann man diese entweder parallel vorwärts und rückwärts schalten oder man kann sie einzeln vorwärts, aber nicht rückwärts schalten.

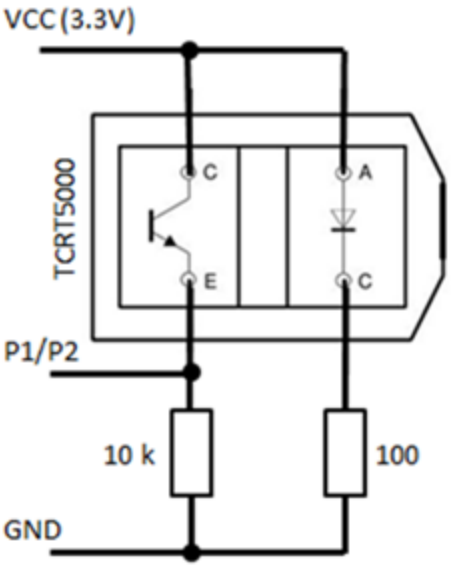
ZUSATZSTOFF

DISTANZ MESSEN MIT INFRAROT-SENSOR

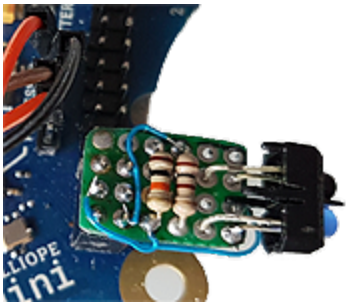
Der Infrarotsensor TCRT5000 enthält eine Fotodiode (IR-LED), die Infrarotlicht (mit einer Wellenlänge von von ungefähr 1000 nm) erzeugt und einer Fotodiode, welche die Intensität des reflektierenden Lichtes messen kann. Es gibt mehrere Bezugsquellen (z.B. 4tronix oder Ebay).



Durch diese Konstruktion können Infrarotsensoren die Änderungen im näheren Sichtfeld registrieren und werden in der Praxis häufig als Bewegungsmelder eingesetzt.



In der Schaltung benötigst du zwei Widerstände von 10 kOhm und 100 Ohm, die du beispielsweise auf einer kleinen Lochplatte aufbaust. Den Ausgang E des Fototransistors verbindest du mit einem der Analog-Eingangspins P1 oder P2 des Calliope. Du kannst also für die Detektion einer Streifenbahn zwei nach unten zeigende Sensoren verwenden.



MUSTERBEISPIEL

Der Calliopy soll mit seinem Infrarotsensor ein Hindernis detektieren und darauf reagieren. In deinem Beispiel wird der Roboter mit *run()* in Vorwärtszustand versetzt. Dabei wird in einer endlosen Wiederholungsschleife mit *pin1.red_analog()* der Sensorwert ständig abgefragt. Wenn der Sensorwert grösser als 100 ist , fährt fährt der Roboter 3000 Millisekunden rückwärts und danach erneut vorwärts. Der Sensor misst die Intensität des reflektierenden Lichtes, dh. je näher ein Hinderniss ist, umso grösser ist der Sensorwert. Helle Gegenstände reflektieren stärker. Du musst also den Schwellenwert an deine Situation anpassen.

Unter der Verwendung des Moduls *cprover* ist das Programm sehr einfach:

Programm: [► Online-Editor]

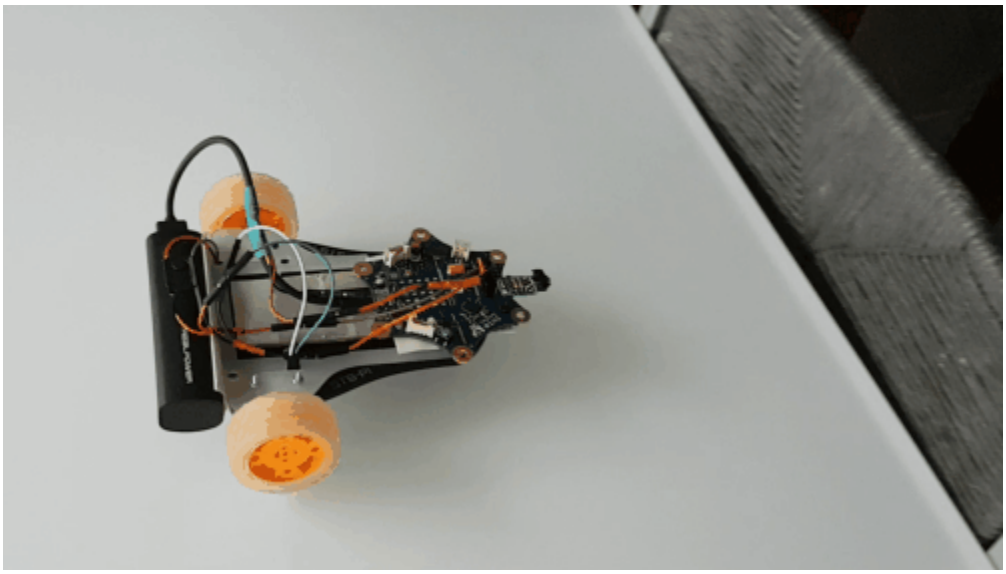
```
from calliope_mini import *
from cprover import *

run()
while True:
    v = pin1.read_analog()
    if v > 100:
        back()
        sleep(3000)
        run()
```

Das Modul *cprover.py* kannst du auf deinen Calliope herunterladen und auch in weiteren Anwendungen mit einem fahrenden Roboter verwenden. Dabei gehst du wie folgt vor: du nimmst das Programm wie üblich in den TigerJython-Editor und wählst "Tools/Modul hinunterladen".

Programm: [► Online-Editor]

```
# cprover.py

from calliope_mini import pin28, pin29, pin30
```

```
def forward():
    pin28.write_analog(v)
    pin29.write_digital(1)
    pin30.write_digital(1)

def left():
    pin28.write_analog(v)
    pin29.write_digital(0)
    pin30.write_digital(1)

def right():
    pin28.write_analog(v)
    pin29.write_digital(1)
    pin30.write_digital(0)

def stop():
    pin28.write_digital(0)

def move():
    left()

def rewind():
    right()

def setSpeed(speed):
    global v
    v = int(speed / 100 * 1023)

setSpeed(100)
```



ZUM SELBST LÖSEN

1. Baue einen fahrenden Roboter ähnlich dem oben gezeigten Calliopy mit zwei Motoren und einer Rollkugel als Lenkrad auf. (Bezugsquelle für das Roboter-Chassis STS-PI: www.pimoroni.com) Als Spannungsversorgung für den Calliope und die Motoren kannst du einen PowerBank verwenden. Die 5V für die Motoren kriegst du durch Auftrennen des

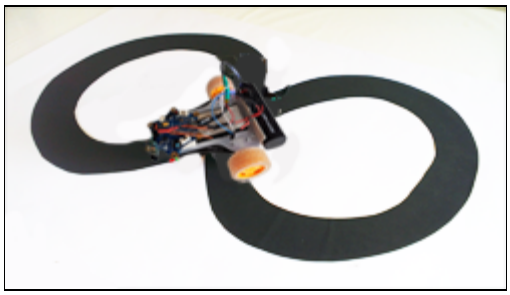
USB-Kabels und herausführen von GND (normalerweise schwarze Litze) und 5V (normalerweise rote Litze).

Hinweis: Viele PowerBanks schalten bei kleinen Belastungen noch einer bestimmten Zeit ab. Bei der Verwendung von IR-Distanzsensoren ist die Belastung normalerweise gross genug. Sonst kann ein zusätzlicher Belastungswiderstand von 100 Ohm zwischen 5V und GND das Abschalten verhindern.

2. Der Roboter kann mit seinem Infrarotsensor auch helle und dunkle Gegenstände unterscheiden. Montiere den Infrarotsensor so, dass er nach unten zeigt und damit einen hellen oder dunklen Untergrund unterscheiden kann.
Baue eine Schwarz-Weise Bahn und schreibe ein Programm, damit der Calliopy der Kante entlang fährt.

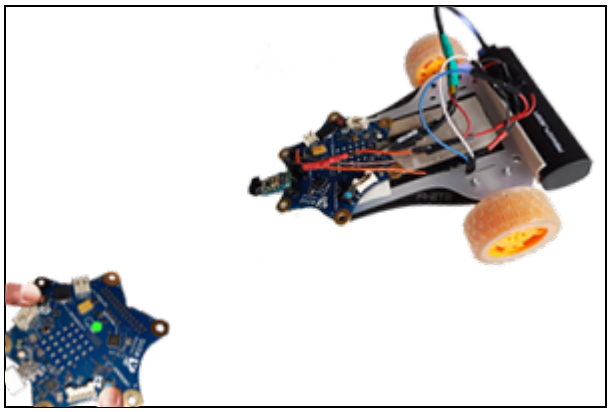
   Verwende die Befehle *leftArc(r)* bzw. *rightArc(r)*, um den Roboboter zu steuern.

3. Montiere zwei zwei Infrarotsensoren, die nach unten zeigen. Jetzt kann der Roboter auf einem beliebigen Streifen fahren. bei guter Programmierung sogar auf einer Bahn in Form einer Acht.



4. **Fernsteuerung mit zweitem Calliope**
Baue einen Roboter, der vorwärts, links und rechts fahren kann. Der Roboter soll mit einem zweiten Caliope (oder mirco:bit) mit folgenden Befehlen ferngesteuert werden.

   - rechter Buton: fahre rechts
   - linker Button: fahre links
   - beide Buttons: fahre vorwärts
   - kein Button: stop



Optimiere die unterstehenden Programme für deinen Calliopy-Rover:

Programm für den Roboter:

Programm: [► Online-Editor]

```
from calliope_mini import *
from cprover import *
import radio

radio.on()

display.show(Image.YES)
while True:
    rec = radio.receive()
    if rec != None:
```

```
        display.show(rec)
    if rec == "F":
        forward()
    elif rec == "L":
        left()
    elif rec == "R":
        right()
    elif rec == "S":
        stop()
    sleep(10)
```

Programm für den zweiten Calliope (Steuerung):

Programm: [▶ Online-Editor]

```
import radio
from calliope_mini import *

radio.on()
while True:
    if button_a.is_pressed() and button_b.is_pressed():
        state = "F"
    elif button_a.is_pressed() and not button_b.is_pressed():
        state = "L"
    elif not button_a.is_pressed() and button_b.is_pressed():
        state = "R"
    elif not button_a.is_pressed() and not button_b.is_pressed():
        state = "S"
    radio.send(state)
    display.show(state)
    sleep(10)
```

12. UMWELTSENSOREN

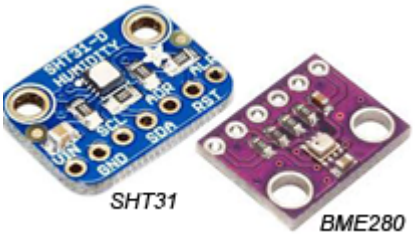
DU LERNST HIER...

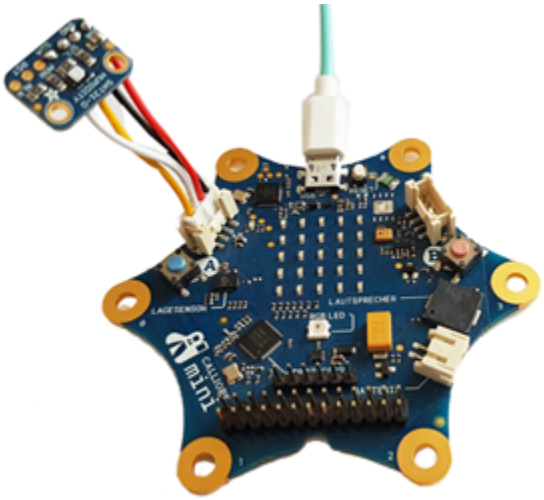
mit dem Temperatur-, Feuchtigkeit- und Luftdrucksensor die Umweltbedingungen erfassen und einfache Regelungssysteme programmieren. Interessant sind Umweltsensoren insbesondere im Zusammenhang mit den IoT (Internet of Things) -Anwendungen

HOCHPRÄZISE UMWELTSENSOREN

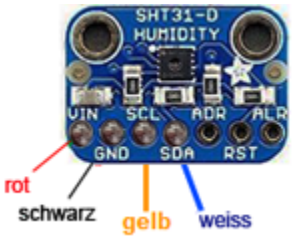
Da Calliope das I^2C-Protokoll für die Kommunikation mit den Sensoren unterstützt, können auch hochpräzise Sensoren an die Grove-Schnittstelle angeschlossen werden. Die TigerJython-Robotik-Bibliothek unterstützt

- SHT31 Temperatur- und Feuchtigkeitssensor
- BME280 Temperatur-, Feuchtigkeits- und Luftdrucksensor





Um die Sensoren am Calliope anzuschliessen, sind einige Lötarbeiten notwendig. Du schneidest ein Grove-Verbindungskabel auf, isolierst die darin enthaltenen Kabel ab und lötest das schwarze Kabel bei GND, das rote bei VCC, das gelbe bei SCL und das weisse bei SDA an.



Damit du den SHT-Sensor verwenden kannst, musst du das Modul *sht_mini* importieren. Das Modul sht_mini wird beim Flashen automatisch auf den Calliope kopiert. Das Module bme280_mini muss du vor der ersten Verwendung nachinstallieren.

MUSTERBEISPIELE

Beispiel 1: Temperatur mit SHT31-Sensor

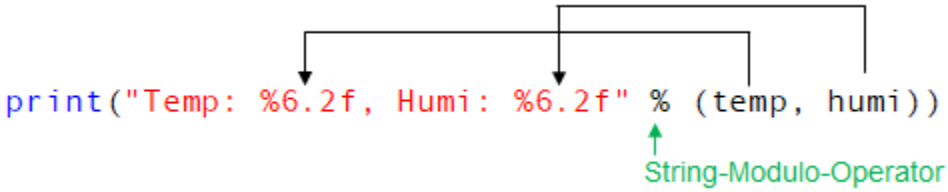
Im ersten einfachen Beispiel wird die Umgebungstemperatur im Terminalfenster angezeigt. Ist der Sensirion SHT31 - Sensor am I²C-Port 1 angeschlossen, liefert der Befehl temp, hume = sht_mini.getValues() ein Tupel mit Temperatur (Grad C) und Luftfeuchtigkeit (%). Mit print(temp) wird die aktuelle Temperatur, mit *print(humi)* die Feuchtigkeit und mit *print(temp, humi)* werden die beiden Werte im Terminalfenster angezeigt. sleep(500) legt die Messperiode von 500 ms fest. Der Sensor reagiert schnell auf die Umgebungsänderungen. Sobald du z. B. den Sensor mit den Fingern berührst, werden höhere Temperatur und Feuchtigkeitswerde angezeigt. Die Programmausführung wird mit Ctrl+C im Terminalfensters beendet.

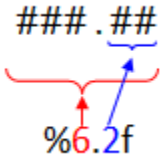
```
25.518
25.518
25.4753
25.502
25.4886
25.8999
27.5341
28.3913
29.0402
29.5182
29.9267
30.3059
```

```
from calliope_mini import *
import sht_mini

while True:
    temp, humi = sht_mini.getValues()
    print(temp)
    sleep(500)
```

Beispiel 2: Temperatur und Feuchtigkeit mit formatierter Ausgabe

Die Sensorwerte sind Dezimalzahlen (float) mit mehreren nachkommastellen. Es ist deshalb vorteilhaft, eine formatierte Ausgabe zu verwenden.





Der Formatstring enthält Platzhalter, welche für die Ausgabewerte die gesamte Anzahl Zeichen und die Anzahl Dezimalstellen angeben. Werte mit mehreren Dezimalstellen werden automatisch gerundet. Willst du nur die Temperatur mit einer Dezimalstelle anzeigen, schreibst du:
print("Temperatur: %6.2f" % temp).

Im nächsten Programm werden die Sensorwerte formatiert mit zwei Dezimalstellen angezeigt.

```
from calliope_mini import *
import sht_mini

while True:
    temp, humi = sht_mini.getValues()
    print("Temp: %6.2f, Humi: %6.2f" % (temp, humi))
    sleep(200)
```

```
Temp:  25.69, Humi:  52.30
Temp:  26.08, Humi:  53.02
Temp:  27.24, Humi:  56.04
Temp:  27.85, Humi:  60.08
Temp:  28.17, Humi:  63.63
Temp:  28.63, Humi:  69.13
Temp:  29.03, Humi:  74.31
Temp:  29.12, Humi:  75.51
Temp:  29.25, Humi:  76.53
```

Beispiel 3: Temperatur-Regelung
Wie funktioniert ein Regelungssysteme? Ein Temperatursensor misst die aktuelle Temperatur (Istwert) und liefert ihn an den Regler. Dieser vergleicht ihn mit dem gewünschten Wert (Sollwert). Ist der aktuelle Wert kleiner als Sollwert, wird die Wärmequelle eingeschaltet, ist er höher als der Sollwert, wird sie ausgeschaltet. In der Regel wird als Sollwert nicht ein einziger Wert, sondern ein Werte-Bereich festgelegt.

In deinem Beispiel programmierst du eine Temperatur-Regelung. Der EV3 ist mit einem Temperatursensor ausgerüstet. Ausserdem brauchst du eine Wärmequelle (Lampe, Föhn, Heizung...), oder kannst du den Sensor vorsichtig mit deinen Fingern berühren. So lange die aktuelle Temperatur tiefer als der Sollwert° ist, werden die roten LEDs eingeschaltet. Liegt der Temperaturwert beim Sollwert (zwischen 26 und 28°), sind die orangen LEDs eingeschaltet. Ist die Temperatur höher als der Sollwert, werden die LEDs ausgeschaltet.

```
from calliope_mini import *
import sht_mini

ts = 27 #sollWert
while True:
    temp, humi = sht_mini.getValues()
    print(temp)
    if temp < ts - 1:
        display.show(Image.SAD)
    elif temp > ts - 1 and temp < ts + 1:
        display.show(Image.HAPPY)
    elif temp > ts + 1:
        display.show(Image.NO)
    sleep(300)
```

Auf dem gleichen Prinzip werden viele Regelsysteme aufgebaut. Anstelle der LEDs kann z.B. eine Heizung ein- und ausgeschaltet werden.

MERKE DIR...

Ist der SHT31 Sensor am I^2C Port angeschlossen, liefert der Befehl *sht_mini.getValues()* ein Tupel mit zwei Werten (Temperatur in ° C und Luftfeuchtigkeit in %). Der BME280 Sensor liefert mit *bme_mini.getValues()* Temperatur, Feuchtigkeit und Luftdruck. Die Sensorwerte können mit dem Befehl *print()* im Terminalfenster angezeigt werden. Für die Anzeige wird oft eine formatierte Ausgabe verwendet.

ZUM SELBST LÖSEN

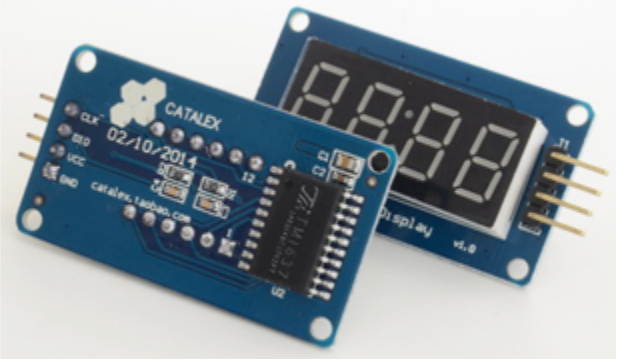
1. Schliesse an den Calliope einen SHT31 oder einen Sensor an. Schreibe ein Programm, welches alle 500 ms die aktuelle die Luftfeuchtigkeit im Terminalfenster anzeigt.

2. Der Temperatursensor misst mit der Messperiode von 500 ms die Temperatur. Ein neuer Messwert soll nur dann angezeigt werden, wenn er sich um mehr als ein 0.5 Grad vom vorhergehenden unterscheidet.

3. Der Temperatursensor misst zu Beginn die Umgebungstemperatur und setzt diesen Messwert als Default-Wert. Danach wird alle 1000 ms die Temperatur gemessen. Wenn die Messerte um 2 Grad höher sind als der Default-Wert, wird ein Alarm ausgelöst und die roten LEDs eingeschaltet. Wenn die Temperatur wieder auf den Default-Wert gesunken ist, wird der Alarm und die roten LEDs ausgeschaltet.

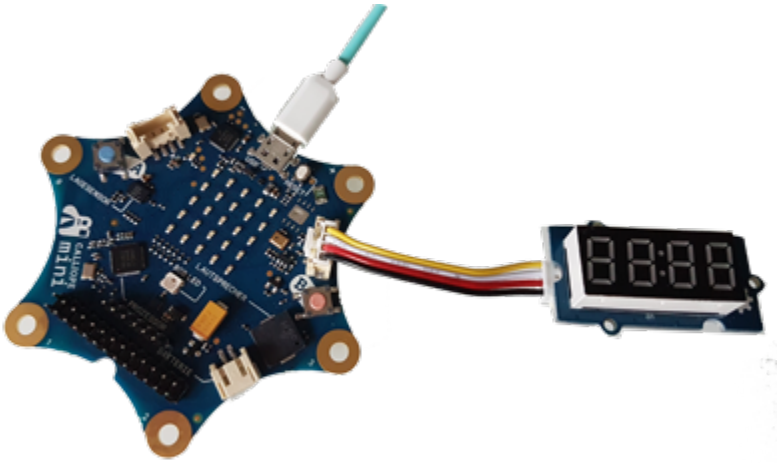
ANHANG: 7-SEGMENT DIGITALANZEIGE

BESCHAFFUNG UND ANSCHLUSS



Die Anzeigeeinheit verwendet den Treiber TM1637 und eine 4-stellige Siebensegmentanzeige mit einem Doppelpunkt in der Mitte. Zur Ansteuerung sind neben der Stromversorgung mit 3V und GND lediglich 2 Leitungen mit den Bezeichnungen *Clock* (CLK) und *Data* (DIO) nötig.

Es wird ein spezielles Protokoll verwendet (ähnlich, aber nicht gleich wie I^2C). Es gibt viele Bezugsquellen (Arduino/Raspberry Pi-Lieferanten, Seed/Grove, eBay, der Preis schwankt zwischen $1 und $10). Es ist darauf zu achten, dass das Display auf für 3V spezifiziert ist. Angeschlossen wird das Display am analogen Grove-Port.



Mit 7 Segmenten kann nur ein beschränkter Zeichensatz dargestellt werden.
Zur Ansteuerung der Anzeige wird eine Pythonmodul *cp7seg* verwendet, das den Programmierer von der trickreichen Kommunikation mit der Anzeige isoliert und ihm einige einfache Darstellungsfunktionen zur Verfügung stellt (siehe API Dokumentation). Du kannst das Modul unter *cp7seg.zip* hinunterladen . Es enthält *cp7seg.py* und eine Minimalversion *cp7segmin.py*. Du kannst das Modul dann im Menü von TigerJython unter *Tools* | *Modul* hinunterladen auf den Calliope kopieren.

TYPISCHE BEISPIELE

1. Von 0 bis 9999 hochzählen

Auf dem Display werden nacheinander die Zahlen von 0 bis 10 000 angezeigt. Die Methode *show(text)* kann direkt Integers anzeigen. Du erkennst, dass für die Anzeige einer einzelnen Zahl ungefähr 30 ms benötigt wird. Mit einer Formatangabe, kannst du die Zahlen rechtsbündig ausschreiben: *d.show("%4d" %n).*

```
from calliope_mini import *

from cp7seg import *
for n in range(10000):
    show(n)
```

2. Die Beschleunigung anzeigen

Die Digitalanzeige wird vielfach eingesetzt, um Messwerte eines Sensors anzuzeigen, beispielweise die x-Komponente der Beschleunigung. Dazu pollst du in einer Endlosschleife den Sensorwert und schreibt in aus. Mit einem *sleep()* wählst du die Periode des Messzyklus, musst aber beachten, dass auch die anderen Aufrufe im Schleifenkörper Prozessorzeit benötigen.

```
from calliope_mini import *
from cp7seg import *

while True:
    acc = accelerometer.get_x()
    show(acc)
    sleep(100)
```

Für Dezimalzahlen wird zuerst mit einer Formatangabe eine vierziffrige Zahl mit 2 Dezimalstellen und Vornullen angefordert (in der Längenangabe wird der Dezimalpunkt mitgezählt). Dann muss der Dezimalpunkt entfernt werden, da dieser ja mit dem Doppelpunkt simuliert wird. Im Programm wird die Beschleunigung in m/s^2 ausgeschrieben.

```
from calliope_mini import *
from cp7seg import *

colon(True)
while True:
    acc = accelerometer.get_x() / 100
    v = "%05.2f" %acc
    v1 = v.replace(".", "")
    show(v1)
    sleep(100)
```

3. Scrolltext

Texte, die mehr als 4 Buchstaben enthalten, kannst du als Scrolltext anzeigen. Die Funktion *scroll()* muss du selbst programmieren.

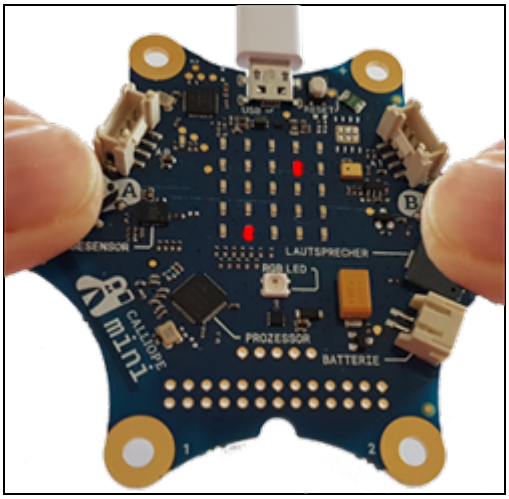
```
from calliope_mini import *
from cp7seg import *
def scroll(text):
    text += "    "
    for i in range(len(text) - 3):
        show(text[i:i + 4])
        sleep(500)
scroll("Happy birthday")
```

ARBEITSBLATT 1: "FANG DAS EI" MIT CALLIOPE

SPIELBESCHREIBUNG

Auf dem 5x5 LED-Display des Calliope bewegt sich ein Pixel auf einer zufälligen Spalte von oben nach unten, analog einem Ei, das von einem Tisch hinunter fällt.

Auf der untersten Zeile kannst du durch Drücken des Buttons A bzw. Buttons B einen Pixel hin und her bewegen, analog einem Korb, mit dem du das Ei auffangen möchtest, bevor es am Boden zerschlägt. Bist du mit dem Korbpixel auf der untersten Zeile dort, wo das Eipixel ankommt, so hast du es "gefangen" und kriegst einen Punkt gutgeschrieben. Das Spiel wird 10 x wiederholt und es geht darum, dass du möglichst viele Eier bzw. Punkte sammelst, also im Maximum 10 von 10.



Nach dem Prinzip von "Teile und Herrsche" teilst du das komplexe Problem in mehrere Teilaufgaben ein.

TEILAUFGABE 1: FALLENDES EIPIXEL

Als erstes schreibst du ein Programm, dass bei jedem Aufruf der Funktion *moveEgg()* das Eipixel auf der fixen Spalte x = 2 um eine Zeile nach unten bewegt. Du gehst von folgendem Programmgerüst aus, und fügt die fehlenden Anweisungen ein:

```
from calliope_mini import *

def move():
    global y, yOld
    # erase old
    ...
    # draw new
    ...
    # update position
    yOld = y
    y = y + 1
    if y == 5:
        y = 0

# Initialization
x = 2
y = 0
yOld = 4

# Animation loop
while True:
    move()
    sleep(300)  # animation speed
```

TEILAUFGABE 2: AUF ZUFÄLLIGER LINIE FALLENDES EIPIXEL

Als nächstes erweiterst du das Programm so, dass die Falllinie zufällig gewählt wird. Verwende den Aufruf *random(a, b)* aus dem Modul *random*, der eine Zufallszahl zwischen a und b (a und b eingeschlossen) liefert Führe dazu analog zu *yOld* auch ein *xOld* ein.

TEILAUFGABE 3: VERSCHIEBBARES KORBPIXEL

Schreibe im gleichen Stil die Funktion *moveBasket()*, welche das Korbpixel auf der untersten Zeile links oder rechts verschiebt, wenn du den linken, bzw. rechten Button drückst. Das Hauptprogramm besteht aus:

```
# Initialization
z = 0

# Animation loop
while True:
    moveBasket()
    sleep(300)  # animation speed
```

wo z die aktuelle x-Koordinate das Korbpixels ist.

TEILAUFGABE 4: ZUSAMMENFÜGEN VON EI UND KORB

Füge nun die Codeteile so zusammen, so dass sich sowohl das Eipixel wie der Korbpixel (allerdings noch unabhängig voneinander) bewegen. Das Hauptprogramm enthält die Schleife:

```
while True:
    moveEgg()
    mobeBasket()
    sleep(300)  # animation speed
```

Du kannst jetzt bereits etwas spielen, indem du versuchst, mit dem Korb am richtigen Ort das Ei zu fangen.

TEILAUFGABE 5: HERAUSFINDEN, OB DER KORB DAS EI FÄNGT

Das Programm soll nun von sich aus herausfinden, ob das Eipixel und das Korbpixel am gleichen Ort zusammentreffen (kollidieren). In diesem Fall erhöhst du die Spielpunktzahl um eins. Den Kollisionstest fügst du am bestem im Hauptprogramm ein, wobei du beachten musst, dass die Position des aktuell angezeigten Eipixels mit xOld, yOld beschrieben wird (x, y ist die Position des nächsten Pixels). Bei einem Hit zeigst du das Bild Image.YES (ein Gutzeichen), bei Misserfolg zeigst du Image.NO (ein Kreuz).

```
if yOld == 4: # egg at bottom
        sleep(1000)
        if xOld == z:  #  hit
            display.show(Image.YES, clear = True)
        else: # miss
```

```
        display.show(Image.NO, clear = True)
```

SPIEL FERTIGSTELLEN UND PERSONALISIEREN

Das Schwierigste ist hinter dir. Du muss jetzt nur noch eine Zählvariable *hits* einfügen, die um eins erhöht wird, falls Ei und Korb zusammentreffen und dann noch dafür sorgen, dass das Spiel genau 10 mal wiederholt wird. Nachher zeigst du auf dem Display die Zahl der Hits an und beendest das Programm. Du kannst es ja durch Klicken des Reset-Buttons beliebig oft starten.

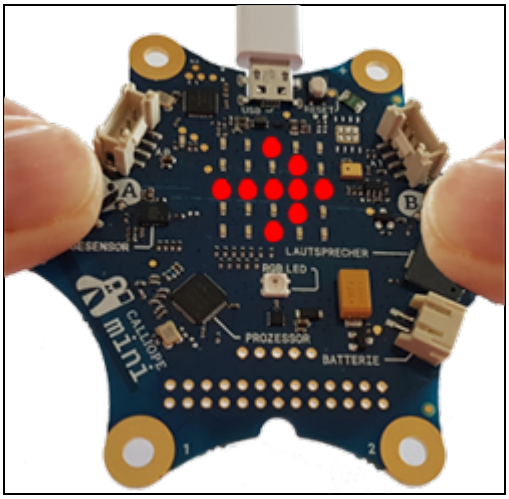
Selbstverständlich ist es nun deiner Fantasie überlassen, dem Spiel durch einige Varianten und Verbesserungen einen persönlichen Touch zu geben. Viel Spass!

ARBEITSBLATT 2: "MEMORY" MIT CALLIOPE

SPIELBESCHREIBUNG

Das Ziel des Spiels ist es, sich an eine möglichst lange Folge von Links-Rechts-Pfeilen zu erinnern. Du kannst damit dein Gedächtnis trainieren!

Das Programm zeigt eine zufällige Folge von Links- bzw. Rechtspfeilen an. Danach musst du diese Folge durch Drücken des linken bzw. rechten Buttons wiedergeben. Ist die Wiedergabe richtig, wird der Vorgang mit einer um einen Pfeil längeren Folge wiederholt, sonst ist das Spiel fertig.



Um die Pfeile auf dem Calliope-Display anzuzeigen verwendest du die Befehle
display.show(Image.ARROW_E)
display.show(Image.ARROW_W)

TEILAUFGABE 1: ZUFÄLLIGE FOLGE MIT ZAHLEN 0 UND 1 ERZEUGEN

Als erstes schreibst du ein Programm welches eine zufällige Folge mit den Zahlen 0 oder 1 erzeugt und diese in einer Liste *seg* speichert. Die Liste ist zu Beginn leer. mit seq.append(x) werden die Zahlen hinzugefügt.Teste das Programm mit verschiedenen n und schreibe die Liste mit *print(seq)* aus.

```
from calliope_mini import *
from random import randint

n = 3
seq = []
repeat n:
    x = randint(0, 1)
    seq.append(x)
print(seq)
```

TEILAUFGABE 2: ZUFÄLLIGE FOLGE MIT ← UND → ERZEUGEN

Für die nächste Teilaufgabe brauchst du eine if-else Struktur. Ergänze das erste Programm wie folgt: Wenn die Zufallszahl 0 ist (if x == 0:), wird ein Linkspfeil angezeigt

```
display.show(Image.ARROW_W)
sleep(500)
```

sonst (wenn die Zufallszahl 1 ist), ein Rechtspfeil

```
display.show(Image.ARROW_E)
sleep(500)
```

Die Pfeile sollen während 500 ms angezeigt (`sleep(500)`)
und danach mit `display.clear()` gelöscht werden.
Dann wartest du wieder 500 ms, bis den nächsten Pfeil angezeigst.

Teste das Programm mit verschiedenen n.

TEILAUFGABE 3: FOLGE MIT ← UND → MIT BUTTONS ERZEUGEN

Im folgenden Beispiel werden die Pfeile mit Drücken der Buttons A bzw, B angezeigt.

```
from calliope_mini import *

while True:
    if button_a.was_pressed():
        display.show(Image.ARROW_W)
        sleep(500)
        display.clear()
    if button_b.was_pressed():
        display.show(Image.ARROW_E)
        sleep(500)
        display.clear()
    sleep(10)
```

Das *sleep(10)* verhindert, dass das Programm in der while True-Schleife unnötig viel Leistung verbraucht.

In einer zweiten Liste *seq2* kannst du mit seq2.append(0) bzw. seq2.append(1) festhalten, ob du den linken oder rechten Button gedrückt hast. Zeige die Liste nach jedem Buttonklick.

TEILAUFGABE 4: INTERAKTION DES SPIELERS EINBAUEN

Im nächsten Entwicklungsschritt fünst du die Teilaufgaben 2 und 3 zusammen. Zuerst wird die Folge seq wie in der Teilaufgaben 2 erzeugt, dann versucht der Spieler mit den Buttons die Folge wiederzugeben, wobei wenn n = 3 ist, muss auch die Liste seq2 nur drei Elemente enthalten.
len(seq2) gibt Anzahl der Elemente des Liste *seq2*.

```
from calliope_mini import *
from random import randint

n = 3
seq = []
repeat n:
    x = randint(0, 1)
    .....

seq2 = []
while len(seq2) < n:
    if button_a.was_pressed():
```

```
        ....
```

Nach der Benutzereingabe vergleichst du die beiden Listen *seq* und *seq2.* Sind sie gleich, so ist die Wiedergabe richtig. Als Rückmeldung kannst du zum Beispiel die Images YES bzw. NO anzeigen.

```
if seq == seq2:
    display.show(Image.YES)
...
```

TEILAUFGABE 5: SCHWIERIGKEITSGRAD STEIGERN

Bisher hat die zufällig erzeugte Sequenz n = 3 Elemente. Nun erweiterst du das Programm so, dass das Spiel mit n = 2 beginnt und bei einer richtigen Wiedergabe n um 1 erhöht wird. Dazu brauchst du eine alles umfassende repeat-Schleife, die so lange läuft, bis die Wiedergabe fehlerhaft ist, und die Schleife nach Anzeige des Image NO mit break abgebrochen wird.

```
from calliope_mini import *
from random import randint

n = 2
repeat:
    #zufallssequenz
    seq = []
    repeat n:
        x = randint(0, 1)
        ....

    #wiedergabe
    seq2 = []
    while len(seq2) < n:
        if button_a.was_pressed():
            .....

    #sequenzen vergleichen
    if seq == seq2:
        ....
        n = n + 1
    else:
        display.show(Image.NO)
        sleep(1000)
        break
```

TEILAUFGABE 6: SPIEL BEENDEN

Am Schluss kannst du noch die Länge der letzten korrekt wiedergegebenen Folge anzeigen, z. Bsp. mit

```
print("Anzahl richtig: ", n-1)
```

oder einem scrollenden Text

```
display.scroll(str(n - 1))
```

TEILAUFGABE 7: DAS SPIEL NACH EIGENEN IDEEN ERWEITERN

Du hast sicher viele Ideen, wie du das Spiel individuell gestalten oder verbessern kannst.

Anregung:
Du kannst das Memory-Spiel auch mit Sound ergänzen. Schaue im Kapitel Sound nach, wie du Töne erzeugen und hörbar machen kannst.

ARBEITSBLATT 3: LICHTSPIELE MIT NEOPIXELS

WAS SIND NEOPIXELS

In absehbarer Zeit wird es nur noch Leuchtmittel geben, die aus LEDs aufgebaut sind. Eine LED ist eine Halbleiter-Diode, die den durchfliessenden Strom mit hohem Wirkungsgrad in Licht umwandelt. Je nach Material kann man verschiedene Farben erzeugen. Kombiniert man eine rote, grüne und blaue LED in einem Gehäuse, so ergibt sich eine Farb-LED, mit der man durch additive Farbmischung jede andere Farbe erzeugen kann. Das Prinzip wird auch beim Farbfernsehen angewendet.

Farb-LEDs werden manchmal mit einer elektronischen Schaltung (Controller) im gleichen Gehäuse kombiniert und hintereinader kaskadiert. Mit nur 3 Zuleitungen kann man trotzdem jeder einzelnen eine andere Farbe geben. Üblicherweise werden die LEDs auf einem LED-Streifen (LEDstrip) montiert. Es gibt aber auch Ringe oder rechteckige Anordnungen (Matrix).

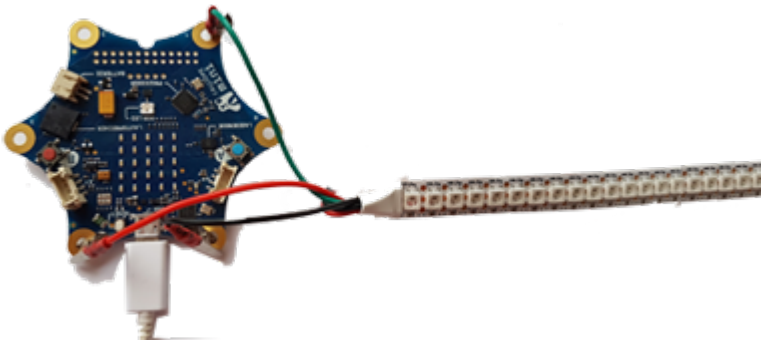
Im Folgenden verwendest du entweder einen LED-Streifen, einen LED-Ring oder eine LED-Matrix mit mindestens 12 LEDs. Diese müssen den **Typ WS2812B** haben. Bezugsquellen: Elektronik-Shops, Online-Versand (Adafruit, Seed, Pi-Shop, Ebay, Aliexpress, usw.)

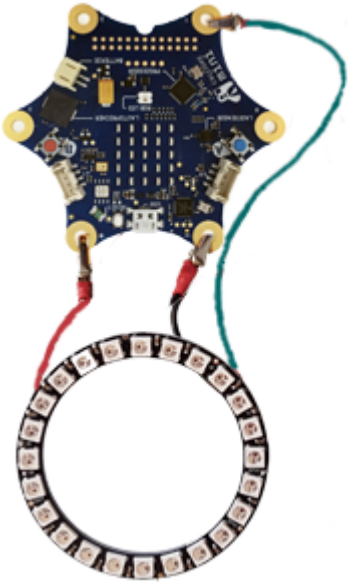
PROGRAMMIEREN MIT GROSSEM SPASSFAKTOR

Ein Lichtsystem, das mit einem Mikrocontroller wie dem Calliope angesteuert wird, ist extrem flexibel und vielseitig, und es es gänzlich deiner Phantasie und deinem Einfallreichtum beim Schreiben des Steuerungsprogramms überlassen, was man schliesslich sehen wird. In diesem Arbeitsblatt lernst du einige grundlegende Verfahren kennen, die du dann auf dein eigenes Projekt übertragen kannst. Wie du sehen wirst, macht das Programmieren von Lichtsystemen deshalb Spass, weil die Auswirkungen jeder Programmzeile sofort optisch sichtbar werden.

ANSCHLUSS

Für einfache Testsysteme kannst du den LED-Streifen, Ring oder Matrix mit drei Leitungen am Calliope anschliessen.





Dabei verwendest du die Pins GND, 3.3V und P1.

Üblich sind folgende Kabelfarben:
GND: schwarz
3.3V (VCC): rot
Datenleitung (DIN): andersfarbig

Achtung: Falls du eine Anordnung mit mehr als 20-30 LEDs verwendest, musst du diese mit einer externen Stromversorgung von 5 V speisen, beispielsweise mit einer PowerBank.

AUFGABE 1: PIXEL AUF BESTIMMTE FARBE SETZEN

Die Ansteuerung der NeoPixels erfolgt mittels eines seriellen Protokolls. Dabei werden der Reihe nach die 3 Farbwerte für das erste Pixel, dann für das zweite usw. gesendet. Das erste Pixel "schnappt" sich seine Werte weg und leitet den Rest des Datenstroms weiter. Unter MicroPython ist die Programmierung sehr einfach: Du erstellst ein "Objekt" NeoPixel, indem du angibt, welches der Datenport ist (üblicherweise pin1) und wieviele NeoPixel deine Anordnung aufweist, also für 24 LEDs:

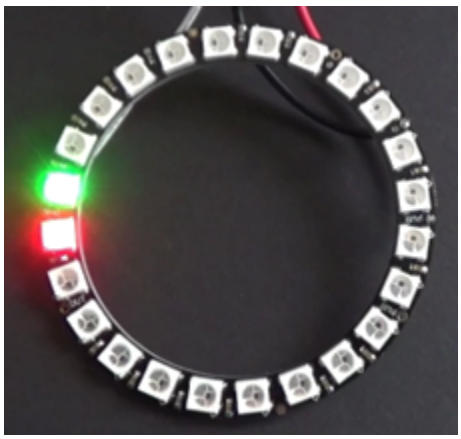
np = NeoPixel(pin1, 24)
(np ist ein beliebiger Variablenname)

Im Speichersystem entsteht nun eine Liste mit 24 Tupels, welche die RGB-Farben der 24 LEDs aufnehmen kann (als (R, G, B)-Farbwerte je im Bereich 0..255).

Auf die einzelnen Werte kannst du mit einem Listenindex zugreifen, also beispielsweise die Farbe der ersten LED auf eine rote Intensität von 30 setzen mit:
np[0] = (30, 0, 0)
oder die Farbe der zweiten LED auf grün setzen mit
np[1] = (0, 30, 0)
Mit dieser Zuweisung ist der Farbwert aber erst in die Liste gesetzt. Damit er auch tatsächlich sichtbar wird, musst du den LEDs die Listeninformation zuzusenden,indem dur
np.show() aufrufst.

Das Programm sieht also so aus:

```
from calliope_mini import *
from neopixel import *

nbLeds = 24
np = NeoPixel(pin1, nbLeds)
np[0] = (30, 0, 0)
np[1] = (0, 30, 0)

np.show()
```



Wie du siehst, musst du neben dem Modul*calliope_mini* auch noch das Modul *neopixel* importieren. Falls du das *np.show()* vergisst, so siehst du keine Veränderung der Farben, da ja deine LEDs erst bei diesem Aufruf den Datenstrom erhalten.

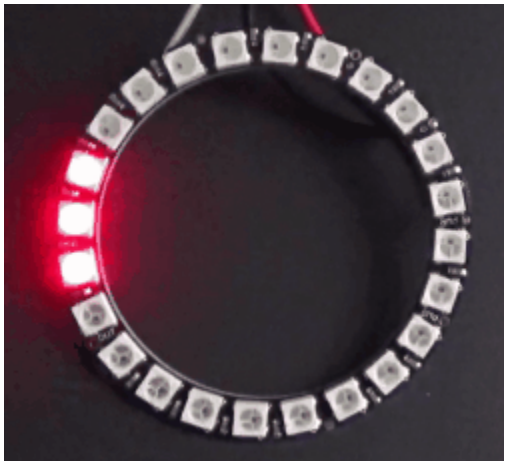
AUFGABE 2: EIN PIXEL LAUFEN LASSEN

Du brauchst vom Programmiertechnischen nur sehr wenig zu wissen, um die folgenden Aufgaben zu lösen. Versuche in dieser ersten Übung ein einzelnes rotes Pixel endlos vom Anfang an das Ende laufen zu lassen. Verwende dazu eine geeignete Wartezeit zwischen dem Pixelwechsel (z.B. 100 ms). Du kannst mit den Farben und Wartezeiten etwas spielen.

Anmerkung:
Um ein einzelnes Pixel zu löschen, verwendest du *np[i] = (0, 0, 0)*. Du kannst auch alle Pixels miteinander mit *np.clear()* löschen.

AUFGABE 3: LÄNGER WERDENDER FARBIGER WURM

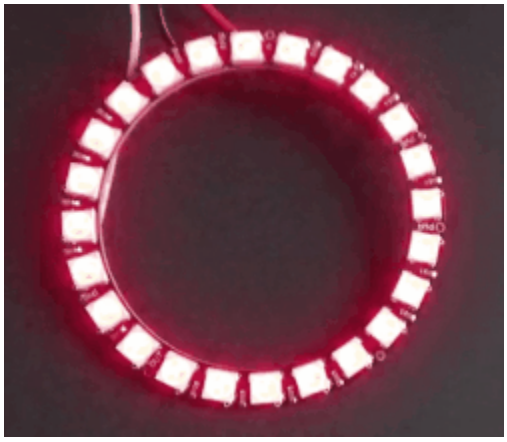
Es ist lustig, einen "Wurm" von gleichfarbigen Pixels zu erzeugen. Dabei soll ausgehend vom gelöschten Zustand die Pixel der Reihe nach mit der gewünschten Farbe angeschaltet werden, bis alle Pixels leuchten. Der Vorgang soll sich endlos mit verschiedenen Farben wiederholen.



Schreibe dazu am besten eine Funktion *wurm(color)*, welche den Wurm erzeugt und rufe sie endlos der Reihe nach mit den Farben rot, grün, blau, gelb, cyan und weiss auf.

Du kannst auch auf das Löschen des alten Wurms verzichten, damit immer alle LEDs eingeschaltet bleiben.

AUFGABE 4: ALLE LEDs MITEINANDER HELL/DUNKEL STEUERN



Für viele Farbspiele möchte man alle LEDs miteinander leuchten lassen. Schreibe ein Programm, das mit fest vorgegebener Farbe alle LEDs in kleinen Schritten hell und wieder dunkel werden lässt. Das Programm soll endlos laufen und die grösste Farbkomponente soll 30 nicht übersteigen.

Spiele mit verschiedenen Farben, die sich auch laufend verändern können.

AUFGABE 5: KERZENFLACKERN

Gehe vom vorhergehenden Programm aus, aber setze die Farbkomponenten beim heller und dunkler werden zufällig. Ist also der momentane maximale Farbwert k, so verwende die Farbe

```
color = int(random() * k, int(random() * k,
        int(random() * k)
```

Du musst dazu das Modul random importieren:
*from random import **

AUFGABE 6: EIGENEN IDEEN VERWIRKLICHEN

Erfinde ein Lichtspiel nach eigenen Ideen.

Dokumentation Callibot und Calliope_mini

Modul callibot
Modul import: from callibot import *

Funktion	Aktion
forward()	setzt den Roboter in Vorwärtsbewegung
backward()	setzt den Roboter in Rückwärtsbewegung
left()	setzt dem Roboter in eine Linksdrehung
right()	setzt dem Roboter in eine Rechtsdrehung
leftArc(radius)	setzt den Roboter auf eine Linkskurve mit gegebenem Radius (in m)
rightArc(radius)	setzt den Roboter auf eine Rechtskurve mit gegebenem Radius (in m)
stop()	stoppt die Bewegung
setSpeed()	setzt die Geschwindigkeit für die Bewegungen (0..100), standard 50
getDistance()	Ultraschallsensor liefert die Distanz (in cm, Bereich 2..150). Falls kein Objekt detektiert: 255 (Simulation -1)
irLeftValue(), irRightValuel()	Infrarotsensoren liefern 1, falls helle Unterlage; 0, falls dunkle Unterlage
setLED(1)	Schaltet beide LEDs ein
setLED(0)	Schaltet beide LEDs aus
setLEDLeft(n), setLEDRight(n)	Schaltet LEDs ein (n = 1) oder aus (n = 0)
setServo(port, angle)	Dreht den am Port S1 oder S2 angeschlossenen Servomotor an die gegebene Winkelposition. (0 <= angle <= 180)
tsValue()	liefert 1, falls beide Touchsensoren gedrückt sind
tsLeftValue(), tsRightValue()	liefert 1, falls der linke bzw. rechte Touchsensor gedrückt ist
RobotContext. enableTrace(True) [nur S]	Roboter hinterlässt Spuren
RobotContext. enableRotCenter(True) [nur S]	Zeichnet bei Bewegungen auf einem Kreisbogen das Rotationszentrum
RobotContext. useBackground("sprite") [nur S]	Fügt ein Hintergrundbild für die Simulation mit Licht- oder Colorsensoren hinzu
RobotContext. setStartPosition(x, y) [nur S]	Setzt den simulierten Roboter an die Position (x, y) im Grafikfenster
RobotContext. setStartDirection(w) [nur S]	Bestimmt die Startrichtung (0: osten, 90: norden, 180: westen, 270: süden)
RobotContext. useObstacle("sprite", x, y) [nur S]	Fügt ein Bild mit Transparrenten Hintergrund an die Position x,y hinzu

setBeamAreaColor() [nur S]	Setzt die Farbe der Strahlbereichsgrenzen
setProximityCircleColor() [nur S]	Setzt die Farbe des Suchkreises
setMeshTriangleColor() [nur S]	Füllfarbe der Maschen
eraseBeamArea() [nur S]	Löscht die Strahlbereichsgrenzen

Modul callibotmot (Motoren einzeln schalten)
Modul import: from callibotmot import *

motL, motR	Linker und rechter Motor
motX.rotate(speed)	Setzt den linken- bzw. rechten Motor in Bewegung mit der Rotationsgeschwindigkeit speed. Für speed > 0 vorwärts, für speed < 0 rückwärts, speed = 0 stopt die Rotation

Modul cbalarm (Alarm ein- und ausschaltenschalten)
(nur Realmodus)
Modul import: from cbalarm import *

setAlarm(1)	Schaltet Alarm ein
setAlarm(1)	Schaltet Alarm aus

Modul calliope_mini

Modul import: from calliope_mini import *

Direkte Funktionsaufrufe

Funktion	Aktion
panic(n)	blockiert das System und zeigt endlos ein "Trauriggesicht" gefolgt von n (für Entwickler)
reset()	startet das System neu (führt main.py aus)
sleep(dt)	hält das Programm während dt Millisekunden an
running_time()	gibt die Zeit in Millisekunden zurück, seit das Board eingeschaltet oder resetted wurde
temperature()	gibt die Temperatur im Lagesensor in Grad Celsius zurück (als Float)

Buttons

button_a	Instanz des Buttons A
button_b	Instanz des Buttons B

Methoden:

is_pressed()	gibt True zurück, falls der Button beim Aufruf gedrückt ist; andernfalls False

was_pressed()	gibt True zurück, falls der Button seit dem letzten Aufruf (oder dem Start des Programms) gedrückt wurde. Ein erneuter Aufruf gibt False zurück, bis der Button wieder gedruckt wird
get_presses()	gibt die Anzahl Tastenbetätigungen seit dem letzten Aufruf (oder dem Start des Programms) zurück. Ein erneuter Aufruf gibt 0 zurück, bis die Taste wieder betätigt wird

FarbLED

led	Instanz der FarbLED

Methoden:

set_red(r)	setzt die rote Farbkomponente (die anderen bleiben gleich) r = 0..255
set_green(g)	setzt die grüne Farbkomponente (die anderen bleiben gleich) g = 0..255
set_blue(r)	setzt die blaue Farbkomponente (die anderen bleiben gleich) b = 0..255
get_red()	liefert die rote Farbkomponente zurück
get_green()	liefert die grüne Farbkomponente zurück
get_blue()	liefert die blaue Farbkomponente zurück
set_colors(r, g, b)	setzt die drei Farbkomponenten r, g, b = 0..255
get_colors()	liefert die drei Farbkomponenten als Tupel zurück
clear()	setzt alle drei Farbkomponenten auf 0

FarbLEDs Callope mini 3
Modul import: from neopixel import *

LEDs = NeoPixel(pin_RGB, 3)	Instanzen LEDs[0], LEDs[1], LEDs[2]

Methoden:

LEDs[0] = (r, g, b)	setzt bei der ersten LED die drei Farbkomponenten, r, g, b = 0..255
LEDs[1] = (r, g, b)	setzt bei der zweiten LED die drei Farbkomponenten, r, g, b = 0..255
LEDs[2] = (r, g, b)	setzt bei der dritten LED die drei Farbkomponenten, r, g, b = 0..255
LEDs.show()	zeigt die gewählte Farben an allen LEDs an

Display

display	Instanz des Displays

Methoden:

set_pixel(x, y, value)	setzt die Intensität des Pixels an der Position x, y. value ist im Bereich 0..9
get_pixel(x, y)	liefert den Wert des Pixels an der Position x, y
clear()	löscht alle Pixels

on()/off()	schaltet den Display an/aus. Im ausgeschalteten Zustand stehen folgende Pins zusätzlich als I/O zur Verfügung: pin4, pin6, pin7, pin9
show(str)	schreibt str auf dem LED-Display aus. Enthält dieser mehrere Zeichen, so werden diese in Laufschrift angezeigt, die auf dem letzten Zeichen stehen bleibt
show(list_of_img, delay = 400, loop = False, wait = True, clear = False)	zeigt alle Images der Liste nacheinander an. Falls loop = True ist, wird die Anzeigesequenz endlos wiederholt. Für wait = True ist die Methode blockierend, andernfalls kehrt sie zurück und die Anzeige erfolge im Hintergrund. delay ist die Anzeigezeit pro Bild in Millisekunden (default: 400). Für clear = True wird die Anzeige nach dem letzten Bild gelöscht
show(img)	zeigt das img auf dem LED-Display. Ist img grösser als 5x5 pixels, so wird der Bereich x, y = 0..4 angezeigt. Ist img kleiner als 5x5 pixels, sind die fehlenden Pixels ausgeschaltet
scroll(str)	zeigt str als Laufschrift. Das letzte Zeichen verschwindet (blockierende Methode)
scroll(str, delay = 150, loop = False, wait = True, monospace = False)	zeigt str als Laufschrift. Falls loop = True ist, wird die Anzeigesequenz endlos wiederholt. Für wait = True ist die Methode blockierend, andernfalls kehrt sie zurück und die Anzeige erfolge im Hintergrund. delay ist die Anzeigezeit pro Spalte in Millisekunden (default: 150)

Image
(Real- und Simulationsmodus)

Image(str)	erzeugt eine Instanz. str hat das Format "aaaaa:bbbbb:ccccc:ddddd:eeeee:", wo a eine Zahl im Bereich 0..9 ist, welche die Intensität des Pixels angibt. a sind die Werte für die erste Zeile, b für die zweite, usw.
Image()	erzeugt eine Instanz mit 5x5 ausgeschalteten Pixels
Image(width, height)	erzeugt eine Instanz mit der gegebenen Zahl horizontaler und vertikaler Pixel, die alle ausgeschaltet sind (value = 0)

Methoden:

set_pixel(x, y, value)	setzt die Intensität des Pixels an der Position x, y. value ist im Bereich 0..9
fill(value)	setzt die Intensität aller Pixels auf den gleichen Wert (value = 0..9)
get_pixel(x, y)	gibt die Intensität des Pixels an der Position x, y
shift_left(n)	gibt ein Image-Objekt zurück, das um n Spalten nach links verschoben ist (rechts eingeschobene Spalten sind ausgeschaltet)
shift_right(n)	gibt ein Image-Objekt zurück, das um n Spalten nach rechts verschoben ist (links eingeschobene Spalten sind ausgeschaltet)
shift_up(n)	gibt ein Image-Objekt zurück, das um n Zeilen nach oben verschoben ist (unten eingeschobene Spalten sind ausgeschaltet)
shift_down(n)	gibt ein Image-Objekt zurück, das um n Zeilen nach unten verschoben ist (oben eingeschobene Spalten sind ausgeschaltet)
copy()	gibt einen Klone des Image zurück
invert()	gibt ein Image-Objekt mit invertieren Pixels zurück (new_value = 9 - value)

crop(x, y, w, h)	gibt einen Bildausschnitt der Breite w und Höhe h zurück. Die obere linke Ecke entspricht dem Pixel x, y des Originalbildes
dest.blit(img, x, y, w, h, xdest, ydest)	kopiert vom gegebenen img einen rechteckigen Bereich an der Position x, y mit Breite w und Höhe h in das Image dest an der Stelle xdest, ydest

Operationen:

image_new = image * n	gibt ein Image-Objekt zurück, bei dem alle Pixel-Intensitäten mit dem Faktor n multipliziert sind
image_new = image1 + image2	gibt ein Image-Objekt zurück, bei dem die Intensitäten der Pixel von image1 und image2 addiert wurden

Vordefinierte Objekte:

- Image.HEART
- Image.HEART_SMALL
- Image.HAPPY
- Image.SMILE
- Image.SAD
- Image.CONFUSED
- Image.ANGRY
- Image.ASLEEP
- Image.SURPRISED
- Image.SILLY
- Image.FABULOUS
- Image.MEH
- Image.YES
- Image.NO
- Image.CLOCK12, Image.CLOCK11, Image.CLOCK10, Image.CLOCK9, Image.CLOCK8, Image.CLOCK7, Image.CLOCK6, Image.CLOCK5, Image.CLOCK4, Image.CLOCK3, Image.CLOCK2, Image.CLOCK1
- Image.ARROW_N, Image.ARROW_NE, Image.ARROW_E, Image.ARROW_SE, Image.ARROW_S, Image.ARROW_SW, Image.ARROW_W, Image.ARROW_NW
- Image.TRIANGLE
- Image.TRIANGLE_LEFT
- Image.CHESSBOARD
- Image.DIAMOND
- Image.DIAMOND_SMALL
- Image.SQUARE
- Image.SQUARE_SMALL
- Image.RABBIT
- Image.COW
- Image.MUSIC_CROTCHET
- Image.MUSIC_QUAVER
- Image.MUSIC_QUAVERS
- Image.PITCHFORK
- Image.XMAS
- Image.PACMAN
- Image.TARGET
- Image.TSHIRT
- Image.ROLLERSKATE
- Image.DUCK
- Image.HOUSE
- Image.TORTOISE
- Image.BUTTERFLY
- Image.STICKFIGURE

- Image.GHOST
- Image.SWORD
- Image.GIRAFFE
- Image.SKULL
- Image.UMBRELLA
- Image.SNAKE
- Listen: Image.ALL_CLOCKS, Image.ALL_ARROWS

General Purpose I/O (GPIO)

Output: 5 mA maximal pro Anschluss 15 mA maximale Last (alle Anschlüsse zusammen)

pin0*, pin1*, pin2*, pin3*, pin18, pin21...pin30	Instanzen für allgemeines Digital-in/Digital-out oder Analog-in/Analog-out (PWM)
pin4... pin15	Instanzen vordefiniert für LED display (display mode)
pin16, pin17	Instanzen vordefiniert für Button A, B (button mode)
pin19, pin20	Instanzen vordefiniert für I2C (i2c mode)

Methoden:

read_digital()	gibt 1 zurück, falls Pin auf logisch 1 (HIGH) ist; gibt 0 zurück, falls Pin auf logisch 0 (LOW) ist (Pulldown 10 kOhm)
write_digital(v)	falls v = 1, wird der Pin auf logisch 1 (HIGH) gesetzt; falls v = 0, wird der Pin auf logisch 0 (LOW) gesetzt (max. Strom: 5 mA)
read_analog()	gibt Wert des ADC im Bereich 0..1023 zurück (Eingangsimpedanz: 10 MOhm) (nur pin1, pin2, pin3, auf pin3 ist der Ausgang des Mikrofon-Verstärkers)
write_analog(v)	setzt den PWM Duty Cycle (v = 0..1023 entsprechend 0.. 100%) (max. Strom: 5 mA)
set_analog_period(period)	setzt die PWM-Periode in Millisekunden
set_analog_period_microseconds(period)	setzt die PWM-Periode in Mikrosekunden (> 300)
set_pull(mode)	setzt den Pullup/Pulldown-Widerstand. (mode: pinx.PULL_UP, pinx.PULL_DOWN, pinx.NO_PULL). Default: Pulldown-Widerstand)

Accelerometer

accelerometer	Instanz des Beschleunigungssensors

Methoden:

get_x(), get_y(), get_z()	gibt die Beschleunigung in x-, y- oder z-Richtung zurück (int, Bereich ca. -2047 bis +2048, entsprechend ungefähr -20 m/s^2 bis +20 m/s^2, die Erdbeschleunigung von ungefähr 10 m/s^2 wird mitgezählt). x-Richtung: FarbLed-USB; y-Richtung: ButtonB-ButtonA; z-Richtung: Normale zum Board von vorne nach hinten

get_values()	gibt ein Tupel mit den Beschleunigungen in x-, y- oder z-Richtung zurück (Einheit wie oben)

Magnetometer

magnetometer	Instanz des Magnetfeldsensors

Methoden:

get_x(), get_y(), get_z()	gibt den aktuellen Wert der x, y oder z-Komponente des Magnetfeldes an der Stelle des Sensors zurück (int, Mikrotesla)
get_values()	gibt ein Tupel der x-, y- und z-Komponenten des Magnetfeldes an der Stelle des Sensors zurück (int, Mikrotesla)

Gyrometer

gyrometer	Instanz des Drehzahlmessers

Methoden:

get_x(), get_y(), get_z()	gibt den aktuellen Wert der x, y oder z-Komponente des Winkelgeschwindigkeit
get_values()	gibt ein Tupel der x-, y- und z-Komponenten der Winkelgeschwindigkeit

---

Modul music
(**Modul import: from music import ***)

Funktionen:

set_tempo(bpm = 120)	setzt die Anzahl Beats pro Minute (default: 120)
pitch(frequency, len, pin = pin0, wait = True)	spielt einen Ton mit gegebener Frequenz in Hertz während der gegebenen Zeit in Millisekunden. pin definiert den Output-Pin am GPIO-Stecker (default: pin0). Falls wait = True, ist die Funktion blockierend; sonst kehrt sie zurück, während der Ton weiter spielt (bis die Abspieldauer erreicht ist oder stop() aufgerufen wird)
play(melody, pin = pin0, wait = True, loop = False)	spielt eine Melodie mit dem aktuellen Tempo.). pin definiert den Output-Pin am GPIO-Stecker (default: pin0). Falls wait = True, ist die Funktion blockierend; sonst kehrt sie zurück, während die Melodie weiter spielt (bis die Abspieldauer erreicht ist oder stop() aufgerufen wird). Falls loop = True, wird die Melodie endlos erneut abgespielt
stop(pin = pin0)	stoppt alle Sound-Ausgaben am gegebenen GPIO-Pin (default: pin0)

Vordefinierte Melodien:

- ADADADUM - Eröffnung von Beethoven’s 5. Sinfonie in C Moll
- ENTERTAINER - Eröffnungsfragment von Scott Joplin’s Ragtime Klassiker “The Entertainer”
- PRELUDE -Eröffnung des ersten Prelude in C Dur von J.S.Bach’s 48 Preludien und Fugen

- ODE - “Ode an Joy” Thema aus Beethoven’s 9. Sinfonie in D Moll
- NYAN - das Nyan Cat Thema
- RINGTONE - ein Klingelton
- FUNK - ein Geräusch für Geheimagenten
- BLUES - ein Boogie-Woogie Blues
- BIRTHDAY - “Happy Birthday to You...”
- WEDDING - der Chorus des Bräutigams aus Wagner’s Oper “Lohengrin”
- FUNERAL - der “Trauerzug”, auch bekannt als Frédéric Chopin’s Klaviersonate No. 2 in B♭Moll
- PUNCHLINE - a lustiger Tonclip, nachdem ein Witz gemacht wurde
- PYTHON - John Philip Sousa’s Marsch “Liberty Bell”, ein Thema aus “Monty Python’s Flying Circus”
- BADDY - Filmclip aus "The Baddy"
- CHASE - Filmclick aus einer Jagdszene
- BA_DING - ein Signalton, der darauf hinweist, dass etwas geschehen ist
- WAWAWAWAA - ein trauriger Posaunenklang
- JUMP_UP - für Spiele, um auf eine Aufwärtsbewegung hinzuweisen
- JUMP_DOWN - für Spiele, um auf eine Abwärtsbewegung hinzuweisen
- POWER_UP - ein Fanfarenklang, der darauf hinweist, dass etwas erreicht wurde
- POWER_DOWN - ein trauriger Fanfarenklang, der darauf hinweist, dass etwas verloren gegangen ist

---

Modul radio:
(**Modul import: from radio import ***)

Computerkommunikation über Bluetooth

Funktionen:

on()	schaltet die Bluetooth-Kommunikation ein. Verbindet mit einem micro:bit mit eingeschaltetem Bluetooth
off()	schaltet die Bluetooth-Kommunikation aus
send(msg)	sendet eine String-Message in den Messagebuffer des Empfängerknotens (First-In-First-Out, FIFO-Buffer)
msg = receive()	gibt die älteste Message (string) des Messagebuffers zurück und entfernt sie aus dem Buffer. Falls der Buffer leer ist, wird None zurückgegeben. Es wird vorausgesetzt, dass die Messages mit send(msg) gesendet wurden, damit sie sich in Strings umwandeln lassen [sonst wird eine ValueError Exception ("received packet is not a string") geworfen]
send_bytes(msg_bytes)	sendet eine Message als Bytes (Klasse *bytes, e.g b'\x01*\x48') in den Messagebuffer des Empfängerknotens (First-In-First-Out, FIFO-Buffer)
receive_bytes()	gibt die älteste Message (bytes) des Messagebuffers zurück und entfernt sie aus dem Buffer. Falls der Buffer leer ist, wird None zurückgegeben. Zum Senden muss send_bytes(msg) verwendet werden (und nicht send(msg))

Modul cpbeeper:
(**Modul import: from cpbeeper import ***)

Tonerzeugung mit dem internen Piezo-Lautsprecher

Funktionen:

beep(pitch = "low",	erzeugt einen Ton (pitch = "low"/"high") mit gegebener Länge (ms), der rep

duration = 100, rep = 1)	Mal wiederholt wird. Die Funktion blockiert, bis der Ton abgespielt ist
beep1(pitch = "low")	dasselbe wie beep(pitch, 70, 1)
beep2(pitch = "low")	dasselbe wie beep(pitch, 70, 2)
beep3(pitch = "low")	dasselbe wie beep(pitch, 70, 3)

Modul sht_mini:
(**Modul import: from sht_mini import ***)
Hochpräziser Temperatur-/Feuchtigkeitsensor SHT32 von Sensirion, angeschlossen am Grove-I2C Port

Funktionen:

sht_mini.getValues()	liefert ein Tupel mit Temperatur (in Grad Celsius) und relativer Luftfeuchtigkeit (in %)

Modul sgp_mini
SGP30 Air Quality Sensor (Co2-Sensor am I2C-Port)
Modul import: import sgp_mini

sgp_mini.getValues()	gibt ein Tupel mit CO2-Konzentration (in ppm) und TVOC (Total Volantile Organic Compunds) zurück. Der Sensor wird auf CO2=400 kalibriert, Bei Werten höber als 1000 ist die Luftqualität schlecht und eine Lüftung dringend notwendig. I2C-Adresse (SGP30: 0x58)

KONTAKT

Die Entwicklergruppe von TigerJython4Kids ist dankbar für jede Art von Rückmeldungen, insbesondere für Fehlermeldungen und Richtigstellungen, Anregungen und Kritik. Wir bieten auch Hilfe und Beratung bei fachlichen oder didaktischen Fragen zu Python und den in TigerJython integrierten Libraries, sowie zur Robotik-Hardware.

Schreiben Sie ein Email an:

help@tigerjython.ch

Lösungen der Aufgaben:
Sind Sie als Lehrperson an einer Ausbildungsinstitution tätig, so können Sie die Lösungen der Aufgaben mit einer Email-Anfrage an die oben genannte Mail-Adresse erhalten. Sie verpflichten sich dabei, die Lösungen nur für den persönlichen Gebrauch zu verwenden, sie nicht zu veröffentlichen und nicht weiter zu geben.