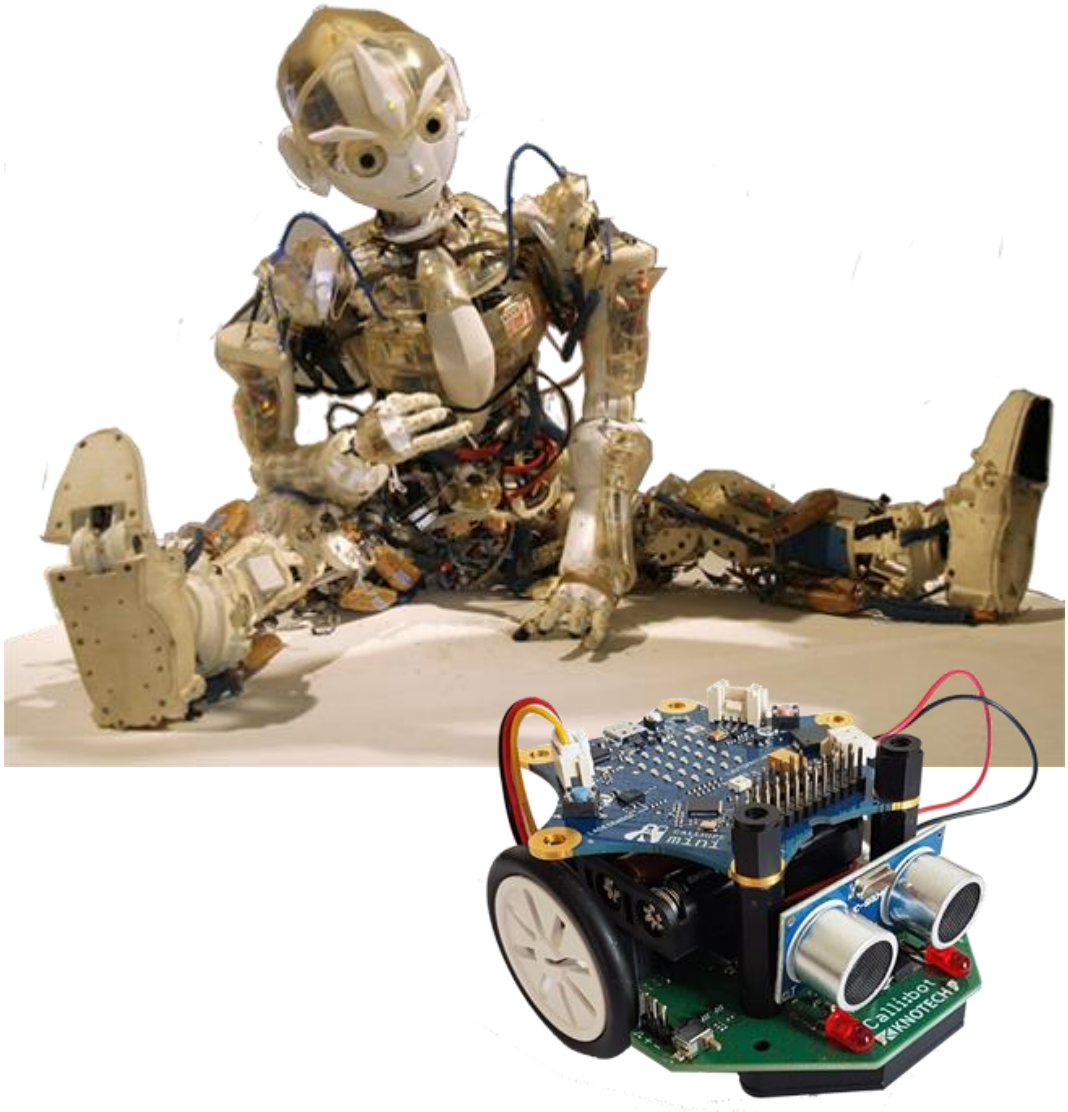
TigerJython4Kids

www.tigerjython4kids.ch

Teil 2: ROBOTIK



mit TigerJython und Calli:bot

www.tigerjython4kids.ch

INHALT

ROBOTIK MIT CALLIBOT

ANHANG:

Version 2.0, März 2024

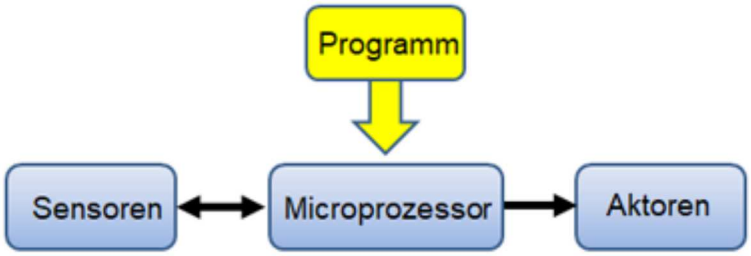
Kontakt: help@tigerjython.com

WAS IST EIN ROBOTER ?

Roboter sind computergesteuerte Maschinen, die automatisierte Tätigkeiten ausführen können. Sie treten in ganz verschiedenen Erscheinungsformen auf, beispielsweise als Industrieroboter, selbstfahrende Autos, Weltraumteleskope oder menschenähnliche Geräte. Eingesetzt werden sie in der Industrie, Technik, Medizin, Forschung, Wissenschaft und praktisch allen Lebensbereichen.

Roboter werden durch eingebaute Computer (Microcontroller) gesteuert. Diese müssen geschickt **programmiert** werden, damit der Roboter die geplanten Aufgaben lösen kann. Roboter der neuesten Generation können mit ihren Sensoren Daten aus ihrer Umgebung erfassen, selbständig Entscheidungen treffen und ihr Verhalten selbst modifizieren. Sie werden daher als **selbstlernende Roboter** oder als "**intelligente Systeme**" bezeichnet.

Der wichtigste Bestandteil eines Roboters ist ein **Mikroprozessor.** Er ist verantwortlich für die Programmausführung, Abfrage und Auswertung der Sensorwerte und Steuerung der aktiven Komponenten. Einige Roboter sind mit mehreren Mikroprozessoren (Coprozessoren) ausgerüstet, die miteinander kommunizieren.



Sensoren sind Messgeräte, welche physikalischen Grössen, wie Helligkeit, Distanz, Temperatur, Luftfeuchtigkeit usw. messen und an den Mikroprozessor weiterleiten.

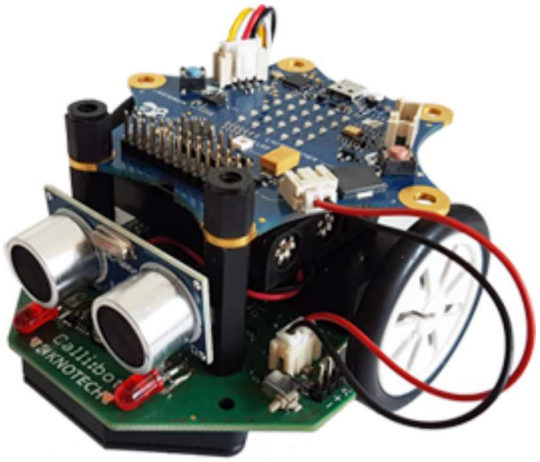
Die **Aktoren** sind die aktiven Komponenten eines Roboters (Motoren, LEDs, Soundbuzzer). Sie führen Befehle, die sie vom Mikroprozessor erhalten, aus.

Der Aufbau und die Funktionsweise von Robotern lassen sich auch an einfachen Roboter-Modellen aufzeigen. TigerJython stellt Robotik-Bibliotheken und die notwendigen Tools zur Verfügung, die es ermöglichen die ersten Erfahrungen mit Robotik zu machen.

ROBOTIK MIT CALLI:BOT

Der Calliope Mini ist ein programmierbarer Computer im Taschenformat, der für Bildungszwecke entwickelt wurde. Er besteht aus einer sternförmigen Platine mit einem 32-bit Microcontroller, Flashspeicher, 25 roten LEDs, zwei Buttons, einigen Sensoren und einer USB-Schnittstelle.

Zusammen mit **Calli:bot**, dem kompakten und günstigen Fahrwerk von www.knotech.de, wird er zu einem fahrenden Roboter, mit dem man alle wichtige Robotik-Konzepte aufzeigen kann.



SIMULATION

Falls du dein Programm zuerst mit dem Computer testen möchtest oder keinen Roboter zur Hand hast, so kannst du den **Simulationsmodus** verwenden. Fast alle Beispiele und Aufgaben aus diesem Lehrgang lassen sich auch im Simulationsmodus ausführen.

Alle Bespiele kannst du herunterladen unter
http://tigerjython4kids.ch/download/callibotEx.zip

EINRICHTEN

ZUSAMMENBAU

Der **Knotech Bausatz** enthält eine 8 x 8.5 cm grosse Platine, an der bereits Motoren, LEDs, Infrarot- und Ultraschalsensoren und weitere Komponenten eingebaut sind. Der Zusammenbau ist sehr einfach, beansprucht nur wenige Minuten.

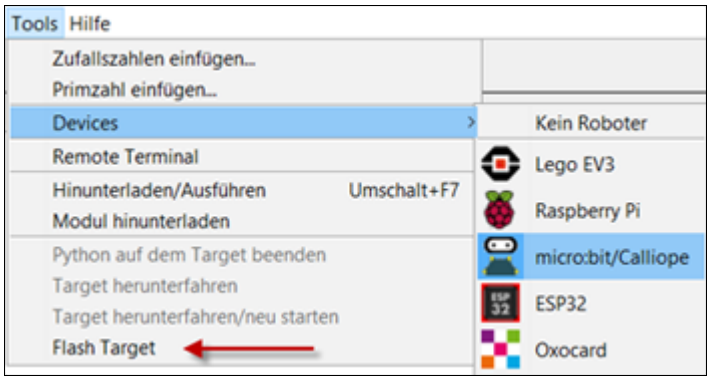
Zusätzlich muss man einen **Calliope mini** und drei AA Batterien beschaffen.

FIRMWARE INSTALLIEREN

Vor der ersten Verwendung muss du auf dem Calliope eine ***Firmware*** installieren. Diese ist notwendig, um Python-Programme auf dem Calli:bot auszuführen. Die Firmware wird mit TigerJython via USB-Kabel auf den Calliope des Roboters hinunterladen. Falls du die Entwicklungsumgebung TigerJython noch nicht installiert hast, lade die Installationsdatei von http://www.tjgroup.ch/download heruter und installiere sie im beliebigen Verzeichnis auf deiner Ferstplatte.

Verbinde den Calliope via USB Kabel mit dem Computer. Im Dateiexplorer erscheint er als ein externes Laufwerk.

Wähle im TigerJython Tools/ Devices/ micro:bit/Calliope und **Flash Target**.



Für ältere WIndows Versionen (< 10) muss du einen Treiber installieren (http://www.tigerjython4kids.ch/download/mbed.zip).

Die Programme für den Callibot kannst du auch mit dem **WebTigerPython** (https://webtigerpython.ethz.ch) entwickeln, der keine lokale Installation benötigt und nur im Webbrowser auch auf iPads und Tabletts läuft. Die Online-Version ersetzt aber nicht vollständig den lokal installierten **TigerJython.** Es fehlt die Roboter-Simulation. Die Programmcodes sind aber kompatibel. Die Online erstellten Programme kannst du problemlos mit TigerJython weiter bearbeitet.

1. LOSLEGEN

DU LERNST HIER...

wie du den Callibot bedienst und ein erstes Roboterprogramm erstellst, dass du auf den Roboter hinunterlädst und dort ausführst.

DAS ERSTE PROGRAMM AUSFÜHREN

Für die Programmentwicklung verwendest du die Entwicklungsumgebung **TigerJython**. Falls sie auf deinem Computer noch nicht installiert ist, musst du sie zuerst hinunterladen.

Starte TigerJython und gibt im Editorfenster das folgende Programm ein:

```
from callibot import *

forward()
delay(2000)
stop()
```

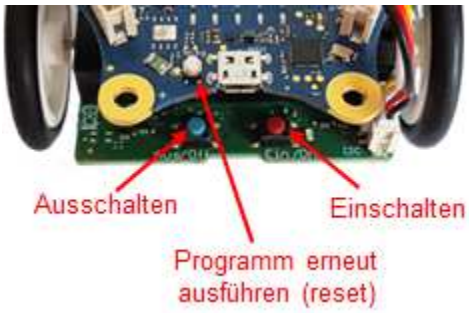
Mit der ersten Zeile importierst du die Robotik-Bibliothek *callibot*. Mit dem Befehl forward() wird der Roboter in die Vorwärtsbewegung versetzt und bleibt 2000 Millisekunden in diesem Zustand. Mit dem Befehl stop() wird er angehalten.

Schliesse den Roboter mit einem USB-Kabel an deinem Computer an.

Wähle unter *Tools / Devices / micro:bit/calliope*. Die Einstellungen bleiben gespeichert. Du kannst sie aber jeder Zeit anschauen oder ändern. Dazu klickst du auf den Button *Einstellungen* in der Menüleiste und wählst das Register *Bibliotheken*.

Schalte den Roboter mit dem kleinen roten Schalter hinten auf dem Chassis ein und klicke auf den Robotik-Button "Hinunterladen/Ausführen".

Der Roboter fährt eine kurze Strecke vorwärts. Während der Programmausführung wird auf deinem Computer ein **Terminalfenster** angezeigt, in dem Fehlermeldungen vom Callibot angezeigt werden. Falls du vergessen hast, den Roboter einzuschalten erscheint die Meldung "Switch the robot on".



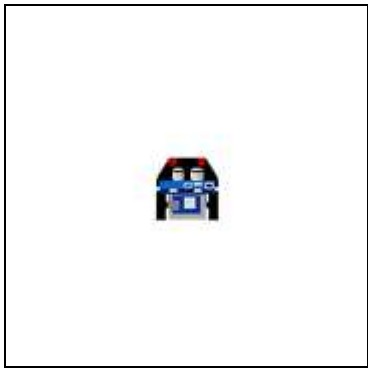
Für die praktische Arbeit mit den fahrenden Roboter ist es oft besser, zuerst das Programm herunterladen, dann den USB-Kabel ausziehen, den Roboter am Boden oder am Ort, wo er fahren kann, platzieren und erst danach den Roboter einschalten und das Programm mit dem reset-Button starten..

Das zuletzt heruntergeladene Programm ist auf dem Roboter gespeichert. Du kannst den USB-Kabel ausziehen und das Programm mit dem **Reset-Button**, der sich neben dem USB-Stecker befindet, erneut ausführen.

Falls du längere Zeit den Roboter nicht brauchst, solltest du die Stromversorgung wieder ausschalten (blauer Schalter), damit die Batterien nicht unnötig entladen werden.

SIMULATIONSMODUS

Die meisten Programme können auch im Simulationsmodus ausgeführt werden. Du verwendest das gleiche Programm, zum Ausführen klickst du aber anstelle des Robotik-Buttons auf den grünen Run-Pfeil.



MERKE DIR...

Du gibst den Programmcode im TigerJython-Editor ein und klickst auf den Roboter-Button, um das Programm auf den Callibot herunterzuladen. Der Calliope des Roboters muss dabei über das USB-Kabel an deinem Computer angeschlossen sein. Vor der Ausführung musst du mit dem kleinen roten Schalter hinten am Callibot die Stromversorgung einschalten.

Für die Simulation startest du die Programmausführung mit dem grünen Pfeil.

2. ROBOTER BEWEGEN

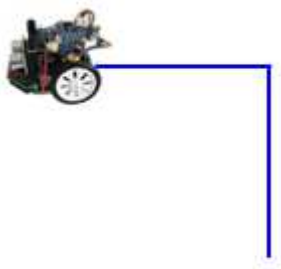
DU LERNST HIER...

die wichtigsten Befehle, mit welchen du deinen Roboter steuern kannst.

MUSTERBEISPIELE

Beispiel 1: **Fahren und abbiegen**
Der Roboter soll zuerst vorwärts fahren, dann links abbiegen und danach wieder ein kurzes Stück vorwärts fahren.

Die Zeilen deines Programms werden der Reihe nach ausgeführt. Mit dem Befehl forward() wird der Roboter in den Zustand "forward" versetzt. delay(2000) gibt die Zeit im Millisekunden an, bis der Mikroprozessor den nächsten Befehl left() erteilt. Dieser versetzt den Roboter in den Zustand "linksdrehen". Während der 550 Millisekunden dreht sich der Roboter ca. um 90° links. Danach fährt er wieder 2000 Millisekunden vorwärts, eher er den Befehl stop() erhält.



```
from callibot import *

forward()
delay(2000)
left()
delay(550)
forward()
delay(2000)
stop()
```

Der Drehwinkel, welchen der Roboter in 550 Millisekunden zurücklegt ist von der Unterlage und insbesondere auch vom Ladezustand der Batterien abhängig. Wenn sich der Roboter zu lansam bewegt und nur um einen kleinen Winkel dreht, muss du die Batterien wechseln. Teste das Programm auch im Simulationsmodus. Hier dreht der Roboter imme genau um 90°.

Beispiel 2: **Mehreres gleichzeitig tun**
Obwohl das nachfolgende Programm aus einer Sequenz von Anweisungen besteht, die der Reihe nach ausgeführt werden, kann der Roboter mehreres gleichzeitig tun. Der Roboter fährt während 4 s vorwärts und anschliessend 4 s rückwärts. Während des Rückwärtsfahrens schaltet er die LEDs ein und aus.

Die Motoren und die LEDs werden durch den Mikroprozessor gesteuert, wobei der Mikroprozessor auch mehrere Aktoren gleichzeitig bedienen kann. Während sich die Motoren nach dem Erhalt des Befehls backward() im Zustand *backward* befinden, werden mit dem Befehl

setLED(1) die LEDs einschaltet und nach 3000 ms mit setLED(0) auschaltet. Nach weiteren 1000 ms erhalten die Motoren den Befehl *stop()* und der Roboter hält an.


```
from callibot import *

forward()
delay(4000)
backward()
setLED(1)
delay(3000)
setLED(0)
delay(1000)
stop()
```

Beispiel 3: **Auf Kreisbogen fahren**
Der Roboter soll zuerst 5 s auf einem Linksbogen, dann 5 s auf einem Rechtsbogen und anhalten. Für das Bogenfahren verwendest du die Befehle leftArc(r) bzw. rightArc(r), wobei der Radius r in Meter ist.

Mit setSpeed(speed) kannst du die Geschwindigkeit ändern. Als *speed* kannst du Werte zwischen 0 und 100 wählen. Default *speed* ist 50.

```
from callibot import *

setSpeed(30)
leftArc(0.2)
delay(5000)
rightArc(0.2)
delay(5000)
stop()
```

MERKE DIR...

Für die Steuerung des Roboters stehen die folgende Befehle zur Verfügung:

forward()	fährt vorwärts
backward()	fährt rückwärts
left()	dreht links
right()	dreht rechts
leftArc(r)	fährt auf Linksbogen mit Radius r
rightArc(r)	fährt auf Rechtsbogen mit Radius r
stop()	hält an
setSpeed(speed)	setzt die Geschwindigkeit (default 50)
delay(ms)	bleibt ms Millisekunden im gleichen Zustand

setLED(1)	LEDs einschalten
setLED(0)	LEDs ausschalten
setAlarm(1)	Schaltet einen Pipton ein
setAlarm(0)	Schaltet den Pipton aus

Bei der Roboterprogrammierung musst du in Zuständen denken. Der Befehl *forward()* versetzt den Roboter in eine Vorwärtsbewegung. Während die Motoren in diesem Zustand sind, kann der Microprozessor Befehle an andere Aktoren erteilen.

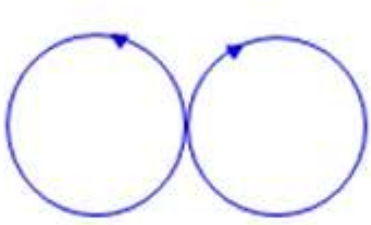
Ein laufendes Programm kannst du jeder Zeit abbrechen, indem du den kleinen Switchschalter hinten auf dem Roboterchassis auf "OFF" stellst.

Du kannst alle Beispiele auch im **Simulationsmodus** ausführen. Um ein Programm zu starten, klickst du anstelle des Robotik-Buttons auf den grünen Run-Pfeil. Willst du im Grafikfenster die **Roboterspuren** anzeigen lassen, fügst du nach der Importzeile folgende Zeile ein:
RobotContext.enableTrace(True)

ZUM SELBST LÖSEN

1. Erstelle mit einigen Gegenständen einen Parcours und schreibe ein Programm so, dass der Roboter vom Start zum Ziel fährt.

2. Der Roboter soll sich so lange auf einen Linksbogen bewegen, bis er einen ganzen Kreis zurückgelegt hat. Danach soll er sich gleich lange auf einem rechtsbogen bewegen.



3. Mit **setAlarm(1)** wird ein Alarm eingeschaltet (ein Pipton). **setAlarm(0)** schaltet den Alarm aus. Der Roboter soll zuerst 5000 ms vorwärts und danach 5000 ms rückwärts fahren. Während des Rückwärtsfahrens soll er Alarm einschalten.

   Damit du den Befehl setAlarm() verwenden kannst, musst du das Modul *cbalarm* importiren.

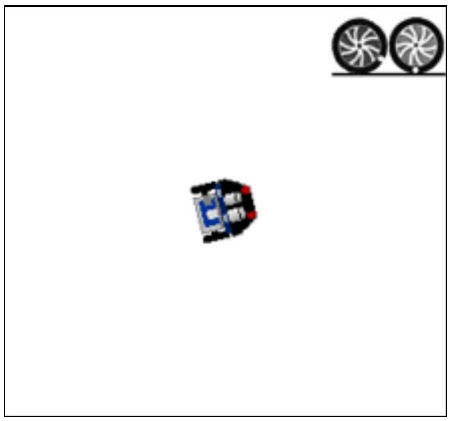
   Mit nebenstehendem Programm wird der Alarm während 3000 ms eingeschaltet.

```
from callibot import *
from cbalarm import *

setAlarm(1)
delay(3000)
setAlarm(0)
```

ZUSATZSTOFF: MOTOREN UND LEDs EINZELN STEUERN

Die Befehle *forward()*, *left()* usw. steuern das ganze Fahrwerk, das aus zwei Motoren besteht. Man kann aber auch die einzelnen Motoren einschalten und ihre Geschwindigkeit regeln. Mit motL.rotate(speed) wird der linke Motor in die Vorwärtsbewegung versetzt und rotiert mit der Geschwindigkeit *speed*. Für die positiven speed-Werte rotiert der Motor vorwärts, bei negativer Werten rückwärts und für speed = 0 hält der Motor an. Entsprechend funktioniert der Befehl *motR.rotate(speed)* für den rechten Motor.

Beispiel 4: Der linke Motor läuft zuerst 3 Sekunden mit der Geschwindigkeit 50 vorwärts, dann 2 Sekunden mit der Geschwindigkeit 30 rückwärts. Anschliessend hält er an. Führe das Programm zuerst im Simulationsmodus aus und beobachte die Rotationen im Grafikfenster.



```
from callibotmot import *

motL.rotate(50)
delay(3000)
motL.rotate(-50)
delay(2000)
motL.rotate(0)
```

Beispiel 5: Leds einzeln ein - und ausschalten
Mit den Befehlen *setLED(1)* und *setLED(0)* kannst du beide LEDs gleichzeitig ein- und ausschalten. Du kannst die LEDs auch einzeln ansprechen:
setLedLeft(1) schaltet die linke LED ein
setLedLeft(0) schaltet die linke LED aus
setLedRight(1) schaltet die rechte LED ein
setLedRight(0) schaltet die rechte LED aus

Dein Programm schaltet jeweils eine LED ein und die andere aus.

```
from callibot import *

setLEDLeft(1)
delay(2000)
setLEDLeft(0)
setLEDRight(1)
delay(2000)
setLEDRight(0)
```

ZUM SELBST LÖSEN

4. Lasse die beiden Motoren mit der gleichen Geschwindigkeit 4000 Millisekunden vorwärts rotieren.

5. Lasse während 4 Sekunden den linke Motor vorwärts und den rechten Motor rückwärts laufen.

6. Wie musst du den linken und den rechten Motor steuern, damit sich der Roboter 4 Sekunden auf einem Linksbogen bewegt?

7. Ergänze das Beispiel 3 so, dass wenn der Roboter auf dem linken Kreisbogen fährt die linke LED leuchtet und wenn er auf dem rechten Kreisbogen fährt die rechte LED leuchtet und die linke ausgeschaltet wird.

3. BEFEHLE WIEDERHOLEN, FUNKTIONEN

DU LERNST HIER...

wie ein Roboter bestimmte Befehlssequenzen wiederholen kann und wie du mit benannten Programmblöcken (Funktionen) deine Programme besser strukturieren kannst.

MUSTERBEISPIELE

Beispiel 1: Quadrat fahren
Um ein Quadrat zu fahren, muss der Roboter vier Mal eine Strecke vorwärts fahren und um 90° drehen. Für die sich wiederholende Bewegung verwendest du eine repeat-Schleife. Diese wird mit repeat eingeleitet gefolgt von Anzahl Wiederholungen und einem Doppelpunkt. Die Befehle, die wiederholt werden sollen, müssen eingerückt sein.

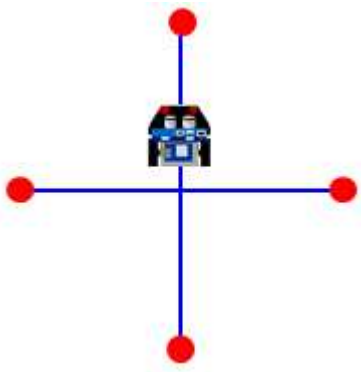
```
from callibot import *

repeat 4:
    forward()
    delay(2000)
    left()
    delay(550)
stop()
```

Führe das Programm zuerst im Simulationsmodus aus. Im Realmodus musst du eventuell die Zeit beim Linksdrehen anpassen.

Im Unterschied zur Simulation, bei der der Roboter ein Quadrat ganz exakt abfahren kann, ist es beim realen Roboter schwierig, exakt geradeaus zu fahren und exakt in einem rechten Winkel abzubiegen. Dies entspricht der Wirklichkeit, denn kein Auto wird bei starrer Radstellung, d.h. blockierter Steuerung, je exakt geradeaus fahren, man muss immer wieder regulierend eingreifen. Deswegen sind Roboter mit Sensoren ausgerüstet, die ihnen helfen, diese Ungenauigkeiten zu korrigieren. Verwende also nicht zu viel Zeit, um dem Roboter ein exaktes Quadrat fahren beizubringen.

Beispiel 2: Befehle beliebig oft wiederholen
In vielen Situationen führt ein Roboter gewisse Tätigkeiten "endlos" aus, bzw. solange bis er abgeschaltet oder das Programm abgebrochen wird. Zum Beispiel, wenn er ständig Sensordaten zurückmelden muss. Dafür verwendet man eine repeat-Schleife ohne Anzahl Wiederholungen.

In deinem Beispiel besucht der Roboter endlos vier Orte, die auf Wegen liegen, die rechtwinklig zu einander stehen und fährt jeweils wieder rückwärts zum Ausgangspunkt.



```
from callibot import *

repeat:
    forward()
    delay(2000)
    backward()
    delay(2000)
    left()
    delay(550)
```

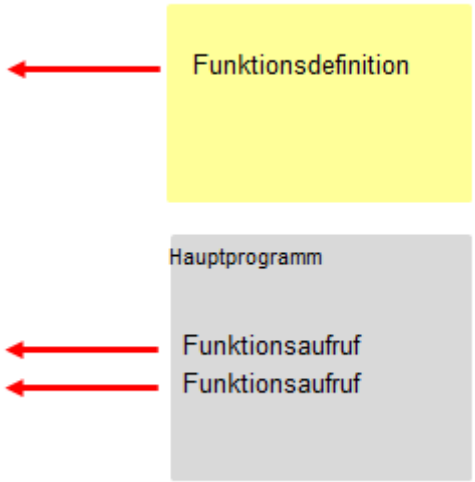
Hier ist es nützlich zu wissen, wie man ein laufendes Programm abbrechen kann: Am einfachsten geht es mit Drücken des blauen Buttons hinten am Chassis.

Beispiel 3: Programme mit eigenen Funktionen strukturieren
Mit benannten Programmblöcken, in Python Funktionen genannt, kannst du deine Programme besser strukturieren. Es ist eine wichtige Programmiertechnik, denn komplexere Programme können mit Hilfe von Funktionen in kleinere, leichtprogrammierbare Teilprogramme zerlegt werden. Mit Vorteil werden Funktionen auch eingesetzt, wenn ein Programmblock mehrmals verwendet wird.

Eine Funktionsdefinition beginnt immer mit dem Schlüsselwort def. Darauf folgen ein Funktionsname, eine Parameterklammer und ein Doppelpunkt. Die Anweisungen im Funktionskörper sind eingerückt. Im Hauptprogramm wird die Funktion aufgerufen.

In deinem Beispiel definierst du eine Funktion blink(), die das einmalige aufleuchten der roten LED bewirkt. Im Hauptprogramm fährt der Roboter zuerst 200 ms vorwärts und danach 2000 ms rückwärts. Während des Rückwärtsfahren blinkt er zweimal mit beiden LEDs. Diese Bewegung wiederholt er zweimal.

```
from callibot import *

def blink():
    setLED(1)
    delay(500)
    setLED(0)
    delay(500)

repeat 2:
    forward()
    delay(2000)
    backward()
    blink()
    blink()
    stop()
```



MERKE DIR...

Um ein Programmblock n mal zu wiederholen, verwendest du eine repeat-Schleife:

```
repeat n:
    Anweisungen
```

Um ein Programmblock endlos zu wiederholen, verwendest du eine Endlos-Schleife:

```
repeat:
    Anweisungen
```

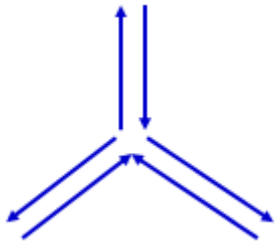
Die eingerückten Zeilen werden so lange wiederholt, bis man das Programm mit Schliessen des Terminalfensters abbricht.

Mit **Funktionen** kannst du deine Programme besser strukturieren und Code-Dupplikation vermeiden. Du musst zwischen der Funktionsdefinition und Funktionsaufruf unterscheiden.

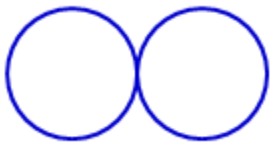
Definition:
```
def name():
    Anweisungen
```
Aufruf:
```
name()
```

ZUM SELBST LÖSEN

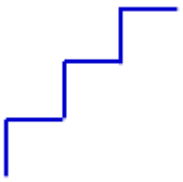
1. Der Roboter startet in der Mitte, fährt eine kurze Strecke vorwärts, dreht um 180° und fährt zurück zum Ausgangspunkt und dreht wieder in die Startrichtung. Dann dreht er etwa um 120° nach rechts. Diese Befehlssequenz wiederholt er drei Mal.



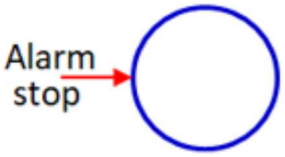
2. Der Roboter soll "endlos" zuerst einen Kreis auf dem Linksbogen und dann einen Kreis auf dem Rechtsbogen fahren.



3. Definiere eine Funktion *step()*, mit der dein Roboter ein Tritt abfährt. Rufe die Funktion dann dreimal auf, so dass der Roboter die nebenstehende Figur abfährt. Löse die Aufgabe zuerst im Simulationsmodus.



4. Der Roboter bewegt sich "endlos" auf einer Kreisbahn mit dem Radius 0.2 m. Jedesmal wenn er einen ganze Kreis zurückgelegt hat, hält er für 3 Sekunden an und schaltet während dieser Zeit den Alarm ein. Danach schaltet er den Alarm aus und bewegt sich weiter auf der Kreisbahn. Löse die Aufgabe zuerst im Simulationsmodus.



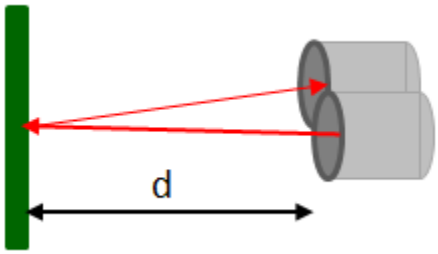
4. ULTRASCHALLSENSOR (DISTANZSENSOR)

DU LERNST HIER...

wie du mit einen Ultraschallsensor Distanzen messen und auswerten kannst.

WIE FUNKTIONIERT EIN ULTRASCHALLSENSOR?

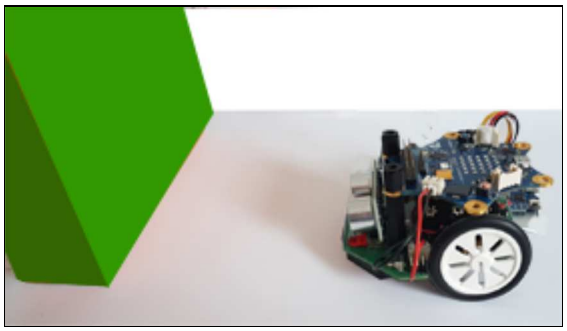
Ein Ultraschallsensor hat einen Sender und einen Empfänger für die Ultraschallwellen. Bei einer Distanzmessung sendet er einen kurzen Ultraschallpuls und misst die Zeit, die dieser benötigt, um zum Objekt und wieder zurück zu laufen. Daraus kann er mit Hilfe der bekannten Schallgeschwindigkeit die Distanz ermitteln.



Der Callibot-Sensor kann die Distanzen im Bereich von ca. 5 bis 200 cm messen und liefert die Werte in cm zurück. Wenn kein Objekt im Messbereich ist, gibt er den Wert 255 zurück.

MUSTERBEISPIELE

Beispiel 1: Ein Objekt erkennen
Der Roboter fährt vorwärts und misst alle 200 Millisekunden die Entfernung zum Objekt, der sich vor ihm befindet. Wenn die Entfernung kleiner als 20 cm ist, hält er an.



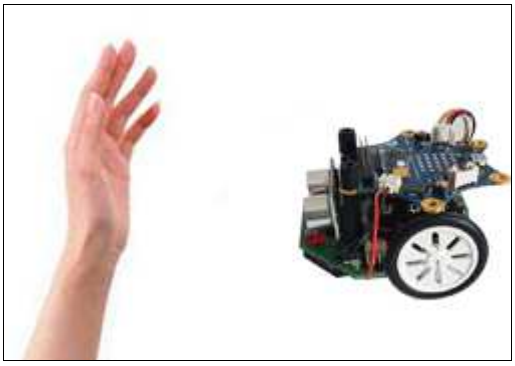
```
from callibot import *

forward()
repeat:
    d = getDistance()
    print(d)
    if d < 20:
        stop()
    delay(200)
```

Der Befehl getDistance() gibt die Distanz zum Objekt zurück. Der Sensorwert wird in einer Endlosschleife abgefragt und in der Variablen *d* gespeichert. delay(200) gibt die **Messperiode** an. Für die Überprüfung der Sensorwerte verwendest du die if-Struktur. Die eingerückte Anweisung *stop()* wird nur dann ausgeführt, wenn die Bedingung nach *if* erfüllt ist. Falls der Roboter via USB-Kable mit dem Computer verbunden ist, werden mit *print(d)* die Sensorwerte im Terminalfenster angezeigt.

Beispiel 2: Ein Regelungssystem für die Distanz zum Objekt

Dein Programm soll dafür sorgen, dass der Roboter in einer bestimmten Distanz zu deiner Hand bleibt. Ist er zu nahe, soll er rückwärts, sonst vorwärts fahren.

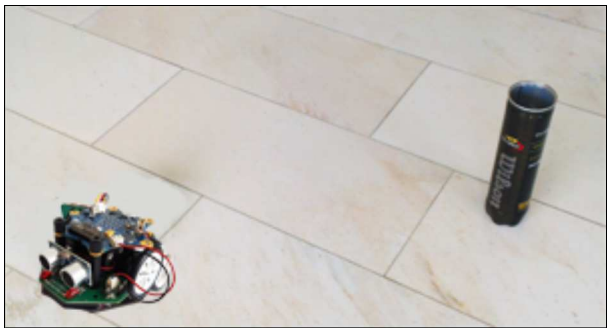
Für die Auswertung der Sensorwerte verwendest du die *if-else-Struktur*. Falls die Bedingung nach *if* stimmt, wird die Anweisung nach *if* ausgeführt, sonst die Anweisung nach else.



```
from callibot import *

forward()
setSpeed(30)
repeat:
    d = getDistance()
    if d < 20:
        backward()
    else:
        forward()
    delay(100)
```

Beispiel 3: Objekt suchen

Ein Roboter mit einem Ultraschallsensor soll ein Objekt finden und danach zu ihm fahren und ihn umstossen. Der Roboter steht beim Start in einer beliebigen Richtung, d.h. er muss zuerst drehen, bis er den Gegenstand detektiert.



```
from callibot import *

def searchTarget():
   repeat:
      right()
      delay(50)
      dist = getDistance()
      if dist != 255:
         right()
         delay(500)
         break

setSpeed(20)
searchTarget()
setSpeed(50)
forward()
```

Den Suchvorgang definierst du in der Funktion searchTarget(). Der Roboter dreht jeweils um einen kleinen Winkel nach rechts und misst die Distanz. Falls er ein Objekt detektiert (der Sensor gibt nicht mehr den Wert 255 zurück), dreht er noch ein wenig weiter, damit er gegen Mitte des Objekts gerichtet ist. Mit break kannst du die repeat-Schleife abbrechen und damit den Suchvorgang beenden. Im Hauptprogramm wird die Funktion *searchTarge()* aufgerufen und der Roboter fährt nach dem erfolgreichen Suchvorgang zum Objekt. Du kannst das Programm noch optimieren, indem du den Roboter stopst, sobald er näher als 5 cm beim Gegenstand ist.

Eine moderne Industrieanlage ohne Sensoren ist heute kaum mehr denkbar. Auch in unserem Alltag umgeben uns Sensoren. Die modernen Autos verfügen über 50 - 100 verschiedene Sensoren: Abstandsensoren, Drehzahl- und Geschwindigkeitsensoren, Füllstansensor des Benzintanks, Temperatursensor usw.

MERKE DIR...

Die Anweisung *getDistance()* gibt den Wert des Ultraschalsensors zurück. Die Sensorwerte werden in einer *repeat*-Schleife periodisch gemessen, *delay(100)* bestimmt die Messperiode. Dieser Befehl ist wichtig, das sonst die Werte zu häufig abgefragt werden, was zur Überlastung des Mikroprozessors führen kann.

ZUM SELBST LÖSEN

1. Das Regelungssystem im Beispiel 2 ist noch nicht optimal. Wenn du die Hand ganz wegnimmst, reagiert der Roboter oft verwirrt, da er entweder weit entfernte Gegenstände oder gar nichts detektiert. Optimiere das Programm so, dass der Roboter im Fall, dass die Distanz grösser als 50 cm ist, stehen bleibt.
   Dazu verwendest du die if-elif-else-Struktur, mit welcher du eine mehrfache Auswahl programmieren kannst.

```
if d < 20:
   ...
elif d >= 20 and d < 50:
   ...
else:
   ...
```

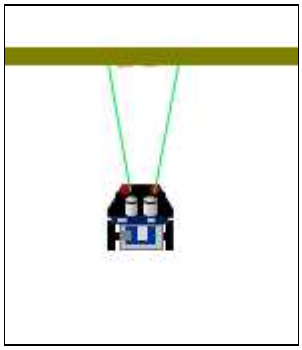
2. Ein Roboter mit einem Ultraschallsensor soll einen höherer Gegenstand (Säule aus Pappkarton, Kerze, Büchse...) finden, indem er am Ort langsam dreht und die Distanz misst. Falls er ein Objekt im Abstand kleiner als 40 cm detektiert, löst er für 4 Sekunden einen Alarm aus.

3. Ein Roboter soll gleich wie im Beispiel 2 ein Objekt finden, indem er am Ort dreht und die Werte seines Ultraschalsensors auswertet. Wenn er ein Objekt entdeckt, fährt er hinzu und bleibt im Abstand von 10 cm vor dem Objekt stehen.

Ein Roboter mit einem Ultraschallsensor versucht den Ausgang aus einem begrenzten Raum zu finden. Er dreht zuerst langsam am Ort und wenn er keine Wand detektiert, fährt er los.

ZUSATZSTOFF: ULTRASCHALLSENSOR IM SIMULATIONSMODUS

Die Distanzmessung mit einem Ultraschallsensor lässt sich auch im Simulationsmodus ausführen. Dazu musst du die die Koordinaten des Objekts (target) im Grafik-Fenster angegeben.



Beispiel 4: Ultraschallsimulation (Objekt erkennen)
Der Roboter soll zum Hindernis fahren. Wenn die Distanz kleiner als 30 cm ist, fährt er eine kurze Strecke rückwärts und danach wieder vorwärts.

Für die Simulation verwendest als Hindernis einen horizontalen Balken, den du mit folgendem Context erzeugst:

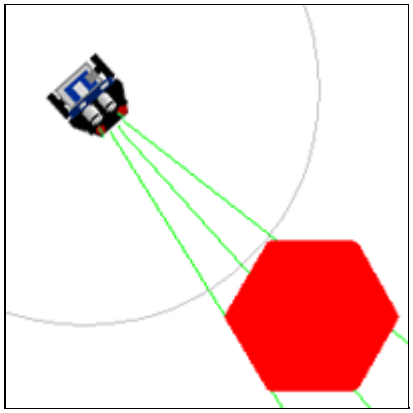
```
target = [[200, 10], [-200, 10], [-200, -10], [200, -10]]
RobotContext.useTarget("sprites/bar0.gif", target, 250, 100)
```

In der ersten Zeile sind die Koordinaten der Eckpunkte des Objekts (ein Rechteck mit dem Mittelpunkt bei (0,0)). Im Grafikfenster wird das Objekt an der Position (250, 100) mit dem Bild *bar0.gif* dargestellt.

```
from callibot import *

target = [[200, 10], [-200, 10], [-200, -10], [200, -10]]
RobotContext.useTarget("sprites/bar0.gif", target, 250, 100)

forward()
repeat:
    d = getDistance()
    if d < 30:
        backward()
        delay(2000)
    else:
        forward()
    delay(100)
```

Der Ultraschallsensor ist beim simulierten EV3 oben auf dem Brick und nicht vorne platziert. Deswegen fährt der Roboter etwas näher zum Balken hin.

Beispiel 5: Ultraschallsimulation (Objekt suchen)
Ein Roboter mit Ultraschallsensor dreht sich langsam am Ort. Wenn er einen Gegenstand detektiert, fährt er hinzu und hält kurz vor ihm an.
Du schreibst eine Funktion searchTarget(), die den Suchvorgang definiert: Der Roboter dreht in kleinen Schritten nach rechts und misst dabei die Distanz *dist*. Wenn er ein Objekt detektiert (dist != -1), dreht er ein wenig weiter und unterbricht den Suchvorgang mit break.



Danach fährt er mit Hilfe einer zweiten repeat-Schleife so lange vorwärts, bis die Distanz zum Objekt kleiner als 30 ist. Mit dem Befehl setBeamAreaColor() kannst du den Suchvorgang besser veranschaulichen.

```
from callibot import *

target = [[50,0], [25,43], [-25,43], [-50,0],[-25,-43], [25,-43]]
RobotContext.useTarget("sprites/redtarget.gif", target, 400, 400)

def searchTarget():
    repeat:
        right()
        delay(50)
        dist = getDistance()
        if dist != -1:
            right()
            delay(600)
            break

setBeamAreaColor(Color.green)
setSpeed(10)
searchTarget()
forward()

repeat:
    dist = getDistance()
    print(dist)
    if dist < 20:
        stop()
        break
```

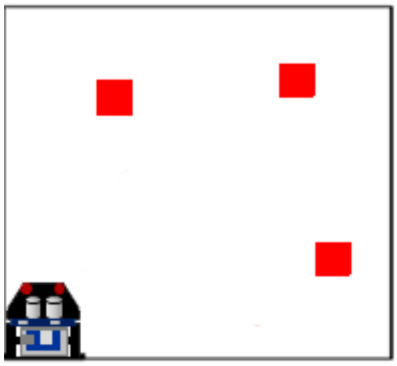
Das Objekt ist mit den Koordinaten eines Sechsecks mit dem Mittelpunkt (0, 0) beschrieben und mit dem Bild *redtarget.gif* an der Position (400, 400) dargestellt wird.

MERKE DIR...

Auch den Ultraschalsensor kannst du simulieren. Dabei musst du das Hindernis mit den Koordinaten der Eckpunkte beschreiben. Wenn der simulierte Sensor kein Objekt detektiert, gibt er den Wert -1 zurück (der reale Roboter 255). Für den Realmodus musst du die Zeilen mit *RobotContext* deaktivieren.

ZUM SELBST LÖSEN

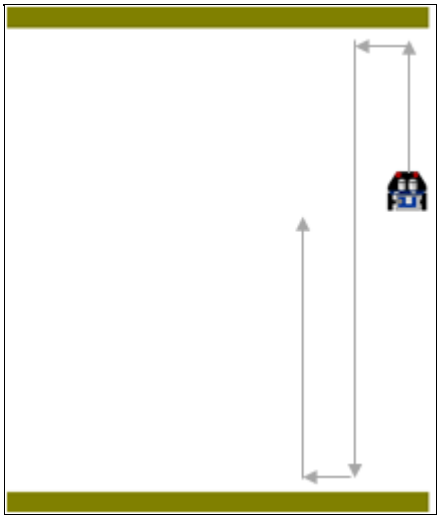
5. Ein Roboter soll mit einem Ultraschallsensor soll drei hohe leichte Gegenstände nacheinander finden und umstossen. Er dreht langsam am Ort und wenn er ein Objekt detektiert, fährt er etwas schneller auf ihn los los bis er ihn zum Fall gebracht hat. Danach fährt er eine kurze Strecke zurück und sucht er das nächste Objekt.



Im Simulationsmodus verwendest du das folgende RobotContext:

```
mesh = [[-30, -30], [-30, 30], [30, -30], [30, 30]]
RobotContext.useTarget("sprites/squaretarget.gif", mesh, 350, 250)
RobotContext.useTarget("sprites/squaretarget.gif", mesh, 100, 150)
RobotContext.useTarget("sprites/squaretarget.gif" ,mesh, 200, 450)
RobotContext.setStartPosition(40, 450)
RobotContext.setStartDirection(250)
```

6. Der Roboter soll eine rechteckige Fläche, die auf allen Seiten begrenzt ist, so abfahren, als ob er sie wie ein Rasenmäher mähen würde. Falls er mit seinem Ultraschalsensor eine Begrenzung detektiert, so dreht er 90 Grad, fährt eine kurze Strecke parallel zur Begrenzung, dreht nochmals um 90 Grad und fährt wieder vorwärts, bis er bei der gegenüberliegenden Begrenzung ankommt. Diese Bewegung wiederholt er, bis er das ganze Feld gemäht hat.



Im Simulationsmodus kannst du folgende Feld-Begrenzung verwenden

```
target = [[200, 10], [-200, 10], [-200, -10], [200, -10]]
RobotContext.useTarget("sprites/bar0.gif", target, 250, 20)
RobotContext.useTarget("sprites/bar0.gif", target, 250, 480)
RobotContext.setStartPosition(430, 480)
```

Die Aufgabe ist nicht so einfach, da der Roboter abwechslungsweise links und rechts drehen muss. Dies lässt sich mit folgendem Trick lösen: du definierst eine Funktion *turn(n)*, die für gerade n eine Drehung links und für die ungeraden n eine Drehung nach rechts bewirkt. n musst du zu Beginn auf 0 setzen.

```
def turn(n):
    if n % 2 == 0:
        left()
    else:
        right()
    delay(540))
```

5. INFRAROTSENSOREN

DU LERNST HIER...

wie der Roboter mit seinen Infrarotsensoren die Helligkeit der Unterlage erkennen kann.

WIE FUNKTIONIERT EIN INFRAROTSENSOR?

Ein Infrarotsensor besteht aus einer Leuchtdiode (LED), welche Licht im Infrarotbereich aussendet und einer Fotodiode, welche die Intensität des reflektierenden Lichtes misst.

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

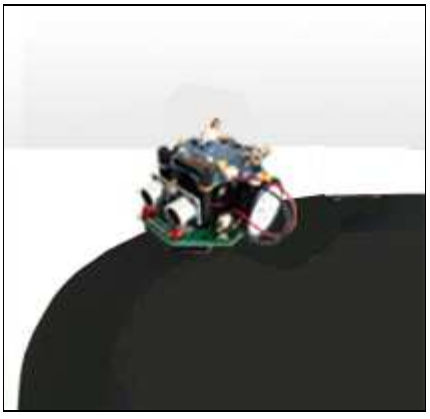
Der Callibot verfügt über **2 Infrarotsensoren**. Man findet sie auf der unteren Seite des Boards. Da das Infrarotlicht an hellen bzw. dunklen Flächen unterschiedlich reflektiert, können diese Sensoren zwischen einer hellen und dunklen Unterlage unterscheiden und geben einen digitalen Wert 0 (dunkel) oder 1 (hell) zurück.

MUSTERBEISPIELE

Beispiel 1: Kante folgen

Der Roboter soll mit seinem linken Infrarotsensor einer dunklen Fläche fahren. Je nachdem, ob er dunkel oder hell "sieht", korrigiert er die Fahrtrichtung mit einem Rechts- oder Linksbogen. Die Sensorwerte werden in einer Endlos-Schleife mit der Messperiode von 100 ms abgefragt.

Mit der Anweisung v = irLeftValue() der Sensorwert in der Variablen v gespeichert.



```
from callibot import *

setSpeed(30)
repeat:
   v = irLeftValue()
   if v == 0:
      rightArc(0.1)
   else:
      leftArc(0.1)
   delay(100)
```

Du kannst das Programm auch im Simulationsmodus ausführen. Dazu musst du nach der Zeite `from callibot import *` folgende zwei Zeilen einfügen:

```
RobotContext.useBackground("sprites/blackarea.gif")
RobotContext.setStartPosition(430, 350)
```

Die erste Zeile fügt das Hintergrundbild "blackarea.gif" hinzu, die zweite Zeile bestimmt die Position, an der der Roboter zu Beginn erscheint. (Das Grafikfenster ist 500 x 500 Pixel gross, die Koordinate (0,0) ist oben links).

Wenn du das Programm danach wieder im Realmodus ausführen willst, musst du die Zeilen mit *RobotContext* deaktivieren (# voranstellen). Du siehst bereits im Simulationsmodus, dass die Robotersteuerung vom Radius des Links- bzw. Rechtsbogen abhängig ist. Ist er zu klein (z.B. 0.05), bewegt sich der Roboter sehr langsam und unruhig. Ist er zu gross (z.B. 0.6), so verliert er oft die Spur.

Beispiel 2: Einem Weg folgen

Um einem Weg zu folgen, so wie es selbstfahrende Autos tun, braucht man mehrere Sensoren. In einer stark vereinfachten Version eines autonomen Fahrzeuges verwendest du zwei Infrarotsensoren.

Die Messwerte des rechten und des linken Sensors speicherst du in den Variablen vR und vL
vR = irRightValue()
vL =irLeftValue()

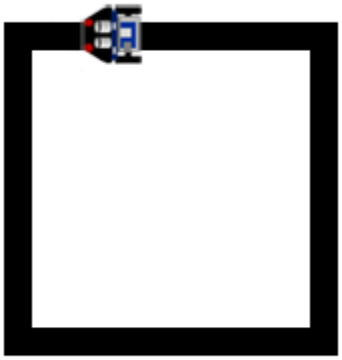
Danach entwickelst du ein Algorithmus, mit dem der Roboter den Weg möglichst genau folgen kann.

```
from callibot import *

#RobotContext.useBackground("sprites/trail.gif")

setSpeed(20)
repeat:
    vR = irRightValue()
    vL = irLeftValue()
    if vL == 0 and vR == 0:
        forward()
    elif vL == 1 and vR == 0:
        rightArc(0.1)
    elif vL == 0 and vR == 1:
        leftArc(0.1)
    delay(100)
```

Für die Simulation verwendest du das Hintergrundbild *trail.gif*. Im Realmodus musst du je nach grösse der nachgebauten Vorlage die Geschwindigkeit und den Radius der Kreisbögen anpassen.

Beispiel 3: Quadrat fahren

Im realen Modus ist es schwierig Motoren so zu steuern, dass der Roboter eine längere Zeit auf einer quadratischen Bahn bleibt. Mit Hilfe derInfrarotsensoren kann der Roboter seine Richtung selbst korrigieren. Um das Programm besser zu strukturieren, definierst du eine Funktion keepOnTrack(), die die Bewegung des Roboters auf den geraden Wegstücken steuert. Eine rechtwinklige Richtungsänderung erfolgt, wenn der Roboter mit beiden Sensoren hell "sieht". Dann dreht er 90° nach links und setzt seine Fahrt auf dem Streifen fort.



```
from callibot import *

RobotContext.useBackground("sprites/field1.gif")
RobotContext.setStartPosition(380, 400)

def keepOnTrack():
    if vL == 0 and vR == 0:
        forward()
    elif vL == 0 and vR == 1:
        leftArc(0.1)
    elif vL == 1 and vR == 0:
        rightArc(0.1)

repeat:
    vR = irRightValue()
    vL = irLeftValue()
    if vL == 1 and vR == 1:
        left()
        delay(500)
    else:
        keepOnTrack()
    delay(100)
```

Für die Simulation verwendest du das Hintergrundbild *field1.gif*.

MERKE DIR...

Mit den Inrarotsensoren kannst du die Helligkeit der Unterlage messen. Der Befehl *irLeft.read_digital()* liefert den Wert des linken Infrarotsensors, als Wert 0, falls die Unterlage dunkel oder 1, falls sie hell ist.

ZUM SELBST LÖSEN

1. Der Roboter soll sich endlos auf einem quadratischen Tisch mit einem weissem Innenkreis bewegen, ohne herunterzufallen. Dabei startet er in der Mitte und fährt geradeaus. Erkennt er den Rand, so fährt er rückwärts, dreht um ungefähr 90 Grad nach links und fährt dann wieder vorwärts.

   Für die Simulation kannst du das Hintergrundbild *circle.gif* verwenden.

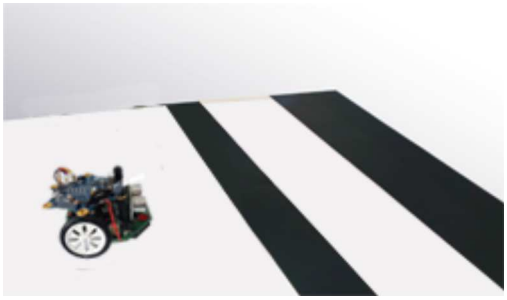
   ```
   RobotContext.useBackground("sprites/circle.gif")
   ```

2. Beim folgen der Pfade mit Kreuzungen verliert der Roboter oft die Spur, insbesondere, wenn du ihn schneller fahren lässt. Ergänze das Programm aus dem Beispiel 2 für den Fall, dass der Roboter die Spur verliert (hell-hell "sieht"). Überlege, was er in diesem Fall tun kann.

   Für die Simulation kannst du das Hintergrundbild *track.gif* verwenden:
   ```
   RobotContext.useBackground("sprites/track.gif")
   ```

3. Der Roboter soll auf weisser Unterlage starten und dann beim Erkennen des ersten schwarzen Streifens 2 Sekunden anhalten und weiterfahren (wie bei einer Haltestelle). Beim Erkennen des zweiten Streifens stoppt er definitiv (Endbahnhof).

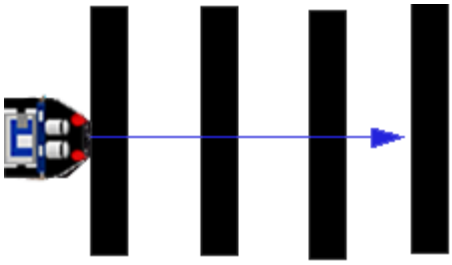
   

   Für den Simulationsmodus benötigst du folgenden Context:

   ```
   RobotContext.useBackground("sprites/blacktapes.gif")
   RobotContext.setStartPosition(10, 250)
   RobotContext.setStartDirection(0)
   ```

   Bemerkung: Die Lösung ist nicht ganz einfach, denn wenn der Roboter bei der Haltestelle anhält, weil er schwarz "sieht", so "sieht" er immer noch schwarz, auch wenn er über den Streifen weiterfährt. Du musst dir mit einer Variablen s merken, in welchem Zustand der Roboter gerade ist, z.B.
   s = 0: erstes Fahren auf weiss,
   s = 1: Haltestelle angehalten,
   s = 2: zwischen Haltestelle und Endbahnhof auf weiss.
   Falls du beim Programmieren stecken bleibst, so kannst du hier ein wenig spicken.

4. Ein Roboter soll quer über fünf dunkle Streifen fahren und diese mit seinen Infrarotsensoren detektieren. Beim Erkennen des vierten Streifens soll er anhalten.



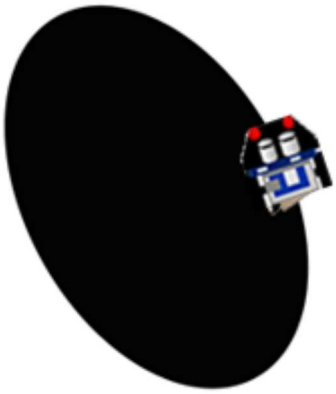
Für die Simulation verwendest du folgenden Context:

```
RobotContext.setStartPosition(0, 250)
RobotContext.setStartDirection(0)
RobotContext.useBackground("sprites/panels.gif")
```

Bemerkung: Du musst wieder einen ähnlichen Trick wie in der vorhergehenden Aufgabe finden. Du kannst z.B. die Anzahl zurückgelegten Streife in der Variable s speichern. Der Wert dieser Variable wird aber nur dann um 1 erhöht, wenn der Roboter von hell auf dunkel wechselt. Du brauchst also eine zweite Variable, in der du den Zustand hell bzw. dunkel speicherst.

5. Dein Programm soll den Roboter so steuern, dass er mit Hilfe seiner beiden Infrarotsensoren möglichst genau der Kante entlang fährt.

   Für die Simulation kannst du das Hintergrundbild *oval.gif* benutzen



```
RobotContext.useBackground("sprites/oval.gif")
RobotContext.setStartPosition(400, 400)
```

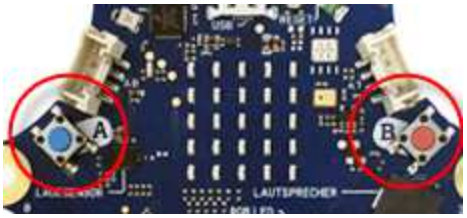
6. BUTTONS

DU LERNST HIER...

wie man die beiden Calliope-Buttons verwendet, um interaktive Programme zu entwickeln.

WIE FUNKTIONIEREN DIE BUTTONS

Der Calliope verfügt über zwei programmierbare Buttons, die es ermöglichen, interaktiv während der Programmausführung den Programmablauf zu beeinflussen. Der linke wird mit der Variablen *button_a*, der rechte mit *button_b* angesprochen.

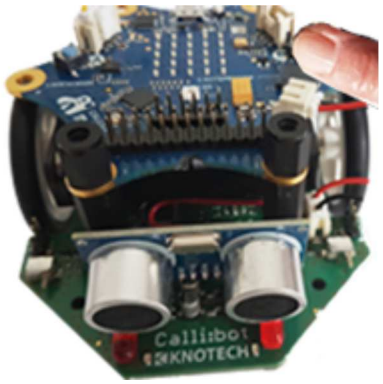


Die Funktion button_a.is_pressed() gibt *True* zurück, wenn der linke Button gedrückt ist (analog *button_b_is pressed()*).

MUSTERBEISPIELE

Beispiel 1: Auf das Drücken eines Buttons reagieren
Das Programm wartet in einer Endlosschleife, bis der Button A oder der Button B gedrückt wird. Dann beginnt die rechte bzw. linke LED mit einer Periode von 1s zu blinken und zwar so lange der Button gedrückt ist.

Der Zustand der Button wird alle 200 Millisekunden abgefragt (delay(100)). Diese Pausen sind wichtig, da sonst das System zu stark belastet wird.




```
from calliope_mini import *
from callibot import *

def blink(led):
    led(1)
    delay(500)
    led(0)
    delay(500)

repeat:
    if button_a.is_pressed():
        blink(setLEDRight)
    if button_b.is_pressed():
        blink(setLEDLeft)
    delay(100)
```

Um das Programm besser zu strukturieren, definierst du eine Funktion blink(led), die eine LED einmal ein- und ausschaltet. Der Parameter *led* kann setLEDLeft oder *setLEDRight* sein, je nachdem, ob der linke oder rechte Button gedrückt wird. Die Befehle für die Buttons sind im Modul calliope_mini, das du zusätzlich importieren musst.

Beispiel 2: Auf das Klicken eines Buttons reagieren (nur Realmodus)
Mit einem Klick auf den Button A oder B soll eine kurze Melodie abgespielt werden. Die Funktion button_a.was_pressed() gibt True zurück, wenn der *Button a* kurz gedrückt wurde. Der Befehl play(JUMP_UP) aus dem Modul music spielt eine kurze Melodie ab. Einige Melodien sind im Modul music eingebaut.

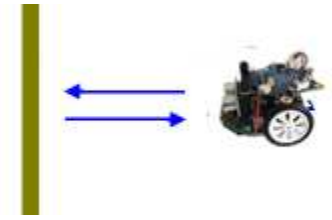
Das Modul music kann nur im Realmodus importiert werden.

```
from calliope_mini import *
from callibot import *
from music import *

repeat:
    if button_a.was_pressed():
        play(JUMP_UP)
    if button_b.was_pressed():
        play(JUMP_DOWN)
```

Beispiel 3: Ein Programm mit Button abbrechen

Bei den fahrenden Robotern ist es vorteilhaft eine Endlos-Schleife zu verwenden, die nur so lange wiederholt wird, bis ein Button gedrückt wurde.

Der Roboter fährt zum Hindernis, das er mit seinem Ultraschallsensor detektiert und wieder zurück.



Solange der Button A nicht gedrückt wurde, wiederholt der Roboter die Befehle in der while-Schleife. Mit dem Buttonklick wird die Schleife beendet und das Programm wird mit dem Befehl stop() fortgesetzt.

```
from calliope_mini import *
from callibot import *

setSpeed(30)
forward()
while not button_a.was_pressed():
    d = getDistance()
    if d < 10:
        backward()
        delay(1000)
        forward()
    delay(200)
stop()
```

Beispiel 4: Den Roboter mit Buttons steuern

Mit Klick auf die Button a bzw. b sollst du den Roboter durch das Labyrinth führen. Der Roboter fährt jeweils vorwärts, wenn er mit seinen Lichtsensoren dunkel "sieht" fährt er kurz zurück und bleibt stehen. Wenn du den Button a drückst, dreht er 90° nach links und fährt wieder vorwärts, bis zur nächsten dunklen Kante. Mit Klick auf Button b biegt er rechts ab.

Bei der Ausführung im Simulationsmodus erscheint zusätzlich ein Fenster mit einem Calliope- Bild (eventuell musst du das andere Grafikfenster verschieben). Die Buttons bedienst du mit Mausklick.

```
from calliope_mini import *
from callibot import *

RobotContext.useBackground("sprites/bg.gif")
RobotContext.setStartPosition(310, 460)

forward()
repeat:
    v = irLeft.read_digital()
    if v == 0:
        backward()
        delay(500)
        stop()
    if button_a.was_pressed():
        left()
        delay(550)
        forward()
    elif button_b.was_pressed():
        right()
        delay(550)
        forward()
    sleep(10)
```

MERKE DIR...

Mit Buttons kannst du interaktive Programme entwickeln. Die Funktion ***is_pressed()*** gibt True zurück, wenn der Button gedrückt ist. Die Funktion ***was_pressed()*** gibt True zurück, wenn seit dem Start des Programms oder seit dem letzten Aufruf irgendwann mal geklickt wurde.

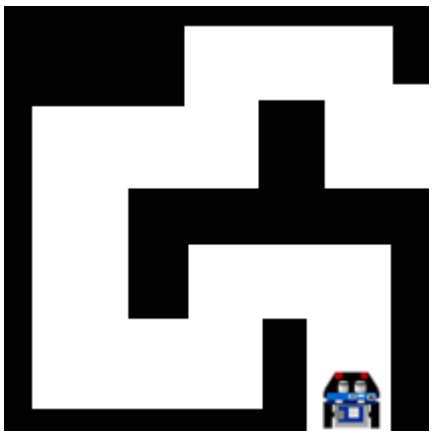
ZUM SELBST LÖSEN

1. Mit Klick auf den Button a wird der Alarm ausgelöst, mit Klick auf den Button b wird der Befehl setAlarm(0) aufgerufen und der Alarm gestoppt.

2. Mit Klick auf den Button a soll der Roboter 2000 Millisekunden vorwärts fahren und anhalten, beim Klicken des Buttons b 2000 Millisekunden rückwärtsfahren und anhalten.

3. Der Roboter soll durch die Parcours durchfahren, indem du ihm mit Klick auf den Button a bzw. Button b mitteilst, ob er links oder rechts abbiegen soll.

   Löse die Aufgabe im Simulationsmodus und verwende das Hintergrundbild *bg2.gif*



7. FERNSTEUERUNG VIA BLUETOOTH

DU LERNST HIER...

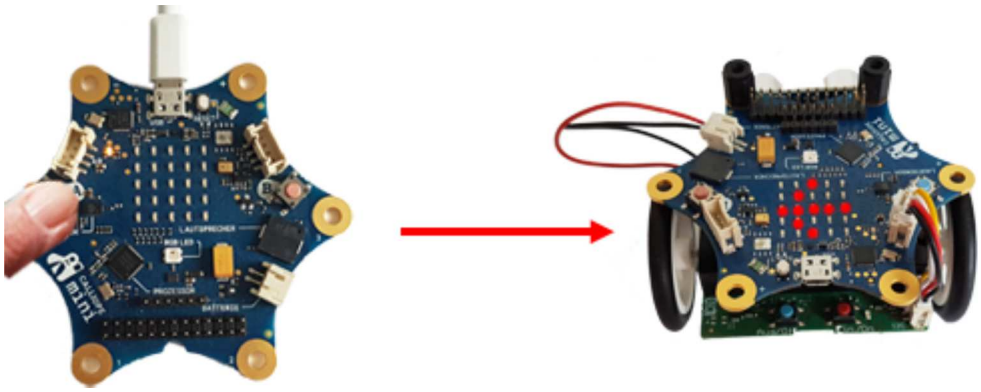
wie du eine Bluetooth-Kommunikation zwischen zwei Bluetooth-fähigen Robotern aufbauen kannst und wie du den Callibot fernsteuern kannst.

SENDEN UND EMPFANGEN

Das im TigerJython integrierte Modul radio ermöglicht die Programmierung einfacher Anwendungen, bei welchen zwei Roboter miteinander über das Bluetooth kommunizieren. Dabei kann jeder Roboter gleichzeitig ein Sender und ein Empfänger sein. Zuerst schalten beide Roboter mit radio.on() das Sende-/Empfangsgerät ein. Dabei wird eine Bluetooth-Verbindung erstellt. Nachfolgend kann jeder mit radio.send(msg) eine Message (als String) an den anderen Roboter senden, die dort in einen Empfangsbuffer gespeichert wird. Mit radio.receive() holst du die Message und der Buffer wird gelöscht.

MUSTERBEISPIELE

Beispiel 1: Button-Klick auf Calliope bewirkt eine Pfeilanzeige auf Callibot
Beim Drücken des Buttons A auf dem Calliope wird die Massage "left" geschickt. Der Callibot empfängt diese Massage und zeigt auf seinem Display ein Links-Pfeil an. Beim Drücken des Buttons B auf dem Calliope wird die Message "right" geschickt und der Callibot zeigt ein Rechts-Pfeil an. Da jeder Calliope gleichzeitig ein Sender und Empfänger ist, kannst du umgekehrt mit Drücken der Buttons auf dem Callibot, die Pfeilanzeige auf dem Calliope bewirken.



Sender Empfänger

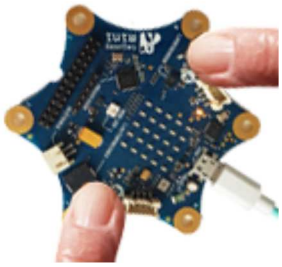
Lade das Programm zuerst auf den Callibot und schalte die Stromversorgung ein. Dann kannst du das USB-Kabel wegnehmen und es für die Übertragung und Programmausführung auf dem zweiten Calliope verwenden.

Mit radio.on() wird die Verbindung aufgebaut. Danach wartet in einer endlosen Schleife jedes Gerät, ob ein Button gedrück wird (dann sendet es eine Message), oder ob es eine Message erhält (dann führt es den entsprechenden Befehl aus).

Die Zeile display.show(Image.ARROW_E) bewirkt die Anzeige eines Linkspfeils.

```
import radio
from calliope_mini import *

radio.on()
while True:
    if button_a.was_pressed():
        radio.send("left")
    elif button_b.was_pressed():
        radio.send("right")
    rec = radio.receive()
    if rec != None:
        if rec == "left":
            display.show(Image.ARROW_E)
        elif rec == "right":
            display.show(Image.ARROW_W)
    sleep(10)
```

Beispiel 2: Callibot mit einem zweiten Calliope steuern



Da du den fahrenden Callibot-Roboter steuern willst, verwendest du in diesem Beispiel den Calliope als Sender und den Callibot als Empfänger. Da der Sender nicht alle Callibot-Befehle braucht und umgekehrt, ist es sinnvoll, zwei verschiedene Programme zu verwenden.

Die Steuerung erfolgt wie folgt:

- Button A gedrückt: links fahren (Zustand LEFT)
- Button B gedrückt: rechts fahren (Zustand RIGHT)
- Beide Buttons gedrückt: geradeaus fahren (Zustand FORWARD)
- Kein Button gedrückt: anhalten (Zustand STOP)

Die sogenannte Zustandsprogrammierung ist in diesem Beispiel sehr sinnvoll. Es hat keinen Sinn alle 10 Millisekunden eine Message "FORWARD" zu senden, wenn der andere Roboter bereits geradeaus fährt. Eine Massage wird nur dann geschickt, wenn der Zustand geändert hat (oldState != state).

Programm für den Calliope:

```
import radio
from calliope_mini import *

radio.on()
display.show(Image.YES)
```

```
state = "STOP"
oldState = ""
while True:
    if button_a.is_pressed() and button_b.is_pressed():
        state = "FORWARD"
    elif button_a.is_pressed():
        state = "LEFT"
    elif button_b.is_pressed():
        state = "RIGHT"
    else:
        state = "STOP"
    if oldState != state:
        radio.send(state)
        oldState = state
    sleep(10)
```

Auf dem Callibot wird nach dem Programmstart ein "Ok"-Zeichen angezeigt, damit du weisst, dass er bereits ist. Danach wartet er auf die Befehle, die er via Bluetooth empfängt. Falls er Message "FORWARD" erhält, führt er den Befehl forward() aus und bleibt in diesem Zustand, bis er eine neue Message erhält. Mit "LEFT" bewegt er sich auf einem Linksbogen, mit "STOP" hält er an.

Programm für den Callibot:

```
from callibot import *
from calliope_mini import *
import radio

radio.on()
display.show(Image.YES)
while True:
    rec = radio.receive()
    if rec == "FORWARD":
        forward()
    elif rec == "LEFT":
        leftArc(0.2)
    elif rec == "RIGHT":
        rightArc(0.2)
    elif rec == "STOP":
        stop()
    sleep(10)
```

Damit du dich mit dem Steuergerät frei bewegen kannst, kannst du nach dem Programmdownload für Stromversorgung des Calliope eine Powerbank oder eine Andere Smartphone-Ladequelle verwenden.

MERKE DIR...

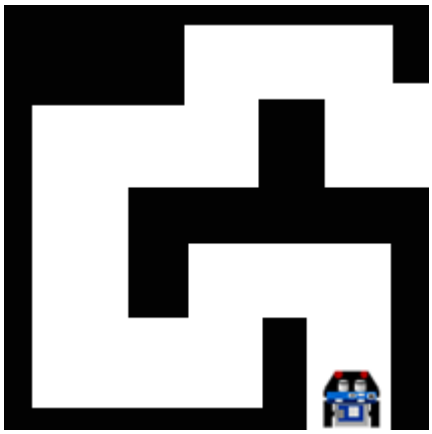
Damit du die Befehle für die Bluetooth-Kommunikation verwenden kannst, musst du das Modul *radio* importieren. Mit *radio.on()* wird die Verbindung aufgebaut. Mit radio.send(msg) wird eine Message verschickt, mit radio.receive() wird sie am anderen Gerät empfangen und in einem Empfangsbuffer so lange gespeichert, bis man sie abgeholt hat.

ZUM SELBST LÖSEN

1. Mit Klick auf den Button A auf dem Calliope löst auf dem Callibot ein Alarm aus. Mit Klick auf den Button B wird der Alarm gestoppt.

2. Mit Klick auf den Button A wird der Roboter in die Vorwärtsbewegung versetzt und auf dem Display erscheint ein Vorwärts-Pfeil, mit Klick auf den Button B hält der Roboter an und auf dem Display erscheint das Image NO. Damit du siehst, ob die beide Geräte bereits sind, soll auf beiden nach dem Start das Image YES angezeigt werden.

3. Mit Hilfe eines zweiten Calliope sollst du den Callibot durch einen Parcours steuern, indem du ihm mit Klick auf die Buttons A und B mitteilst, ob er geradeaus fahren oder links bzw. rechts abbiegen soll.



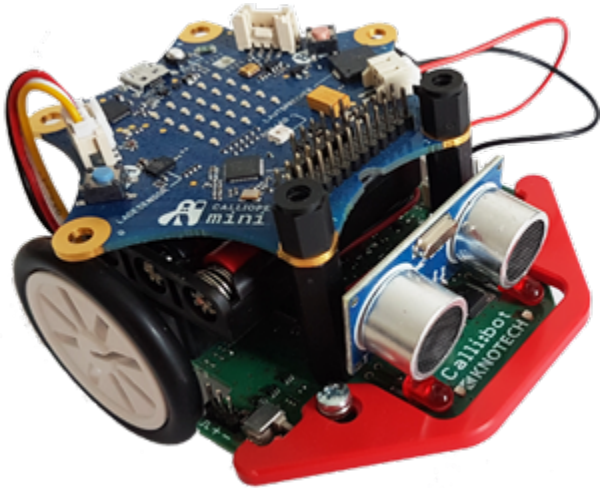
8. TOUCHSENSOREN (BERÜHRUNGSSENSOREN)

DU LERNST HIER...

wie du Touchsensoren dafür einsetzen kannst, Hindernisse zu erkennen und entsprechend zu reagieren.

WIE FUNKTIONIERT EIN TOUCHSENSOR

Der Touchsensor ist wie ein Schalter aufgebaut, der beim Drücken der kleinen Metallstifts einen Kontakt macht. Der Touchsensor kennt zwei Zustände *gedrückt* (1) und *nicht gedrückt* (0). Der Callibot verfügt über zwei Touchsensoren.

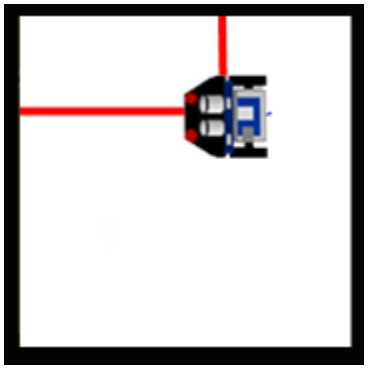


Um die Berührungen mit den Gegenständen zu detektieren, ist es praktisch an der Front des Callibots eine Stossstange zu befestigen. Wenn der Roboter frontal ein Objekt berührt, werden beide Sensoren gedrückt. Wenn er seitlich ankommt, wird nur der linke- bzw. rechte Sensor gedrückt.

Die Stossstange kannst du bei knotech.de bestellen.

MUSTERBEISPIELE

Beispiel 1: Ein Hindernis registrieren und ausweichen
Der Roboter soll sich in einem mit vier Wänden beschränken Raum bewegen. Wenn er mit dem Touchsensor eine Wand berührt, fährt er eine kurze Strecke zurück, dreht 90° nach links und fährt in der neuen Richtung weiter.



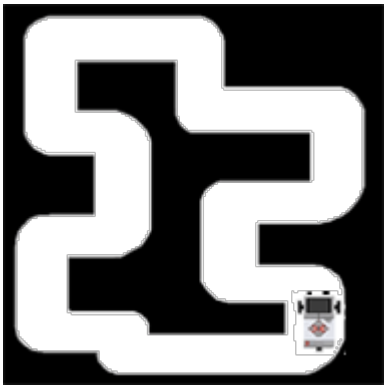
Wenn der Touchsensor frontal eine Wand berührt, werden beide Sensoren gedrückt und der Befehl tsValue() gibt 1 zurück. Die Sensorwerte werden alle 100 ms in einer repeat-Schleife abgefragt.

```
from callibot import *
#RobotContext.useObstacle("sprites/field1.gif", 250, 250)

forward()
repeat:
    ts = tsValue()
    if ts == 1:
        backward()
        delay(1500)
        left()
        delay(550)
        forward()
    delay(100)
```

Du kannst daa Programm auch im **Simulationsmodus** ausführen. Dazu musst du die Zeile *RobotContext useObstacle("sprites/field1.gif, 250, 250)* aktivieren. Das Hintergrundbild *field1.gif* hat einen transparenten Hintergrund und wird mit der Angabe der Koordinaten im Grafikfenster poitioniert.

Beispiel 2: Kanalroboter

Ein Roboter mit zwei Touchsensoren soll den Weg im Kanal finden. Mit zwei Touchsensoren kannst du den Roboter effizienter steuern. Je nachdem, ob er ein Hindernis mit dem linken oder mit dem rechten Sensor berührt, fährt er zuerst kurz zurück und dreht um einen kleinen Winkel in die Gegenrichtung. Die Berührung des linken oder rechten Sensors überprüfst du mit den Befehlen tsLLeftValue() bzw. tsRightValue(). Diese gehen 1 zurück, falls der Sensor berührt wurde, sonst 0.
Baue eine Hindernisbahn und teste das folgende Programm.



```
from callibot import *
#RobotContext.useObstacle("sprites/racetrack.gif", 250, 250)
#RobotContext.setStartPosition(420, 460)

setSpeed(30)
forward()
repeat:
    tsL = tsLeftValue()
    tsR = tsRightValue()
    if tsL == 1:
        backward()
        delay(250)
        right()
        delay(200)
        forward()
    elif tsR == 1:
        backward()
        delay(250)
        left()
        delay(200)
        forward()
    delay(100)
```

Für die Simulation aktivierst du die beiden RobotContext-Zeilen. Das Bild racetrack.gif ist in der TigerJython Bildbibliothek vorhanden.

MERKE DIR...

Die Touchsensoren können nur zwei Werte zurückgeben und zwar 1, falls der Sensor gedrückt wurde, sonst 0. Zum Abfragen der Sensorwerte stehen dir drei Befehle zur Verfügung:

- tsValue() verwendest du, wenn die Stossstange frontal berührt wird
- tsLeftValue() verwendest du, um die Berührung des linken Sensors abzufangen
- tsRightValue() gibt 1 zurück, wenn der rechte Sensor berührt wurde, sonst 0

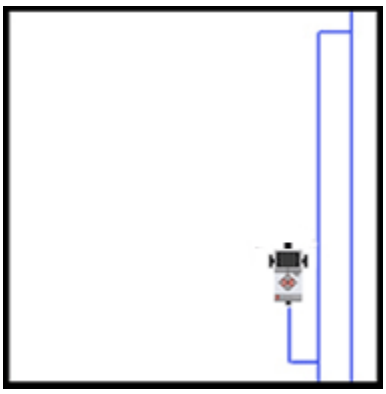
ZUM SELBST LÖSEN

1. Ein Roboter mit einem oder zwei Tousensoren bewegt sich im Käfig mit einem Ausgang. Er startet in der Mitte und soll möglichst rasch den Ausgang finden.

```
RobotContext.useObstacle("sprites/trap.gif", 250, 250)
```

2. Der Roboter soll eine rechteckige Fläche, die auf allen Seiten begrenzt ist, so abfahren, als ob er sie wie ein Rasenmäher mähen würde. Falls er auf eine Begrenzung trifft, so fährt er zuerst ein wenig zurück, dreht um 90 Grad, fährt eine kurze Strecke parallel zur Wand, dreht nochmals um 90 Grad und fährt wieder vorwärts.

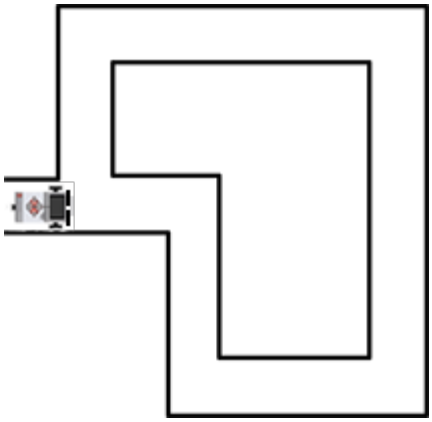
   Beachte, dass der Rasenmäher immer einmal links und einmal rechts drehen muss. Um es so zu programmieren, gibt es ein einfacher Trick:



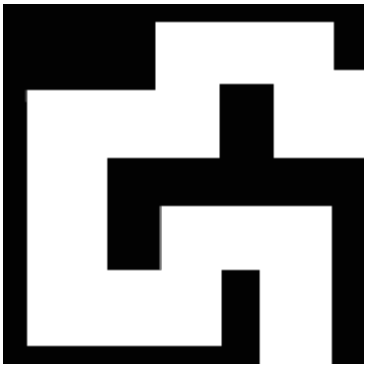
   Du wählst Variable n, die zu Beginn 0 ist, nach einer Linksdrehung auf 1 und bei einer Rechtsdrehung wieder auf 0 gesetzt wird. Du überprüfst bei jeder Wende, ob n 0 oder 1 ist und machst die entsprechende Drehung. Um das Programm besser zu strukturieren, kannst du die Links-Wende als Funktion *turnLeft()* und die Rechts-Wende als Funktion *turnRight()* definieren. Falls du nicht weiter kommst, kannst du hier spicken:

Kanalroboter.

3. Ein Roboter mit zwei Touchsensoren soll das ganze Rohr durchfahren und wieder zum Ausgangsort zurückkehren.



4. Dein Roboter soll das nebenstehende Labyrinth mit Hilfe seines Touchsensors und mit Hilfe der Buttons *button_a* und *button_b* durchfahren. Er startet unten rechts und fährt vorwärts. Wenn er mit dem Touchsensor eine Wand berührt, fährt er eine kurze Strecke rückwärts, bleibt stehen und wartet, bis du den Button left oder right gedrückt hast. Dann dreht er rechtwinklig in dieser Richtung und fährt vorwärts.



9. CO_2 - SENSOR

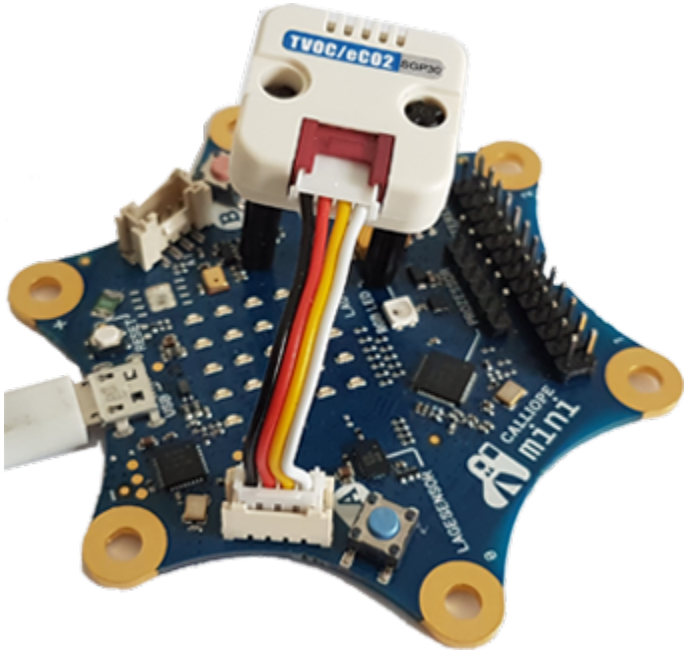
DU LERNST HIER...

mit einem CO_2-Sensor die CO_2-Konzentration im Raum messen und überwachen. Der CO_2-Wert ist ein zuverlässiger Indikator für die Luftqualität. Frische Raumluft ist heute besonders wichtig, um das Ansteckungsrisiko mit Covid-19 zu reduzieren.

SGP30 AIR QUALITY SENSOR

Der CO_2-Sensor **SGP30** liefert hochpräzise Messwerte der CO_2-Konzentration in der Luft. Das Modul *sgp_mini*, welches die Kommunikation mit diesem Sensor unterstützt, ist im TigerJython (ab Version 2.22 [Sep-18-2021]) integriert. Nach der Installation einer neuen Version musst du den Calliope neu flashen.

Der Gas Sensor TVOC/eCO2 mit integriertem SGP30 kann z.B. bei www.brack.ch für **Fr. 12.60** oder bei www.mouser.ch für **Fr. 10.25** bestellt werden.



Mit wenig Aufwand kannst du eine einfache Messstation für die Messung der CO_2-Konzentration in deinem Klassenzimmer einrichten.

Am Calliope wird der Sensor mit dem mitgelieferten I^2C-Kabel an der I^2C- Schnittstelle (rechts vom der USB-Büchse) angeschlossen.

Beim Callibot wird diese Schnittstelle auch für die Motorsteuerung verwendet. Da du die Motoren bei der CO2-Messung nicht brauchst, kannst du den Motoranschluss vorübergehend wegnehmen.

MUSTERBEISPIELE

Beispiel 1: CO_2 - Konzentration messen und anzeigen

In deinem Programm importierst du das Modul

sgp_mini, in welchen das Abfragen der Messwerte des SGP30-Sensors implementiert ist. Der Befehl sgp_mini.getValues() gibt ein Tupel mit zwei Werten zurück:

- CO_2-Konzentration in ppm*
- TVOC Total Volantile Organic Componds*

Die beiden Werte werden mit *print*-Befehl im Terminal-Fenster ausgeschrieben. sleep(500) gibt die Messperiode an.

```
CO2 =  449   TVOC =  0
CO2 =  495   TVOC =  26
CO2 =  533   TVOC =  29
CO2 =  567   TVOC =  54
CO2 =  680   TVOC =  76
CO2 =  764   TVOC =  92
CO2 =  680   TVOC =  76
CO2 =  536   TVOC =  40
CO2 =  581   TVOC =  17
CO2 =  478   TVOC =  6
```

Programm:

```
from calliope_mini import *
import sgp_mini

while True:
    co2, voc = sgp_mini.getValues()
    print ("CO2 = ", co2, "  TVOC = ", voc)
    sleep(500)
```

► In Zwischenablage kopieren

Nach dem Programmstart wird der Sensor zuerst Kalibriert und gibt die ersten 20 Sekunden den CO_2-Wert 400 zurück. Danach werden die gemessenen CO_2-Werte korrekt angezeigt.

* Der CO_2-Gehalt in der Luft wird in parts per million, kurz ppm angegeben. SGP30-Sensor gibt die Werte im Bereich 400 - 60000 ppm zurück, wobei bei Werten grösser als 1000, wird die Luft nicht mehr als "frisch" bezeichnet.

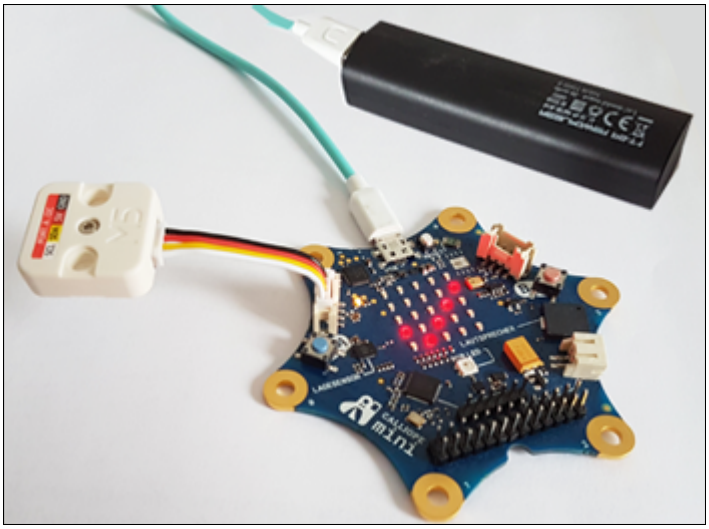
* In den Innenräumen gibt es viele Quallen, die Schadstoffe abgeben (Lampen, Bodenbeläge, Reinigungsmittel...). Je nach ihrer Konzentration gibt der Sensor TVOC-Werte im Bereich 0 bis 60 000 zurück. Für den primären Zweck, das Ansteckungsrisiko mit Covid-19 zu reduzieren, hat dieser Messwert keine grosse Bedeutung.

Beispiel 2: Ein Messgerät für CO2 Konzentration im Klassenzimmer

Grenzwerte für die Messung der CO2 Konzentration:

- < 1000 ppm: Luft ist frisch
- 1000 - 1400 ppm: bald lüften
- > 1400 ppm: Fenster öffnen

Mit den LEDs auf deinem Calliope kannst für diese Messbereiche verschiedene Symbole anzeigen. Wenn der Grenzwert von 1400 ppm überschritten wird, wird ein Warnsignal abgespielt.



Das Programm bleibt auf dem Calliope gespeichert. Du kannst ihn also beim Computer ausstecken und an eine andere Stromquelle, beispielsweise Powerbank, anschliessen.

Programm:

```
from calliope_mini import *
import sgp_mini
from music import pitch

while True:
    co2, voc = sgp_mini.getValues()
    print ("Co2 = ", co2)
    if co2 < 1000:
        display.show(Image.YES)
    elif co2 < 1400:
        display.show(Image.ARROW_S)
    else:
        display.show(Image.NO)
        pitch(800, 500)
    sleep(500)
```

► In Zwischenablage kopieren

MERKE DIR...

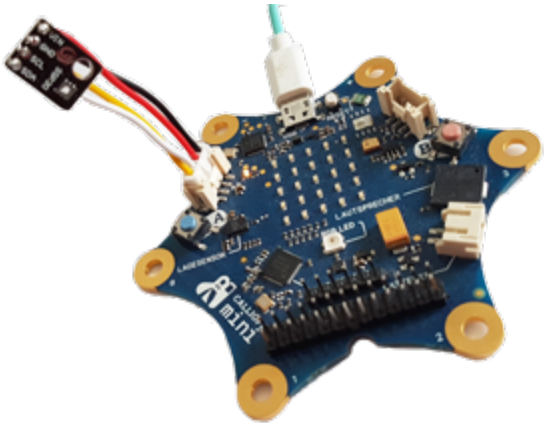
Der Sensor misst den CO_2-Gehalt in ppm (parts per million) und liefert Messwerte im Bereich 400-60 000. Für Werte < 1000 ist die Luft gut, bei Werten > 1400 ist eine Frischluftzufuhr unbedingt empfohlen. Eine hohe CO_2-Konzentration im Raum erhöht das Ansteckungsrisiko mit dem Corona-Virus.

ZUM SELBST LÖSEN

1. Messe die CO_2-Werte mit der Messperiode von 2 Sekunden und schreibe die Ergebnisse mit Laufschrift auf dem Calliope-Display.

ZUSATZAUFGABE: CO2-SENSOR SELBST ZUSAMMENLÖTEN

Falls du Freude an der Elektronik hast, kannst du einen CO2- Sensor verwenden, bei dem alle Elektronik-Komponenten sichtbar sind. Der GY-SPG30 Air Quality Sensor (erhältlich z.B. bei eckstein-shop.de) ist mit dem oben verwendeten SPG30-Sensor kompatibel und kann mir dem gleichen Modul *sgp_mini* programmiert werden.



Der Sensor wird mit einem I^2C-Kabel angeschlossen, welches am Sensor angelötet werden muss.

Du nimmst ein Grove-Kabel mit einem I^2C-Stecker, isolierst die viel dünne Kabel ab und lötest das schwarze Kabel bei GND, das rote bei VCC, das gelbe bei SCL und das weisse bei SDA an.

rot
schwarz
gelb
weiss

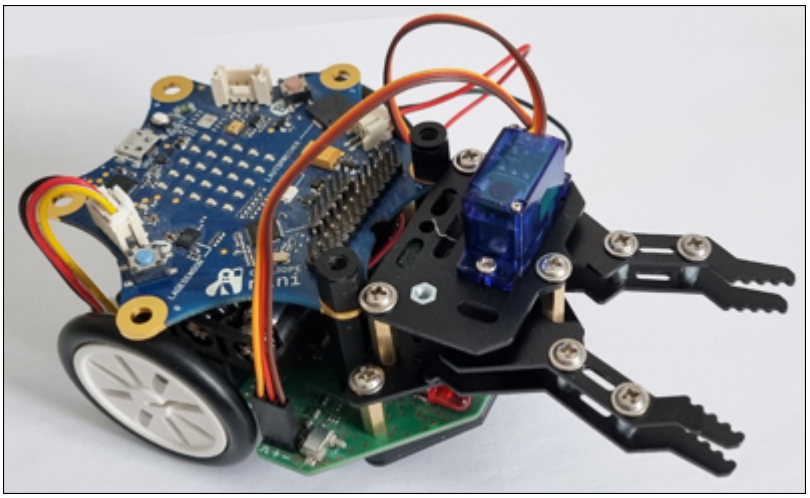
Am Calliope wird der Sensor an der I^2C-Schnittstelle (rechts vom der USB-Büchse) angeschlossen.

10. MAQUEEN MECHANIC (Servo Motoren)

DU LERNST HIER...

mechanische Bauelemente wie Bagger oder Greifarm mit Servomotoren steuern.

WAS IST MAQUEEN MECHANIC?



Mechanic Beatle

Maqueen Mechanic ist ein Bausatz, der aus Bauteilen besteht, welche mit Servo-Motoren gesteuert werden: z. Bsp. Loader, der kleine Objekte aufladen kann, Beatle, der mit einem Greifarm Objekte fasst. Maqueen Mechanic wurde für die mbRoboter entwickelt, kann aber auch mit Callibot verwendet werden.

Mit dieser Erweiterung kannst du interessante Anwendungen programmieren.

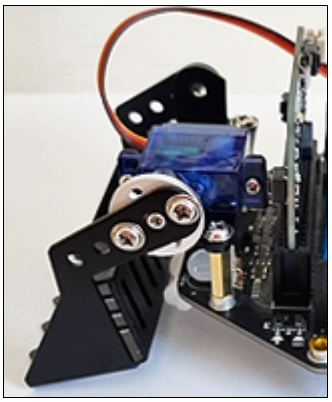
Die benötigten Bauelemente und Servomotoren können einzeln (Maqueen Mechanics Loader, Maqueen Mechanic Beatle) oder in einem Bausatz (Maqueen Mechanics Roboter-Kit) bestellt werden. Angeschlossen werden die Servomotoren an den **Ports S1** oder **S2**, die sich links und rechts auf dem Callibot befinden (braun auf "**-**").

MUSTERBEISPIELE

Beispiel 1: Loader mit Servomotor bewegen
Der Servomotor bewegt den Loader zuerst zur Ausgangslage (unten), dann wird die Schaufel nach oben bewegt und nach 2000 Millisekunden wieder nach unten in die Ausgangslage.

Der Servomotor wird mit dem Befehl setServer("S1", 160) bewegt. Führe das unten stehende Programm mit verschiedenen Winkeln aus um herauszufinden, für welchen Wert sich die Schaufel in der Ausgangslage (unten) befindet. Die Winkel sind von der Position, in der der Servomotor vor dem Anschrauben der Schaufel war, abhängig.

```
from callibot import *

setServo("S1", 160)
delay(2000)
setServo("S1", 100)
delay(2000)
setServo("S1", 160)
```



Beispiel 2: Mit Ultraschallsensor den Loader positionieren
Der Roboter soll einen Gegenstand, der sich bei einer senkrechten Wand befindet aufladen und wegtransportieren. Um den Abstand zur Wand zu messen, verwendet er den Ultraschallsensor.

```
from callibot import *

setServo("S1", 160)
setSpeed(20)
forward()
repeat:
    d = getDistance()
    print(d)
    if d < 12:
        stop()
        delay(1000)
        setServo("S1", 100)
        setSpeed(20)
        backward()
        delay(4000)
        stop()
        setServo("S1", 160)
    delay(100)
```

Beispiel 3: Mit Mechanic Beatle (Greifarm) Objekte fassen
Der Servomotor bewegt die beiden Arme des Greifarms gleichzeitig. Mit dem Befehl
setServo("S1", 120)
in die offene Stellung, mit
setServo("S1", 160)
kann der Roboter Gegenstände packen. Die Winkel muss du anpassen. Es kommt darauf an, in welcher Lage der Servomotor eingebaut ist.

Der Roboter bewegt sich zum Gegenstand, der sich in einem kurzen Abstand vor ihm befindet, packt ihn mit seinem Greifarm, fährt eine kurze Strecke zurück, dreht nach links und stellt den Gegenstand am neuen Ort ab (ähnlich wie im unten stehenden Video).

```
from callibot import *

setServo("S1", 170)
delay(1000)
setServo("S1", 120)
forward()
```

```
delay(1000)
stop()
setServo("S1", 160)
backward()
delay(500)
left()
delay(700)
stop()
setServo("S1", 120)
```

Beispiel 4: Mit Ultraschallsensor den Greifarm positionieren
Der Roboter soll einen Gegenstand, der sich bei einer senkrechten Wand befindet aufladen und wegtransportieren. Um den Abstand zur Wand zu messen, verwendet er den Ultraschallsensor.

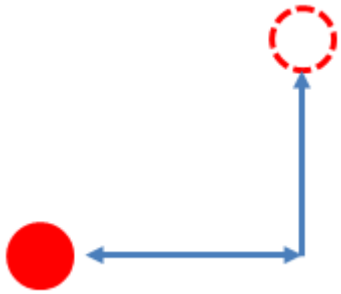
```
from callibot import *

setServo("S1", 120)
forward()
repeat:
    d = getDistance()
    print(d)
    if d < 20:
        stop()
        delay(500)
        setServo("S1", 160)
        delay(500)
        backward()
        delay(2500)
        left()
        delay(700)
        stop()
        setServo("S1", 120)
    delay(100)
```

MERKE DIR...

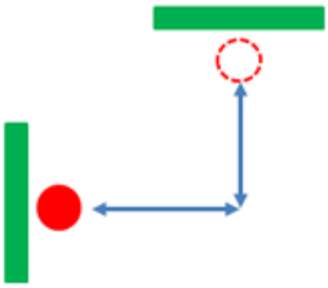
Der Servomotor wird mit dem Befehl *setServo(Port, Winkel)* an die gewünschte Position bewegt, wobei der erste Parameter S1 oder S2 ist und der Winkel im Bereich 0° bis 180° gewählt werden kann.

ZUM SELBST LÖSEN

1. Roboter mit einem Loader oder einem Greifarm bewegt sich 2000 Millisekunden vorwärts und lägt einen Gegenstand auf. Dann bewegt er sich 2000 Millisekunden Rückwärts, dreht ca. 90° nach links, bewegt sich 2000 Millisekunden vorwärts und stellt den Gegenstand ab.



2. Ergänze die Situation aus der Aufgabe 1 mit zwei senkrechten Wänden, so dass der Roboter die Vorwärtsbewegungen jeweils mit Hilfe vom Distanzsensor beendet und den Aufladen- und Abladen-Vorgang mehrmals wiederholt.



3. Der Ultraschallsensor dreht sich mit Hilfe vom Servomotor in kleinen Schritten so lange, bis er ein Objekt erkennt. Danach löst er Alarm aus. Du kannst die Suche einschränken, indem du nur Objekte in einer Distanz zwischen 30 und 70 cm suchen lässt.

4. Mit Hilfe eines Servomotors dreht der Ultraschallsensor an eine Position parallel zur Fahrtrichtung, so dass er seitwärts den Abstand messen kann. Der Roboter bewegt sich danach Vorwärts etwa im Abstand von 15 cm entlang einer geraden Wand und korrigiert laufend seine Fahrtrichtung.

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.