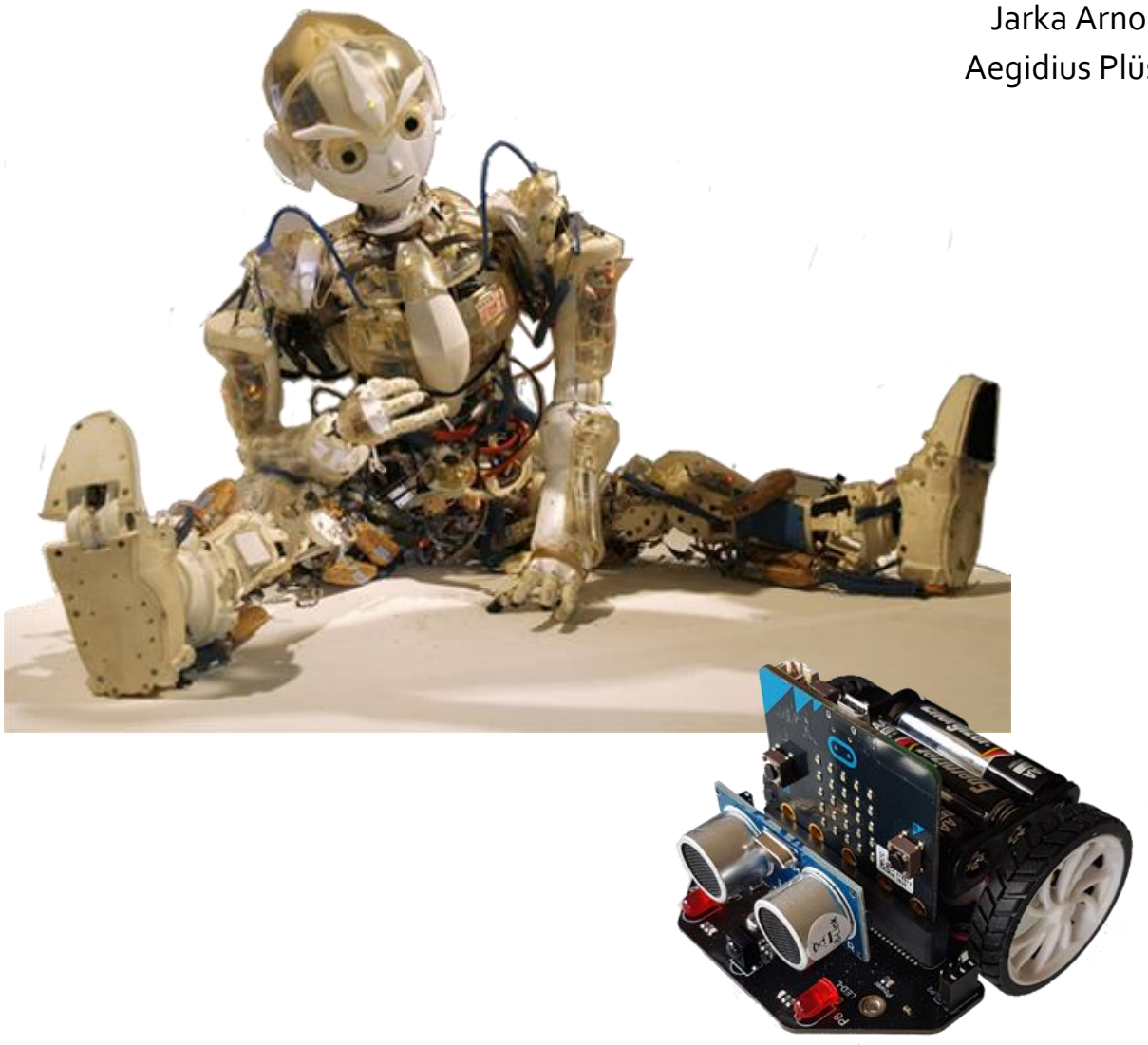
TigerJython4Kids

www.tigerjython4kids.ch

Teil 2: ROBOTIK

Jarka Arnold
Aegidius Plüss



mit TigerJython und mbRobot

www.tigerjython4kids.ch

INHALT

Teil 2: ROBOTIK

Was ist ein Roboter? ... 3
MbRobot (Maqueen) ... 4
Einrichten ... 5
Loslegen ... 7
Roboter steuern ... 9
Befehle wiederholen ... 13
Ultraschallsensor ... 16
Infrarotsensor ... 22
Buttons ... 27
Roboter per WLAN steuern ... 31
Internet of Things (IoT) ... 37
CO2 Sensor ... 42
Bagger mit Servomotor (Mechanic Loader) ... 45
Greifarm mit Servomotor (Mechanic Beatle) ... 47

ANHANG:

Dokumentation micro:bit und mbRobot ... 51
Über die Autoren / Kontakt ... 59

Version 2.0, März 2024

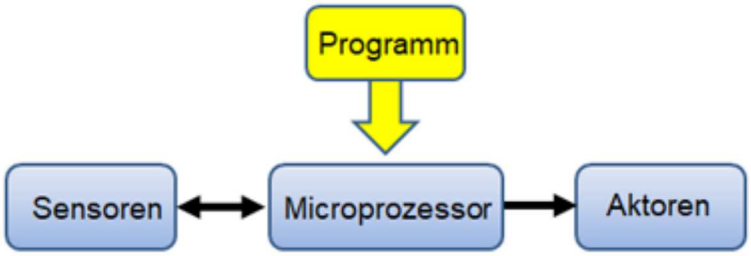
Autoren: Jarka Arnold, Aegidius Plüss
Kontakt: help@tigerjython.com

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.



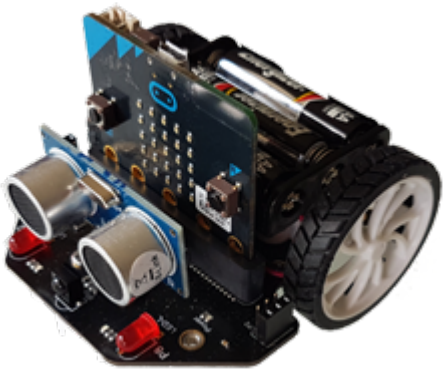
Sensoren sind Messgeräte, welche physikalischen Grössen, wie Helligkeit, Distanz, Temperatur, Luftfeuchtigkeit usw. messen und an den Mikroprozessor weiterleiten.

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.

MBROBOT, MAQUEEN PLUS und MAQUEEN PLUS V2

Der micro:bit ist ein programmierbarer Computer. Er besteht aus einer 4 x 5 cm grossen Platine mit einem 32-bit Microcontroller, Flashspeicher, 25 roten LEDs, zwei Buttons, einigen Sensoren und einer USB-Schnittstelle. Zusammen mit dem einfachen Maqueen-Fahrwerk, wird er zu einem fahrenden Roboter und bietet eine preisgünstige Alternative zum LEGO EV3 Roboter.



Mit dem Standardbausatz kann man die wichtigsten Programmierkonzepte der Robotik aufzeigen.

Für weiterführende Anwendungen in den Bereichen "Vernetzte Roboter" und "IoT" (Internet of Things) kann der mbRobot mit einem WLAN-fähigen Microcontroller (ESP 32) ergänzt werden. Die notwendigen Tools sind im TigerJython integriert.

MAQUEEN PLUS und MAQUEEN PLUS V2

Alle Programme in diesem Lehrgang können auch mit **Maqueen Plus** und mit dem neuesten Modell **Maqueen Plus V2** ausgeführt werden. Diese verfügen über mehrfarbigen LEDs und die Möglichkeit Servomotoren und andere Zusatzgeräte einfach anzuschliessen. Leider sind die beiden Modelle und der Maqueen-Roboter hardwaremässeg nicht kompatibel. Unsere Bibliotheken ermöglichen es aber, alle drei Modelle mit den gleichen Programmen zu steuern.

Die Programme unterscheiden sich nur in der Import-Zeile:
Für mbRobot: **from mbrobot import ***
Für Maqueen Plus: **from mbrobot_plus import ***
Für Maqueen Plus V2: **from mbrobot_plusV2 import ***

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.

EINRICHTEN

ZUSAMMENBAU

Der **Maqueen Bausatz** enthält eine 8 x 8.5 cm grosse Platine, an der bereits Motoren, LEDs, Infrarot- und Ultraschalsensoren und weitere Komponenten eingebaut sind.

Zusätzlich muss man einen **micro:bit** und drei AA Batterien beschaffen. Wir empfehlen, anstelle der runden Batterien, eine flache **Lipo-Akku**-Batterie (3.7 V, 1.5 Ah) zu beschaffen und diese mit einem doppelseitigen Klebband zu befestigen. Vorteilhaft ist, micro:bit V2 zu verwenden, da dieser mehr Speicherplatz hat.

Erhältlich sind auch runde **CR123A Li-ion Akku**-Batterien, die direkt mit einem USB-Kabel aufgeladen werden können.

Maqueen Plus-Bausatz enthält zusätzlich einige Metalelemente, an die man zusätzliche Bauteile befestigen kann und eine Befestigung für eine Akku-Batterie (Li-ion 18650). Der Zusammenbau nach der Anleitung, die mitgeliefert wird, ist einfach.

Maqueen Plus V2 wird weitgehend zusammengebaut ausgeliefert.
Er ist in zwei Varianten erhältlich:

- mit einer Befestigung für vier AA Batterien
- mit einer Befestigung für eine aufladbare Akku-Batterie (Li-ion 18650)

Eine aufladbare Batterie ist vorteilhaft. Zusätzlich muss ein micro:bit V2 beschafft werden.

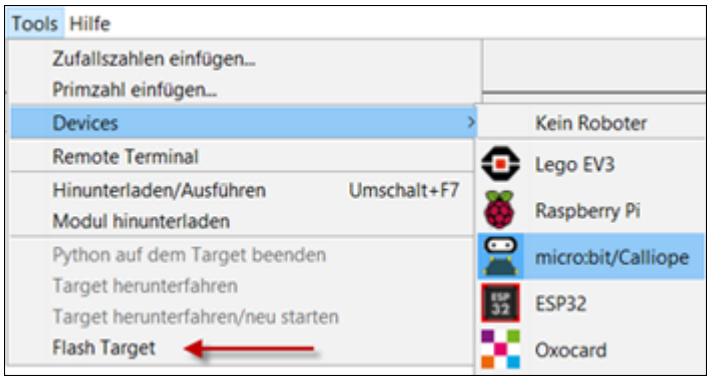
FIRMWARE INSTALLIEREN

Die Programme werden mit TigerJython entwickelt, via USB-Kabel auf den micro:bit des Roboters hinunterladen und dort mit Python ausgeführt. Vor der ersten Verwendung muss du

auf dem micro:bit eine ***Firmware*** installieren.

Schliesse den micro:bit über ein USB-Kabel am Computer an.

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



TigerJython unterstützt auch die Modelle *Maqueen Plus* und *Maqueen PlusV2*. Praktisch alle Musterbeispiele und Aufgaben in diesem Lehrgang können auch mit diesen Robotern ausgeführt werden. Leider ist die Hardware der drei Maqueen-Roboter unterschiedlich und es sind daher drei verschiedene Bibliotheken (Module) erforderlich.

Das Modul *mbrobot* wird beim Flashen vom micro:bit automatisch auf den micro:bit kopiert. Die Module *mbrobot_plus* bzw. *mbrobot_plusV2* musst du zusätzlich auf den micro:bit herunterladen. Dabei gehst du wie folgt vor:

- Verwende *micro:bit V2*, da dieser mehr Speicherplatz als *micro:bit V1* hat.
- Lade die passende zip-Datei mit folgendem Link herunter:
  https://tigerjython4ids.ch/download/ mbrobot_plus.zip
  https://tigerjython4kids.ch/download/mbrobot_plusV2.zip.
- Packe die heruntergeladenen Datei *mbrobot_plus.zip* bzw. *mbrobot_plusV2.zip* aus.
- Starte TigerJython und schliesse den micro:bit deines Roboters mit dem USB-Kabel am Computer an (Firmware muss bereits installiert sein)
- Öffne die Datei ***mbrobot_plus.py*** bzw. ***mbrobot_plusV2.py*** im TigerJython-Editor
- Wähle "Tools/Modul hinunterladen" und dann "Editor"
- Warte bis im TigerJython-Ausgabefenster der Download bestätigt wird
- Teste die Installation mit dem folgenden Programm:

```
from mbrobot_plus import *
#from mbrobot_plusV2 import *
forward()
delay(2000)
stop()
```

Der Roboter soll 2000 Millisekunden geradeaus fahren.

1. LOSLEGEN

DU LERNST HIER...

wie du den mbRobot 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 mbrobot import *

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

Mit der ersten Zeile importierst du die Robotik-Bibliothek *mbrobot*. 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*. Die Einstellungen bleiben gespeichert. Du kannst sie jeder Zeit anschauen oder ändern. Dazu klickst du auf den Button *Einstellungen* in der Menüleiste und wählst das Register *Bibliotheken*.

Klicke auf den Robotik-Button "Hinunterladen/Ausführen".

Während der Programmausführung wird auf deinem Computer ein **Terminalfenster** angezeigt, in dem Fehlermeldungen vom mbRobot angezeigt werden.

Du siehst jetzt eine Meldung "switch the robot on".

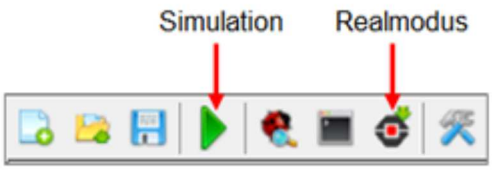
Ziehe den USB-Kabel vom micro:bit aus und kippe den kleinen Schalter hinten auf dem Maqueen-Chassis auf "ON*. Der Roboter fährt eine kurze Strecke vorwärts.

Auf der Rückseite des micro:bits befindet sich ein ***Reset***-Button, mit dem das zuletzt heruntergeladene Programm erneut ausgeführt werden kann. Du kannst auch den kleinen Schalter auf "OFF" - und wieder auf "ON" schalten. Dann wird das letzen Programm erneut ausgeführt.

Falls du längere Zeit den Roboter nicht brauchst, solltest du die Stromversorgung wieder ausschalten (Schalter auf "OFF"), damit der Akku nicht unnötig entladen wird.

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 mbRobot herunterzuladen. Der micro:bit des Roboters muss dabei über das USB-Kabel an deinem Computer angeschlossen sein. Vor der Ausführung musst du mit dem kleinen Schalter hinten am mbRobot die Stromversorgung einschalten.

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

2. ROBOTER STEUERN (AKTOREN)

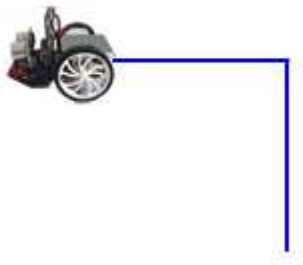
DU LERNST HIER...

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 Programm 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 mbrobot import *

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

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 mbrobot 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 mbrobot import *

setSpeed(40)
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

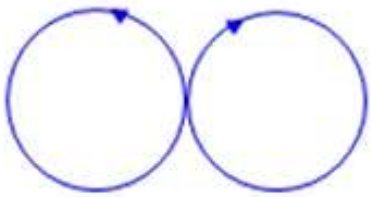
Bei der Roboterprogrammierung musst du in Zuständen denken. Der Befehl *forward()* versetzt den Roboter in eine Vorwärtsbewegung. Während die Motoren in diesem Zustand sind, kann der Microprozessor Befehle an andere Aktoren erteilen.

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

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 mbalarm importieren.

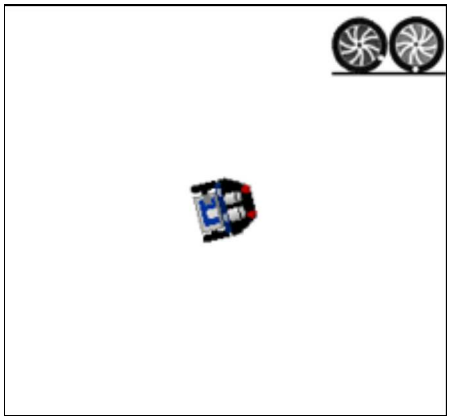
   Mit nebenstehendem Programm wird der Alarm während 3000 ms eingeschaltet.

```
from mbrobot import *
from mbalarm 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 mbrobotmot import *

motL.rotate(50)
delay(3000)
motL.rotate(-30)
delay(2000)
motL.rotate(0)
```

Beispiel 5: Leds eonzeln ein - und ausschalten
Mit den Befehlen *setLED(1)* und *setLED(0)* kannst du beide LEDs gleichzeitig ein- und ausschalten. Beim genauen Hinsehen auf die Platine deines Roboters, siehst du, dass die linke LED mit LED-L und die rechte mit LED-R beschriftet ist. Du kannst die LEDs auch einzeln ansprechen:
ledLeft.write_digital(1) schaltet die linke LED ein / ledLeft.write_digital(0) aus
ledRight.write_digital(1) schaltet die rechte LED ein / ledRight.write_digital(0) aus

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

```
from mbrobot import *

ledLeft.write_digital(1)
ledRight.write_digital(0)
delay(600)
ledLeft.write_digital(0)
ledRight.write_digital(1)
delay(600)
ledLeft.write_digital(1)
ledRight.write_digital(0)
delay(600)
ledLeft.write_digital(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 Kreibogen 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

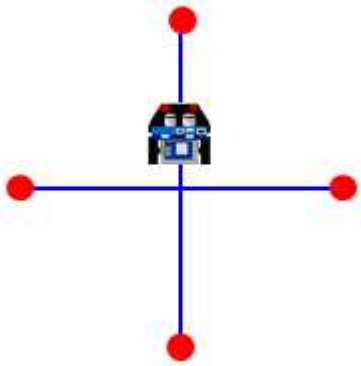
Beispiel 1: Quadrat fahren
Um ein Quadrat zu fahren, muss der Roboter vier Mal eine Strecke vorwärts fahren und um 90° drehen. Für die sich wiederholende Bewegung verwendest du eine repeat-Schleife. Diese wird mit repeat eingeleitet gefolgt von Anzahl Wiederholungen und einem Doppelpunkt. Die Befehle,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

```
from mbrobot 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 rechtwinklich zu einander stehen und fährt jeweils wieder rückwärts zum Ausgangspunkt.

```
from mbrobot import *

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

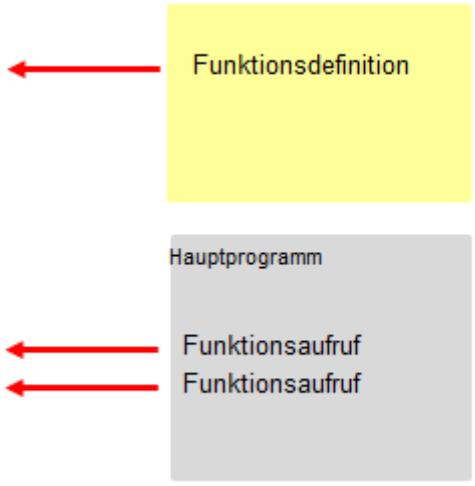
Hier ist es nützlich zu wissen, wie man ein laufendes Programm abbrechen kann: Am einfachsten geht es mit switsch

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 macht der Roboter eine ähnlich Bewegung wie im vorhergehenden Beispiel. Bevor er rückwärtsfährt hält er an, Blinkt zweimal und fährt rückwärts zum Ausgangspunkt.

```
from mbrobot import *

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

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



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
```

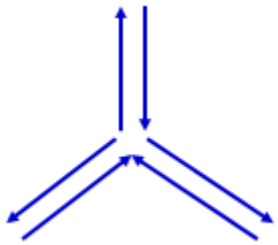
Die eingerückten Zeilen werden so lange wiederholt, bis man das Programm mit Schliessen des Terminalfensters abbricht.

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

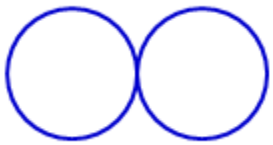
Definition:
```
def name():
    Anweisungen
```
Aufruf:
```
name()
```

ZUM SELBST LÖSEN

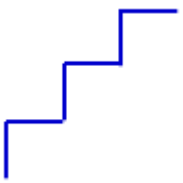
1. Der Roboter startet in der Mitte, fährt eine kurze Strecke vorwärts, dreht um 180° und fährt zurück zum Ausgangspunkt und dreht wieder in die Startrichtung. Dann dreht er etwa um 120° nach rechts. Diese Befehlssequenz wiederholt er drei Mal.



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.



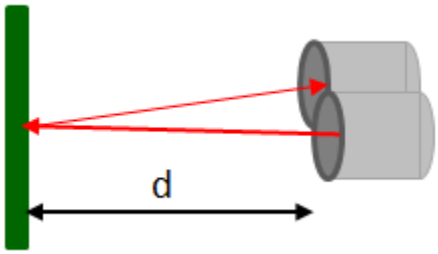
4. ULTRASCHALLSENSOR (DISTANZSENSOR)

DU LERNST HIER...

wie du mit einen Ultraschallsensor Distanzen messen und auswerten kannst.

WIE FUNKTIONIERT EIN ULTRASCHALLSENSOR?

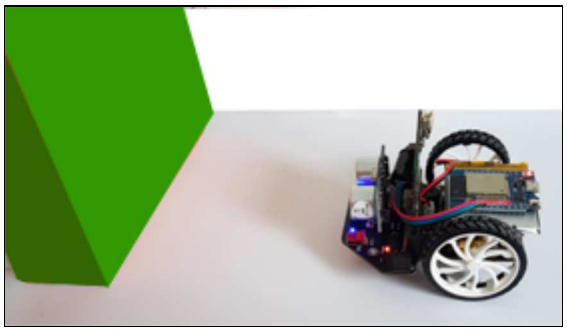
Ein Ultraschallsensor hat einen Sender und einen Empfänger für die Ultraschallwellen. Bei einer Distanzmessung sendet er einen kurzen Ultraschallpuls und misst die Zeit, die dieser benötigt, um zum Objekt und wieder zurück zu laufen. Daraus kann er mit Hilfe der bekannten Schallgeschwindigkeit die Distanz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



Der Maqueen-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.



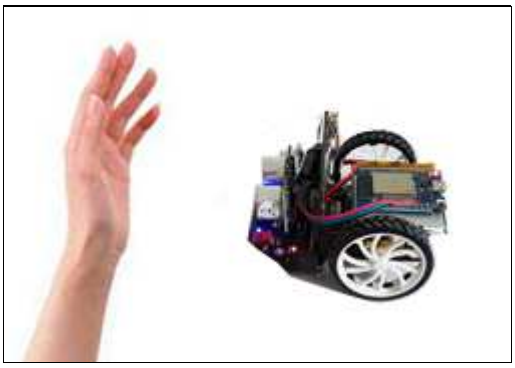
```
from mbrobot import *

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. Mit *print(d)* werden 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.

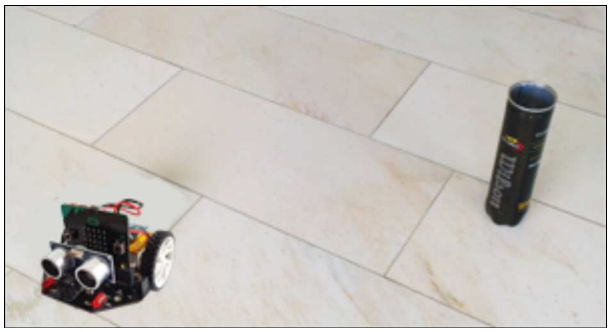
Für die Auswertung der Sensorwerte verwendest du die *if-else-Struktur*. Falls die Bedingung nach *if* stimmt, wird die Anweisung nach *if* ausgeführt, sonst die Anweisung nach else.



```
from mbrobot import *

forward()
setSpeed(10)
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 mbrobot import *

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

setSpeed(10)
searchTarget()
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.

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. Beim Einparkieren werden Distanzsensoren verwendet, die ähnlich funktionieren, wie diejenigen unseres kleinen mbRobots.

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.

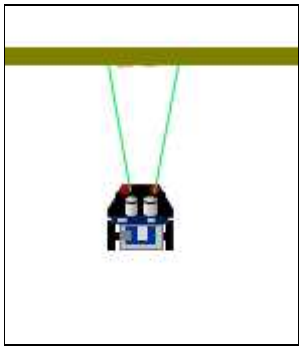
4. Ein Roboter mit einem Ultraschallsensor versucht den Ausgang aus einem begrenzten Raum zu finden. Er dreht zuerst langsam am Ort und wenn er keine Wand detektiert, fährt er los.

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:



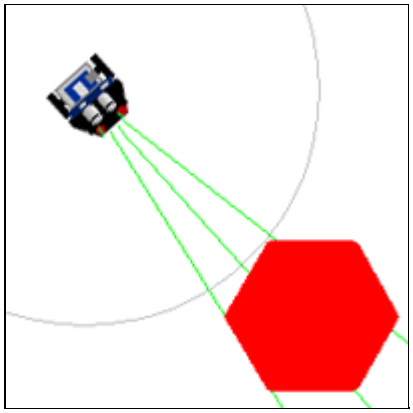
```
target = [[200, 10], [-200, 10], [-200, -10], [200, -10]]
RobotContext.useTarget("sprites/bar0.gif", target, 250, 100)
```

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 mbrobot 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 mbrobot 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
```

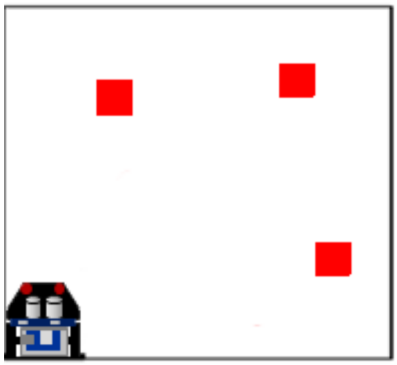
Das Objekt ist mit den Koordinaten eines Sechsecks mit dem Mittelpunkt (0, 0) beschrieben und mit dem Bild *redtarget.gif* an der Position (400, 400) dargestellt wird.

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)
```

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.

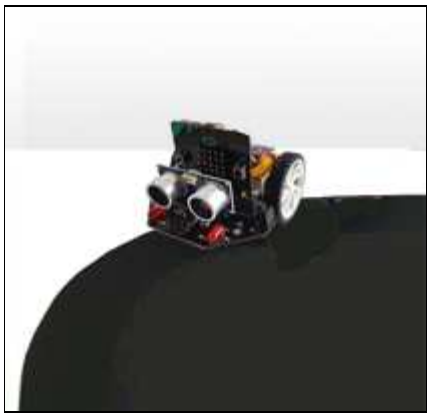
Der mbRobot verfügt über **2 Infrarotsensoren**. Man findet sie auf der unteren Seite des Boards, bezeichnet mit Line-R und Line-L. Da das Infrarottlicht an hellen bzw. dunklen Flächen unterschiedlich reflektiert, können die Sensoren zwischen einer hellen und dunklen Unterlage unterscheiden und geben 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 = irLeft.read_digital() der Sensorwert in der Variablen v gespeichert.



```
from mbrobot import *

repeat:
   v = irLeft.read_digital()
   if v == 0:
      rightArc(0.2)
   else:
      leftArc(0.2)
   delay(100)
```

Du kannst das Programm auch im Simulationsmodus ausführen. Dazu musst du nach der Zeite `from mbrobot 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 = irRight.read_digital()
vL =irLeft.read_digital()

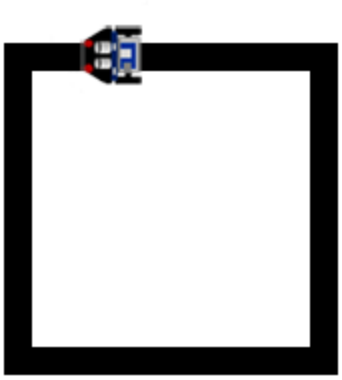
Danach entwickelst du ein Algorithmus, mit dem der Roboter den Weg möglichst genau folgen kann.

```
from mbrobot import *

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

setSpeed(30)
repeat:
    vR = irRight.read_digital()
    vL = irLeft.read_digital()
    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 mbrobot 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 = irRight.read_digital()
    vL = irLeft.read_digital()
    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.

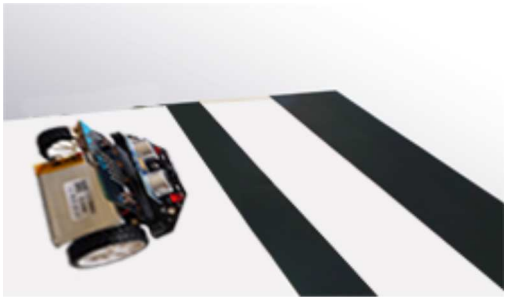
   ```
   RobotContext.useBackground("sprites/circle.gif")
   ```

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).

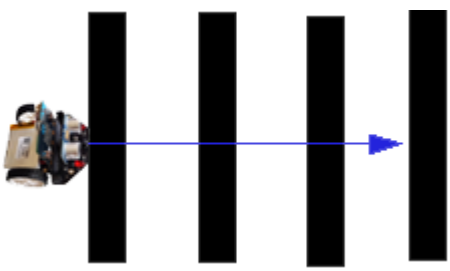
   

   Für den Simulationsmodus benötigst du folgenden Context:

   ```
   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.

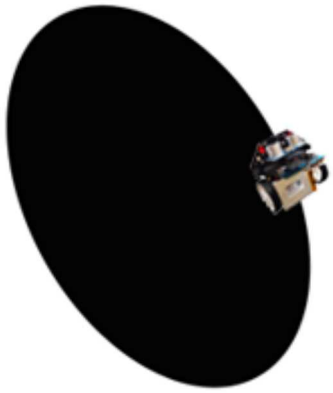
Für die Simulation verwendest du folgenden Context:

```
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

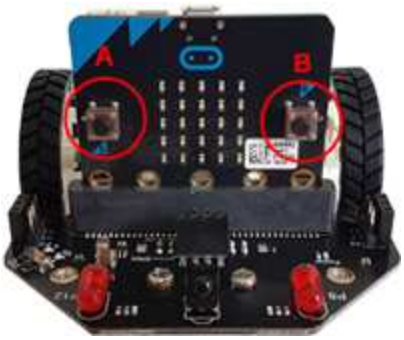
DU LERNST HIER...

wie man die beiden micro:bit-Buttons verwendet, um interaktive Programme zu entwickeln.

WIE FUNKTIONIEREN DIE BUTTONS

Der micro:bit 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 microbit import *
from mbrobot import *

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

repeat:
    if button_a.is_pressed():
        blink(ledRight)
    if button_b.is_pressed():
        blink(ledLeft)
    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 ledLeft oder *ledRight* sein, je nachdem, ob der linke oder rechte Button gedrückt wird. Die Befehle für die Buttons sind im Modul microbit, das du zusätzlich importieren musst.

Beispiel 2: Auf das Klicken eines Buttons reagieren

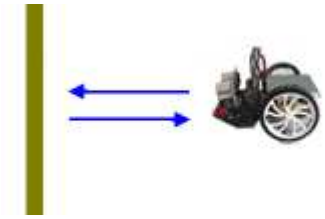
Mit einem Klick auf den Button A oder B soll eine kurze Melodie abgespielt werden. Die Funktion button_a.was_pressed() gibt True zurück, wenn der *Button a* kurz gedrückt wurde. Der Befehl play(JUMP_UP) aus dem Modul music spielt eine kurze Melodie ab. Einige Melodien sind im Modul music eingebaut.

```
from microbit import *
from mbrobot 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 microbit import *
from mbrobot import *

setSpeed(15)
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 micro:bit Bild (eventuell musst du das andere Grafikfenster verschieben). Die Buttons bedienst du mit Mausklick.

```
from microbit import *
from mbrobot 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.

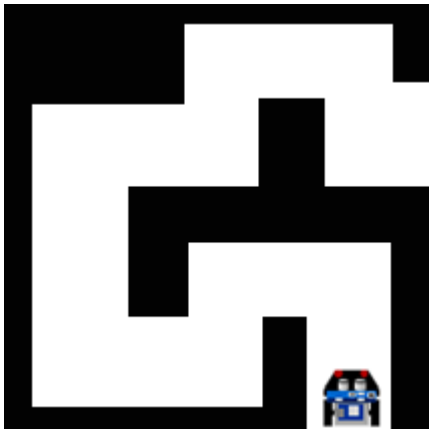
ZUM SELBST LÖSEN

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. ROBOTER PER WLAN-STEUERN

DU LERNST HIER...

wie du auf dem ESP32-Microcontroller einen einfachen Webserver einrichten und mit interaktiven Webseiten den Roboter über das WLAN steuern kannst.

VERNETZTE ROBOTER

Durch den Einbau von kleinen **WLAN-fähigen Microcontrollern** können Geräte über ein Netzwerk mit übergeordneten Systemen kommunizieren. In der Industrie werden Lager- und Transportroboter bequem und effizient **per WLAN gesteuert**. Immer häufiger können Roboter, auch Haushaltsroboter über das Internet ferngesteuert werden.

Der mbRobot kann mit einem leistungsfähigen ESP32 Mikrocontroller, der über einen integrierten WLAN verfügt, erweitert werden. Der ESP32 wird über die I^2C-Schnittstelle (vorne beim Ultraschallsensor) angeschlossen und mit einem Doppelklebband an der Lipo-Batterie befestigt.

Eine Anleitung zur Installation des ESP32 Microcontrollers findest du hier

Die Kommunikation kann über einen bestehenden **Accesspoint** erfolgen. In grösseren Institutionen oder im Klassenverband ist es vorteilhaft, eigenen, billigen WLAN-Router (muss nicht mit Internet verbunden sein) zu verwenden. Du kannst aber auch **WLAN- Hotspot** deines Smartphones nutzen.

Der mbRobot dient als Webserver, kann Webseiten zur Verfügung stellen und auf die eingehenden Anfragen (**Requests**) der Benutzer (**Clients**) reagieren. Die Clients (Smartphone, PC) können mit einem Webbrowser auf diese Webseiten zugreifen. Die notwendigen Befehle sind im Modul ***linkup*** enthalten, dieses muss zusätzlich importiert werden.

MUSTERBEISPIELE

Beispiel 1: LEDs ferngesteuert über das WLAN ein- und ausschalten
Der Roboter erhält seine IP-Adresse vom Accesspoint. Du kannst sie im Terminalfenster oder als Scrolltext auf dem micro:bit anzeigen, indem du im folgenden Programm die SSID und Passwort deines Accesspoints eingibst und das Programm auf dem micro:bit ausführst:

```
from linkup import *
ipAddress = connectAP(ssid = "xx", password = "yy")
display.scroll(ipAddress, wait = False)
print(ipAddress)
```

Die IP-Adresse besteht aus 4 Zahlen getrennt durch einen Punkt (z.B. 192.168.0.22)

Mit deinem Programm startest du auf dem Roboter einen Webserver und lädst eine einfache Webseite hoch. Diese enthält nur eine Überschrift und zwei Links.

Gibt der Client (Smartphone) die IP-Adresse des Webservers (Roboter) im Browser ein, so wird im seinem Browser diese Webseite angezeigt. Mit Klick auf den Link Light ON wird "/ON" als sogenannte URL-Parameter zum Server zurückgeschickt, dieser führt den im Programm festgelegten Befehl aus und schaltet die LEDs ein.

In den nachfolgenden Beispielen musst du immer zwei Programme über das USB-Kabel auf den micro:bit des Roboter herunterladen, obwohl die Programme teilweise auf dem ESP ausgeführt werden. Der ESP kommuniziert während der Programmausführung mit dem micro:bit wie ein Coprozessor.

Mit dem ersten Programm wird der HTML-Code der Webseite hochgeladen.

```
from linkup import *
html = """
<!DOCTYPE html>
<html>
  <head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <head>
  <body>
  <h2>Welcome to the WebRobot</h2>
    <p><a href="on">Light ON></a></p>
    <p><a href="off">Light OFF></a></p>
  <body>
</html>
"""
print("Saving HTML")
saveHTML(html)
```

Der HTML-Code wird in einem Python-Programm als ein mehrzeiliger String zwischen zwei dreifachen Anführungszeichen """ ..""" eingegeben. Mit der Zeile

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

wird die Webseite an die Breite des Gerätes angepasst. So erscheint sie auf einem Smartphone, Tablet oder PC korrekt. Mit dem Befehl saveHTML(html) wird der HTML-Code auf den Webserver hochgeladen.

Mit dem zweiten Programm wird festgelegt, wie der Webserver auf die Anfragen (Requests) der Benutzer reagieren soll und der Webserver gestartet .

```
from mbrobot import *
from linkup import *

def onRequest(clientIP, state, params):
    if state == "/on":
        setLED(1)
    elif state == "/off":
        setLED(0)

ipAddress = connectAP(ssid = "xx", password = "yy")
display.scroll(ipAddress, wait = False)
startHTTPServer(onRequest)
```

Die Aktionen des Webservers erfolgen Eventgesteuert. Wenn der Benutzer die IP-Adresse des Servers wählt, wird die Funktion onRequest() aufgerufen und die Webseite an die clientIP zurückgeschickt. Wählt der Client den Link *Light ON*, so wird als sogenannter URL-Parameter state "/on" zurückgeschickt und der Server führt den Befehl *setLED(1)* aus.

Der Roboter erhält vom Accesspoint in der Regel immer die gleiche IP-Adresse. Falls du die IP-Adresse nicht mehr weisst, füge die nachfolgenden Zeilen in deinem Programm ein (vor der Zeile *start HTTPServer()*). Die IP-Adresse wird nach dem Programmstart im Terminal-Fenster angezeigt.

```
ipAdress = connectAP(ssid = "xx", password = "yy")
print(ipAdress)
```

Teste das Programm mit deinem Smartphone. Dazu muss er aber mit dem gleichen Accesspoint wie der Roboter verbunden sein.

Beispiel 2: Roboter per WLAN steuern
Du möchtest nun deinen Roboter per WLAN steuern. In dieser einfachen Version stehen dir nur zwei Befehle zur Verfügung. Mit dem ersten Link kannst du den Roboter in die Vorwärtsbewegung versetzen, mit dem zweiten Link kannst du ihn stoppen.


```
from linkup import *

html = """
<!DOCTYPE html>
<html>
  <head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <head>
  <body>
  <h2>WebRobot</h2>
     <p>Press to change the state:</p>
```

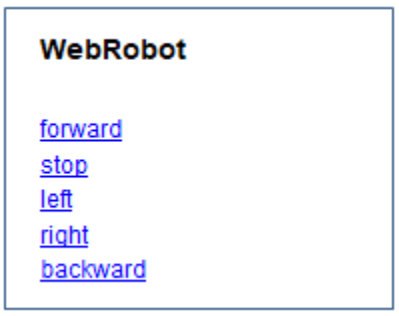
```
    <p><a href="forward">forward</a></p>
    <p><a href="stop">stop</a></p>
  </body>
</html>
"""
print("Saving HTML")
saveHTML(html)
```

```
from mbrobot import *
from linkup import *

def onRequest(clientIP, state, params):
    if state == "/forward":
        forward()
    elif state == "/stop":
        stop()

#ipAddress = connectAP(ssid="xx", password="yy")
#display.scroll(ipAddress, wait = False)
startHTTPServer(onRequest)
```

Wenn der Benutzer auf den Link "forward" klickt, erhält der Parameter state den Wert "/forward". In der Funktion onRequest() legst du fest, welchen Befehl der Roboter in diesem Fall ausführen muss. Falls du die IP-Adresse des Roboters nicht mehr weisst, aktiviere die auskommentierten Zeilen.

ZUM SELBST LÖSEN

1. Ergänze das Beispiel 2 mit den Links *left, right* und *backward*, um eine vollständige Smartphone-Steuerung zu realisieren. Im Zustand *left* soll der Roboter auf einem Linksbogen mit dem Radius 0.1 fahren und bei *right* auf einem Rechtsbogen mit dem Radius 0.1.



MERKE DIR...

Auf dem Microcontroller des Roboters kann ein Webserver eingerichtet werden, der eine interaktive Webseite speichern und zur Verfügung stellen kann. Die Webseite kann mit Browser auf einem Smartphones oder PC angezeigt werden. Mit Klick auf Links oder Buttons werden im Programm festgelegten Aktionen ausgelöst.

ZUSATZSTOFF

Beispiel 3: Steuerung mit Buttons

Unter Verwendung von Interaktiven Schaltflächen kannst du die Webseite benutzerfreundlicher gestalten. Die sogenannten Submit-Buttons, sind in einem Formular angeordnet. Die beiden Buttons haben den Namen 'btn', mit der Beschriftung erhält jeder Button seinen Wert. Der ersten Button value "forward" , der zweite Button value "stop".

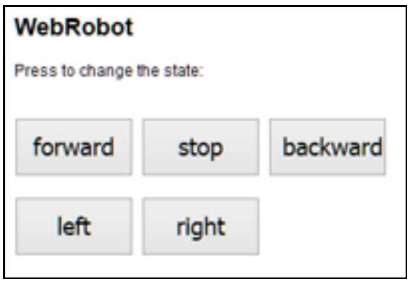

```
from linkup import *

html = """<!DOCTYPE html>
<html>
  <head> <title>Web Rover</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <h2>WebRobot</h2>
    <form method="get">
       <p><input type="submit" style="font-size:18px; height:40px;
           width:110px" name="btn" value="forward"/></p>
       <p><input type="submit" style="font-size:18px; height:40px;
           width:110px" name="btn" value="stop"/></p>
    </form>
  </body>
</html>
"""

print("Saving HTML...")
saveHTML(html)
```

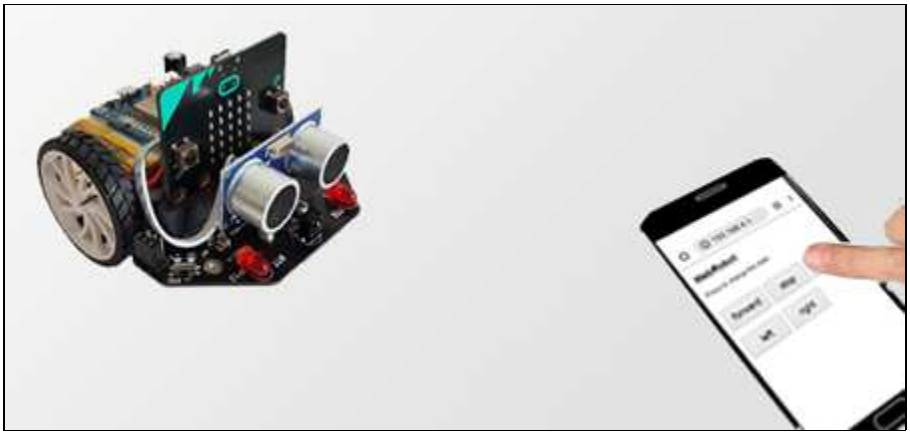
Die Informationen werden wieder im URL übermittelt. In der Funktion onRequest() speicherst du den Wert des Parameters "btn" in der Variablen state und legst mit *if-elif* fest, welche Aktion der Roboter ausführen soll. Der zweite Teil des Programms ist gleich wie im Beispiel 2.

```
from mbrobot import *
from linkup import *

def onRequest(clientIP, filename, params):
    if "btn" in params:
        state = params["btn"]
        if state == "forward":
            forward()
        elif state == "stop":
            stop()

state = "stop"
setSpeed(40)
#ipAddress = connectAP(ssid="xx", password="yy")
#display.scroll(ipAddress, wait = False)
startHTTPServer(onRequest)
```

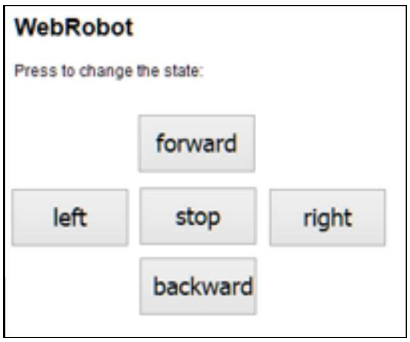
ZUM SELBST LÖSEN

2. Ergänze das Beispiel 3 mit den Buttons *left, right* und *backward*, um eine vollständige Smartphone-Steuerung zu realisieren. Im Zustand left soll der Roboter auf einem Linksbogen mit dem Radius 0.1 fahren und bei right auf einem Rechtsbogen mit dem Radius 0.1.





3* Informieren dich in einem HTML-Tutorial, wie man Elemente in einer Tabelle anordnen kann und verbessere die Benutzeroberfläche deiner Robotersteuerung.



8. INTERNET OF THINGS (IoT)

DU LERNST HIER...

die Sensordaten deines Roboters per WLAN zur Verfügung stellen, ähnlich wie Milliarden IoT-Geräten, die weltweit miteinander und mit übergeordneten Systemen kommunizieren.

WAS IST IOT?

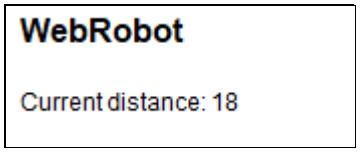
Das IoT ist eines der aktuellsten Themen der heutigen Informatik. Durch Einbau von kleinen, billigen aber WLAN-fähigen Microcontrollern können Geräte und Systeme (kurz Dinge) über ein Netzwerk Daten und Informationen untereinander und mit übergeordneten Systemen kommunizieren. Im Zusammenhang mit IoT spricht man häufig von Web 3.0, einem Internet, in dem Systeme mit Sensoren automatisch eine grosse Menge von Daten erfassen, auf Cloud-Servern speichern und durch weit entfernte Kommandozentralen gesteuert werden.

Auch dein Roboter verfügt über einen WLAN-fähiger Microcontroller und Sensoren, mit welchen du einfache IoT-Anwendungen programmieren kannst.

MUSTERBEISPIELE

Beispiel 1: Messwerte des Ultraschallsensors per WLAN zur Verfügung stellen

Du übernimmst das Konzept aus dem vorhergehenden Kapitel. Mit deinem Programm wird eine einfache Webseite auf den Roboter hochgeladen. Diese dient aber nicht zur Steuerung des Roboters, sondern zum Anzeigen der Sensorwerte.



Der Roboter misst die Distanz zum Hindernis und stellt diese Werte auf der Webseite zur Verfügung. Diese kann mit einem Webbrowser auf einem Smartphone oder PC angezeigt werden.

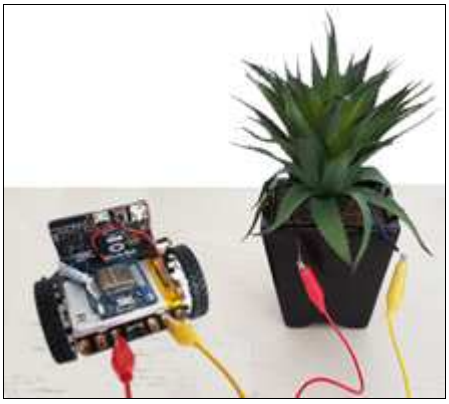
```
from linkup import *

html = """<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="refresh" content="3">
  </head>
  <body>
    <h2>WebRobot</H2>
    Current distance: %s<br>
  </body>
</html>
"""
print("Saving HTML")
saveHTML(html)
```

Die Webseite besteht nur aus einer Überschrift und einer zweiten Zeile, in der die Distanz angezeigt wird. %s ist ein Platzhalter für den Messwert, welchen der Ultraschallsensor zurückgibt. Der zweiter Meta-Tag im HTML-Code bewirkt, dass die Webseite alle 3 Sekunden automatisch aktualisiert wird.

```
from mbrobot import *
from linkup import *

def onRequest(clientIP, state, params):
   d = getDistance()
   return [d]

ipAddress = connectAP(ssid="xx", password="yy")
print(ipAddress)
startHTTPServer(onRequest)
print("Server starting")
```

Wenn ein Benutzer im Webbrowser die IP-Adresse des Roboters (z.B. 192.168.0.22) eingibt, wird die Webseite mit der aktuell gemessenen Distanz angezeigt. Das Smartphone/PC muss mit dem gleichen Accesspoint wir der Roboter verbunden sein.

Beispiel 2: **Wasserstand online überwachen**



Der Wasserstand einer Pflanze kann über das WLAN z.B. mit einem Smartphone überwacht werden.

Die micro:bit-Pins sind auch auf dem mbRobot zugänglich, so dass man sehr einfach mit Krokodilklammern einen Stromkreis aufbauen kann. Das eine Kabel verbindet man mit Pin1, das andere mit 3V.

Pin1 und Pin2 werden für den Ultrasonic-Sensor verwendet. Wenn man diesen entfernt, so sind die beiden Pins für andere Anwendungen frei verwendbar.

Bei niedrigem Wasserstand sind die Messwerte, die man mit dem Befehl pin1.read_analog() erhält, wesentlich kleiner als beim hohen Wasserstand, wenn die beiden Kontakte im Wasser sind. Im Browser des Smartphones oder PCs werden die Werte angezeigt.

```
from linkup import *

html = """<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="refresh" content="5">
  </head>
  <body>
     <h2>Remote services</h2>
     Current humidity: %s<br>
  </body>
</html>
"""

print("Saving HTML...")
saveHTML(html)
print("Done")
```

```
from linkup import *
from mbrobot import *
from microbit import *

def onRequest(clientIP, filename, params):
   v = pin1.read_analog()
   return [v]

#ipAddress = connectAP(ssid = "xx", password = "yy")
#print(ipAddress)
startHTTPServer(onRequest)
```

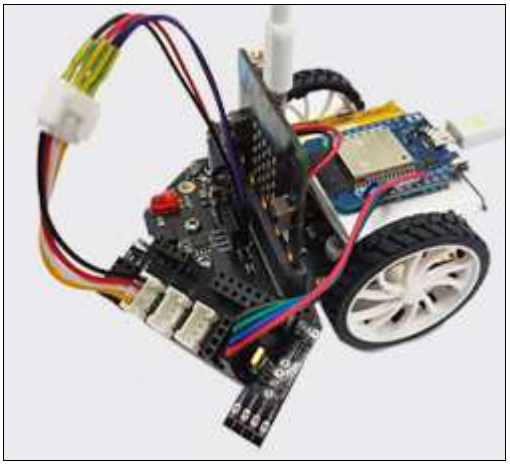
Falls du di IP-Adresse deines Roboters nicht mehr weisst, kannst du die beiden auskommentierten Zeilen aktivieren. Die IP wird danach im Terminal-Fenster angezeigt. Verbinde deinen Smartphone mit dem gleichen Accesspoint und gib im Browser diese IP-Adresse ein, um den Wasserstand anzuzeigen.

Umweltsensoren

Die Robotik-Bibliothek unterstützt mehrere Sensoren, mit welchen man die wichtigsten Umweltindikatoren: die Umgebungstemperatur, die Luftfeuchtigkeit und den Luftdruck übermitteln kann. Sensirion Temperatur- und Luftfeuchtigkeits-Sensoren, die im nächsten Beispiel verwendet werden, sind hochpräzise Geräte, mit einer Genauigkeit von ±0,3°C bei den Temperaturmessungen und ±2% bei der Luftfeuchtigkeit. Man kann sie praktisch an jeden Microcontroller anschliessen.

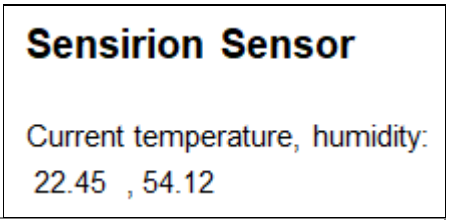
Beispiel 3: Sensirion Temperatur- und Feuchtigkeitssensor
Der Sensirion SHT31 und SHT85 sind billige und vielseitig einsetzbare digitale Temperatur- und Feuchtigkeitssensoren hoher Präzision. Interessant sind solche Messungen insbesondere zusammen mit Datenübertragung, wenn der Roboter die Temperatur und Luftfeuchtigkeit misst und diese Daten über das WLAN an einen Computer oder Smartphone liefert.

Am einfachsten Schliesst man den Sensirion-Sensor mit Hilfe des IoT-Sets an. Dieser enthält den SHT85-Sensor und alle notwendigen Komponenten, um den Sensor am mbRobot oder direkt am micro:bit anzuschliessen.



Mit dem ersten Programm wird der HTML-Code einer einfachen Webseiten hochgeladen.

Diese enthält eine Überschrift und die Anzeige der Sensorwerte. %s, %s sind Platzhalter für die Sensorwerte *temp* und *humi*.




```
from linkup import *

html = """<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="refresh" content="3">
  </head>
  <body>
    <h2>Sensirion Sensor</H2>
    Current temperature, humidity: <br><br>
    %s, %s
  </body>
</body>
</html>
"""
print("Saving HTML")
saveHTML(html)
```

Im Hauptprogramm importierst du das Modul *sht*, welches die Funktionen für den Sensirion-Sensor enhält. Mit dem Befehl *sht.getValues()* werden die aktuellen Temperatur und Feuchtigkeit gemessen und in den Variablen *temp* und *humi* gespeichert. Die beide Return-Werte erscheinen danach auf der Webseite.

```
from linkup import *
from mbrobot import *
import sht

def onRequest(clientIP, filename, params):
   temp, humi = sht.getValues()
   return [temp, humi]

#ipAddress = connectAP(ssid = "xx", password = "yy")
#print(ipAddress)
startHTTPServer(onRequest)
```

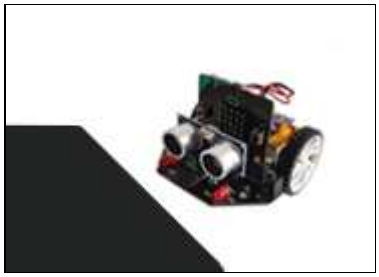
Um die Werte auf deinem Smartphone anzuzeigen, musst du im Browser die IP-Adresse des Roboters eingeben. Falls du diese nicht mehr weisst, kannst du die beide auskommentierte Zeilen aktivieren, damit die IP-Adresse im Terminalfenster angezeigt wird.

MERKE DIR...

Als **IoT** («Internet of Things») bezeichnet man eine aktuelle Technologie, die es durch den Einbau von billigen, WLAN-fähigen Microcontrollern den Geräten (kurz "Dingen") ermöglicht, über das Internet Sensordaten und Informationen untereinander auszutauschen und mit übergeordneten Systemen zu kommunizieren. Auch mit deinem mbRobot kannst du IoT-Anwendungen programmieren.

ZUM SELBST LÖSEN

1. Programmiere eine IoT-Anwendung, bei der der Roboter die Werte des linken Infrarotsensors auf einer Webseite zur Verfügung stellt, so dass du diese mit deinem Smartphone anzeigen kannst.



2. Programmiere eine Alarmanlage. Ein Roboter mit einem Ultraschallsensor. überwacht einen Raum. Du kannst jederzeit mit deinem Smartphone die Distanz, die er detektiert abfragen.

Falls sich ein Objekt näher als 30 cm befindet, schaltet er Alarm ein, wenn das Objekt entfernt, stellt er den Alarm ab.

9. CO_2 SENSOR

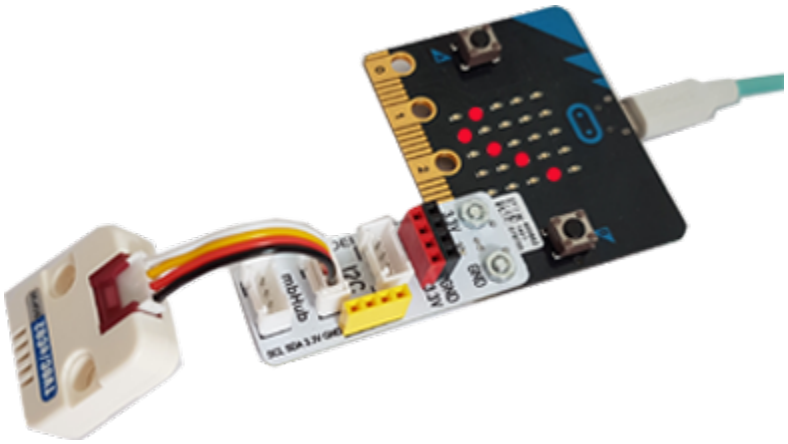
DU LERNST HIER...

mit dem 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 über die CO_2-Konzentration in der Luft. Das Modul *sgp*, 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 micro:bit 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. Das Anschluss Kabel ist in der Lieferung integriert.



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

.

Am mico:bit wird der Sensor mit dem mitgelieferten I2C Kabel über einen **I^2C-Hub** angeschlossen. Ein solcher Hub kann für Fr. 6.- bei der TigerJython-Group bestellt werden. Für den Versand in der Schweiz wird Fr. 1.-, aussehalb der Schweiz Fr. 3.- verrechnet.

Die Bestellung erfolgt mit einem EMail an admin@tjgroup.ch.

MUSTERBEISPIELE

Beispiel 1: CO2 Konzentration messen und anzeigen

In deinem Programm importierst du mit import sgp das Modul *sgp*, in dem SGP30-Sensor und das Abfragen der Messwerte des SGP30-Sensor implementiert ist. Der Befehl sgp.getValues() gibt ein Tupel mit zwei Werten zurück:

- CO2 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 microbit import *
import sgp

while True:
    co2, voc = sgp.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 CO2-Wert 400 zurück. Dann 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 für die Werte grösser als 1000, wird die Luft nicht mehr als frisch bezeichnet.

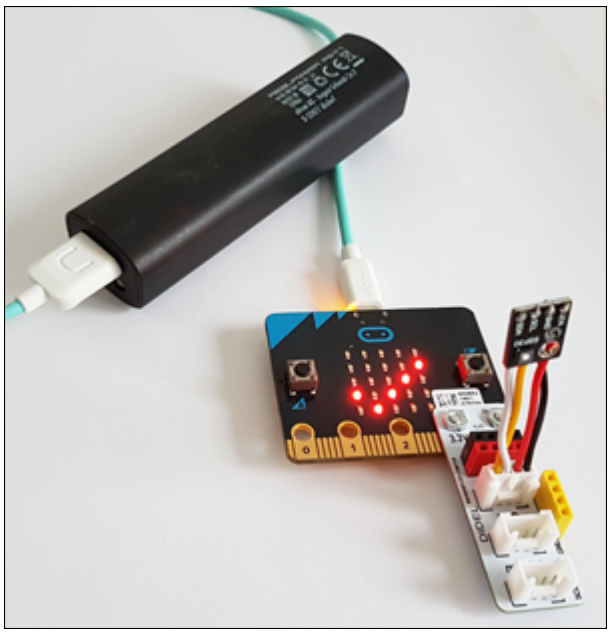
* In Innenräumen gibt es viele Quallen, die Schadstoffe abgeben (Lampen, Bodenbeläge, Reinigungsmittel...). Der Sensor gibt TVOC-Werte im Bereich 0 bis 60 000 zurück.

Beispiel 2: Ein Messgerät für CO2 Konzentration im Klassenzimmer

Für die Messung der CO_2 Konzentration gelten folgende Grenzwerte:

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

Mit den LEDs auf deinem micro:bit kannst für diese Messbereiche verschiedene Symbole anzeigen. Das Programm bleibt auf dem micro:bit gespeichert. Du kannst ihn also beim Computer ausstecken und an eine andere Stromquelle, beispielsweise Powerbank, anschliessen.



Programm:

```
from microbit import *
import sgp

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

► In Zwischenablage kopieren

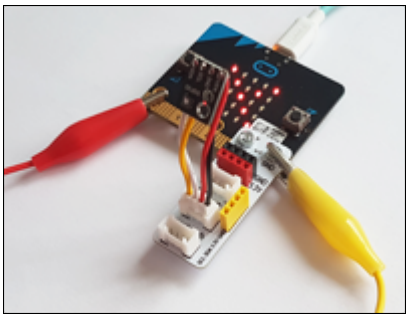
MERKE DIR...

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 micro:bit-Display.

2. Baue eine CO_2-Messanlage, die bei einem CO_2-Wert > 1400 einen akustischen Signal abspielt. Micro:bit V2 verfügt über einen Lautsprecher, beim micro:bit V1 musst du wie im Kapitel "Sound" einen Lautsprecher anschliessen. Zum Testen kannst du den Schwellenwert 1400 ppm herabsetzen.
   Der micro:bitV2 verfügt über



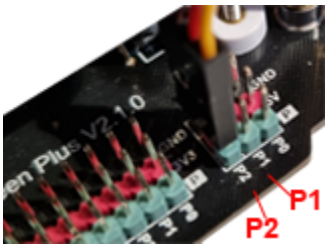
10. BAGGER MIT SERVOMOTOR (Mechanic Loader)

DU LERNST HIER...

eine MaqueenMechanic-Baggerschaufel (Loader) an deinen mbRoboter anschliessen und mit einem Servomotor steuern. Der Loader kann kleine Gegenstände aufladen und transportieren.

LOADER ZUSAMMENBAUEN

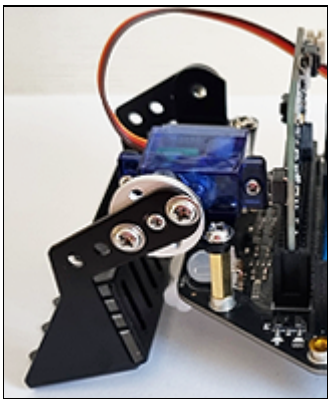
Den Bagger mit einem Servomotor kannst du einzeln (Maqueen Mechanics Loader) oder in einem Bausatz (Maqueen Mechanics Roboter-Kit) bestellen. Der Servomotor wird am **Port S1 bzw. S2** des Maqueen 4 oder Maqueen Plus angeschlossen (schwarzes Kabel auf GND).



Beim Maqueen Plus V2 wird der Servomotor am **Port P1** bzw. **P2** angeschlossen. Diese befinden sich hinten rechts auf dem Chassis. Im deinem Programm musst du die Servo-Ports im Befehl *setServo()* mit P1 bzw. P2 (statt S1 bzw. S2) aufrufen.

MUSTERBEISPIELE

Beispiel 1: Funktionalität des Servomotors testen



Der Servomotor bewegt den Loader zuerst zur Ausgangslage (unten), dann wird die Schaufel nach oben gedreht und kehrt nach 2000 Millisekunden nach unten in die Ausgangslage.

Mit dem Befehl *setServo("S1 ", 160)* dreht der Servomotor an die Position mit dem Drehwinkel 160°. S1 ist der Anschluss Port. Führe das unten stehende Programm mit verschiedenen Winkeln zwischen 0 und 180° aus um herauszufinden, für welchen Wert sich die Schaufel in der Ausgangslage (unten) befindet.

```
from mbrobot_plus import *
#from mbrobot 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 (ähnlich wie im oben stehenden Video). Um den Abstand zur Wand zu messen, verwendest du den Ultraschallsensor.

```
from mbrobot_plus import *
#from mbrobot 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)
```

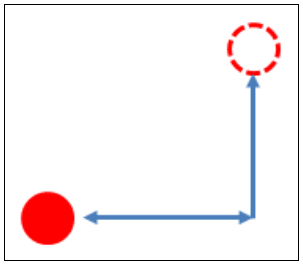
Die Distanz zur Wand wird in einer "endlosen" repeat-Schleife alle 100 Millisekunden abgefragt. Ist sie kleiner als 12 cm, wird der Gegenstand aufgeladen.

MERKE DIR...

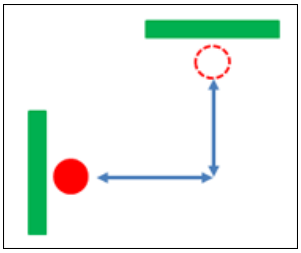
Der Servomotor wird mit dem Befehl *setServo(Port, Winkel)* an die gewünschte Position bewegt, wobei der erste Parameter S1 oder S2 ist und der Winkel im Bereich 0° bis 180° gewählt werden kann.

ZUM SELBST LÖSEN

1. Roboter mit einem Loader bewegt sich 2000 Millisekunden vorwärts und lägt ein 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 Ultraschallsensor beendet und den Aufladen- und Abladen-Vorgang mehrmals wiederholt.



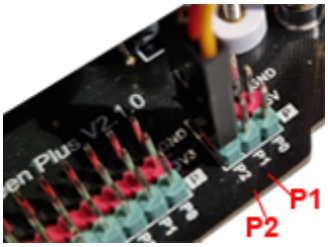
11. GREIFARM MIT SERVOMOTOR (Mechanic Beatle)

DU LERNST HIER...

einen Greifarm (Maqueen Mechanic Beatle) an deinen mbRoboter anschliessen und mit einem Servomotor steuern. Der Greifarm kann leichte Gegenstände packen und transportieren.

GREIFARM ZUSAMMENBAUEN

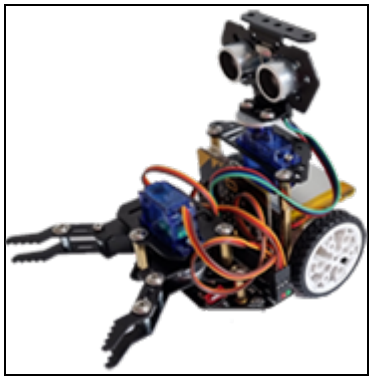
Den Greifarm mit einem Servomotor kannst du einzeln (Maqueen Mechanics Beatle) oder in einem Bausatz (Maqueen Mechanics Roboter-Kit) bestellen. Der Servomotor wird am **Port S1 bzw. S2** des Maqueens 4 oder an am Ports **S1 bzw. S2** der Maqueens Plus angeschlossen.



Beim Maqueen Plus V2 wird der Servomotor am **Port P1** bzw. **P2** angeschlossen. Diese befinden sich hinten rechts auf dem Chassis. Im deinem Programm musst du die Servo-Ports im Befehl *setServo()* mit P1 bzw. P2 (statt S1 bzw. S2) aufrufen.

MUSTERBEISPIELE

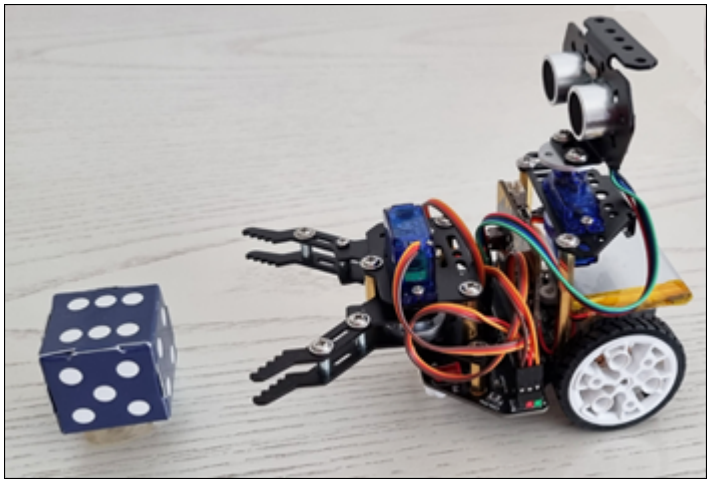
Beispiel 1: Greifarm testen Der Servomotor bewegt die beiden Arme des Greifarms gleichzeitig. Mit dem Befehl
setServo("S1", 60)
in die offene Stellung, mit
setServo("S1", 95)
kann der Roboter Gegenstände packen. Die Winkel musst du eventuell noch anpassen.



```
from mbrobot import *
#from mbrobot_plus import *

setServo("S1", 95)
delay(1000)
setServo("S1", 60)
delay(1000)
setServo("S1", 95)
```

Beispiel 2: Gegenstand fassen

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 oben stehenden Video).



```
from mbrobot import *
#from mbrobot_plus import *

setServo("S1", 95)
delay(1000)
setServo("S1", 60)
forward()
delay(1000)
stop()
setServo("S1", 95)
backward()
delay(500)
left()
delay(700)
stop()
setServo("S1", 60)
```

Beispiel 3: Mit Ultraschallsensor den Greiarm positionieren

Im Beispiel 2 startet der Roboter immer im gleichen Abstand vom Gegenstand und die Länge der Vorwärtsbewegung wird durch eine Zeitangabe (delay(1000)) bestimmt. Mit Hilfe eines Ultraschallsensors kann der Roboter die richtige Position selbst erkennen. Der Roboter bewegt sich vorwärts, bis sein Abstand zum Gegenstand oder zu einer hinter ihm stehenden Wand kleiner als 20 cm ist. Dann packt er den Gegenstand mit seinem Greifarm und transportiert ihn weg (ähnlich wie im unten stehenden Video).

```
from mbrobot import *
#from mbrobot_plus import *

setServo("S1", 60)
forward()
repeat:
    d = getDistance()
    print(d)
    if d < 20:
        stop()
        delay(500)
        setServo("S1", 95)
        delay(500)
        backward()
```

```
        delay(2500)
        left()
        delay(700)
        stop()
        setServo("S1", 60)
    delay(100)
```

Die Distanz zur Wand wird in einer "endlosen" repeat-Schleife alle 100 Millisekunden abgefragt. Ist sie kleiner als 20 cm, hält der Roboter an und packt den Gegenstand mit seinem Greifarm.

Beispiel 4: Ultraschallsensor mit Servomotor drehen
Gemäss der oben stehenden Bauanleitung wird der Ultraschallsensor auf einem zweiten Servomotor befestigt. Dieser wird am Port S2 angeschlossen. Den drehbaren Ultraschallsensor kannst du verwenden, um Gegenstände zu suchen (Aufgabe 3) oder um den Roboter in einem bestimmten Abstand einer Wand nach fahren zu lassen (Aufgabe 4).

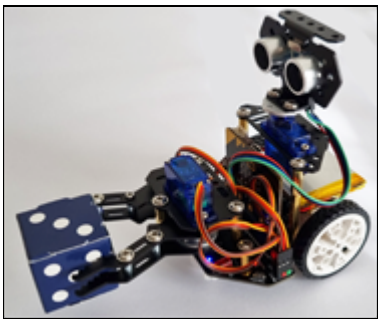
```
from mbrobot import *
#from mbrobot_plus import *

setServo("S2", 90)
delay(2000)
setServo("S2", 0)
delay(2000)
setServo("S2", 90)
delay(2000)

angle = 90
while angle >= 0:
    setServo("S2", angle)
    delay(500)
    angle = angle - 10

angle = 0
while angle <= 90:
    setServo("S2", angle)
    delay(500)
    angle = angle + 10
```

Das Programm zeigt, wie man den Ultraschallsensor mit einem zweiten Servomotor von der frontaler Position in eine Position, die parallel zur Fahrtrichtung ist drehen kann. Zuerst in einem Schritt und dann in kleinen Schritten.

MERKE DIR...

Der Servomotor wird mit dem Befehl *setServo(Port, Winkel)* um den gewünschten Winkel gedreht, wobei du für den erste Parameter S1 oder S2 und den Winkel eine Zahl zwischen 0 und 180 wählen kannst. Der Servomotor bewegt gleichzeitig beide Arme. Am Port S2 kannst du einen zweiten Servomotor anschliessen, mit dem du den Ultraschallsensor drehen kannst.

ZUM SELBST LÖSEN



1. Roboter mit einem Greifarm bewegt sich 2000 Millisekunden vorwärts und lädt einen Gegenstand auf. Dann dreht er 180 ° nach links, fährt 2000 Millisekunden vorwärts und stellt den Gegenstand ab. Danach dreht er 180° nach rechts und holt den nächsten Gegenstand.

2. Ergänze die Situation aus der Aufgabe 1 mit zwei Wänden, so dass der Roboter die Vorwärtsbewegungen jeweils mit Hilfe eines Ultraschallsensors 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.

Dokumentation mbRobot und micro:bit

Modul mbrobot
(Real- und Simultionsmodus)
Modul import: from mbrobot import * (from mbrobot_plus import *, from mbrobot_plusV2 import *)

Maqueen Roboter

Funktion	Aktion
forward()	setzt den mbRobot in Vorwärtsbewegung
backward()	setzt den mbRobot in Rückwärtsbewegung
left()	setzt dem mbRobot in eine Linksdrehung
right()	setzt dem mbRobot in eine Rechtsdrehung
leftArc(radius)	setzt den mbRobot auf eine Linkskurve mit gegebenem Radius (in m)
rightArc(radius)	setzt den mbRobot auf eine Rechtskurve mit gegebenem Radius (in m)
stop()	stoppt die Bewegung
setSpeed()	setzt die Geschwindigkeit für die Bewegungen (0..100), standard 50
getDistance()	liefert die Distanz (in cm, ungefähr im Bereich 2..200) gemessen mit dem Ultrasonic-Sensor (255: Fehlmessung)
irLeft.read_digital(), irRight.readDigital()	Infrarotsensoren liefern 1, falls helle Unterlage; 0, falls dunkle Unterlage
setLED(1)	Schaltet beide LEDs ein
setLED(0)	Schaltet beide LEDs aus
ledLeft.write_digital(n), ledRight.write_digital(n)	schaltet LEDs ein (n = 1) oder aus (n = 0)
setAlarm(1)	Schaltet Alarm ein (Pipton)
setAlarm(0)	Schaltet Alarm aus
setServo(port, angle)	Dreht den am Port S1 oder S2 (bzw. P1 oder P2) angeschlossenen Servomotor an die gegebene Winkelposition. (0 <= angle <= 180)
playTone(freq, duration)	Spielt einen Ton gegebener Frequenz und Dauer
setBeamAreaColor() [nur Simulation]	setzt die Farbe der Strahlbereichsgrenzen
setProximityCircleColor() [nur Simulation]	setzt die Farbe des Suchkreises
eraseBeamArea() [nur Simulation	löscht die Strahlbereichsgrenzen

Modul mbrobotmot (Motoren einzeln schalten)
(Real- und Simulationsmodus)
Modul import: from mbrobotmot 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

Direkte Funktionsaufrufe
Modul import: from microbit import *

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

Klasse Button
(Real- uns Simulationsmodus)

button_a	Objektreferenz (Instanz) des Buttons A
button_b	Objektreferenz (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

Klasse Display
(Real- und Simulationsmodus)

display	Objektreferenz (Instanz)

Methoden:

set_pixel(x, y, value)	setzt die Intenistät des Pixels an der Position x, y. value ist im Bereich 0..9
clear()	löscht alle Pixels
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)

Klasse MicroBitTouchPin
(Real- und Simulationsmodus)

pin0, pin1, pin2, pin8, pin12, pin16	Instanzen für allgemeines Digital-in/Digital-out
pin0, pin1, pin2	Instanzen für Analog-in/Analog-out (PWM)
pin3, pin4, pin6, pin7, pin9, pin10	Instanzen vordefiniert für LED display (display mode)
pin5, pin11	Instanzen vordefiniert für Button A, B (button mode)
pin13, pin14, pin15	Instanzen vordefiniert für SPI (spi mode)
pin19, pin20	Instanzen vordefiniert für I2C (i2c mode)

Methoden:

read_digital()	gibt True zurück, falls Pin auf logisch 1 (HIGH) ist; gibt False 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; fals 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)
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)

Klasse Accelerometer
(Real- und Simulationsmodus)

accelerometer	Objektreferenz (Instanz)

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 Erdgeschleunigung von ungefähr 10 m/s^2 wird mitgezählt). x-Richtung: ButtonA-ButtonB; y-Richtung: Pin2-USB; z-Richtung: Normale zu Board
get_values()	gibt ein Tupel mit den Beschleunigungen in x-, y- oder z-Richtung zurück (Einheit wie oben)
current_gesture()	gibt die aktuelle Geste zurück. Folgende Gesten werden erkannt: " up", " down", " left", " right", " face up", " face down", " freefall", " 3g", " 6g", " 8g", " shake"
is_gesture(name)	gibt True zurück, falls die aktuelle Geste gleich name ist
get_gestures())	gibt eine Tupel mit den zuletzt gemachten Gesten zurück. Die letzte Geste ist am Ende des Tupels. Löscht zudem die Gestenhistorie
was_gesture(name)	gibt True zurück, falls sich name in der Gestenhistorie befindet

Klasse Compass
(Realmodus)

compass	Objektreferenz (Instanz)

Methods:

calibrate()	startet eine blockierende Kalibrierungsroutine, die für genaue Messungen nötig ist. Man muss den micro:bit in verschiedenen Richtungen schief stellen, so dass der blinkende Punkt die Randpixel erreicht und diese anzündet. Erst wenn eine Kreisfigur erstellt ist, fährt das Programm weiter
is_calibrated()	gibt True zurück, falls der Sensor kalibriert wurde
clear_calibration()	setzt den Sensor auf den nicht-kalibrierten Zustand zurück
heading()	gibt den aktuellen Winkel des micro:bit zur Nordrichtung (Grad, int)
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, keine Kalibrierung nötig)
get_values()	gibt ein Tupel der x-, y- und z-Komponenten des Magnetfeldes an der Stelle des Sensors zurück (int, Mikrotesla, keine Kalibrierung nötig)

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

set_tempo(bpm = 120)	setzt die Anzahl Beats pro Minute (default: 120)
pitch(frequency, len, pin = microbit.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: P0). 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 = microbit.pin0, wait = True, loop = False)	spielt eine Melodie mit dem aktuellen Tempo.). pin definiert den Output-Pin am GPIO-Stecker (default: P0). 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

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
- 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
- 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:
(Realmodus)
Modul import: from radio import *
micro:bit-Kommunikation 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 clap
(Mini:Maqueen Rover, eingebautes Mikrophon am Port 2)
Modul import: from clap import *

wait_for_clap(timeout = 1000, sensitivity = 75)	wartet auf ein einzelnes Klatschen. Kehrt nach dem angegebenen Timeout (in ms) zurück, wenn kein Klatschen detektiert wurde. sensitivity im Bereich 0..100
wait_for_double_clap(timeout = 1000, spread = 500, sensitivity = 75)	wartet auf ein Doppelklatschen. Die beiden Klatschen müssen innerhalb des spread-Intervalls erfolgen (in ms). Kehrt nach dem angegebenen Timeout (in ms) zurück, wenn kein Doppelklatschen detektiert wurde. sensitivity im Bereich 0..100

Modul linkup
(ESP32 Coprozessor am I2C-Port)
Modul import: from linkup import *

connectAP(ssid, password)	verbindet mit dem bestehenden Access-Point (Hotspot, Router) mit ssid und password. Gibt die erhaltene gepunktete IP-Adresse zurück; leer, falls das Einloggen misslingt
createAP(ssid, password)	erzeugt einen Access-Point mit ssid und password. Falls password leer ist, ist der AP offen, d.h. man kann sich ohne Authentifikation einloggen
httpGet(url)	führt einen HTTP GET Request durch und liefert den Response zurück. url in der Form "http://<server>?key=value&key=value&..." Statt http kann auch https verwendet werden
httpPost(url, content)	führt einen HTTP POST Request durch und liefert den Response zurück. url in der Form "http://<server>". content im Format "key=value&key=value&..." Statt http kann auch https verwendet werden
httpDelete(url)	führt einen HTTP DELETE Request mit der gegebenen Ressource (Dateiname) aus
startHTTPServer(handler)	startet einen HTTP Server (Webserver auf Port 80), der auf HTTP GET Requests hört. Bei einem GET Request wird die benutzerdefinierte Callbackfunktion handler(clientIP, filename, params) aufgerufen. clientIP: gepunktete IP-Adresse des Clients filename: Dateiname der URL mit vorgestelltem "/". Fehlt der Dateiname: "/" params: Dictionary mit den GET Request Parametern {key: value}. Beispiel: Für die URL http://192.168.0.101/on?a=ok&b=3 ist filename = "/on" und params = {"a" : "ok", "b" : "3"} Rückgabe: - ein einzelner Wert (String, Float, Integer): dieser wird unverändert an den Browser zurückgesendet. Es kann sich um eine HTML-Webpage handeln oder um einen einzelnen Float/Integer. Die Webpage darf nicht länger als 250 Zeichen sein - ein Tupel oder eine Liste. Die darin enthaltenden Werte werden in die %-Formatangaben der vorher mit saveHTML() gespeicherten HTML-Standarddatei eingebaut und an den Browser zurückgesendet - keiner: Es wird die vorher mit saveHTML() gespeicherte HTML-Textdatei unverändert an den Browser zurückgesendet Alle HTTP-Replies werden mit einem Header 200 OK versehen. Die Funktion ist blockierend. Um wieder in den Kommandomodus zu kommen, muss der Micro:LinkUp neu gebootet werden

saveHTML(text)	liefert die GET Request Parameter als Dictionary zurück {key : value}
sendReply(reply)	speichert den Text auf dem LinkUp als eine HTML-Standarddatei. Sie kann %-Formatangaben enthalten, die mit den Rückgabewerten der Callbackfunktion handler ersetzt werden. Ist text leer, so wird die Standarddatei gelöscht

Modul mqtt
(Micro:LinkUp ESP32 coprocessor breakout am I2C-Port)
Modul import: import mqtt

Funktion	Aktion
mqtt.connectAP(ssid, password)	verbindet mit dem bestehenden Access-Point (Hotspot, Router) mit ssid und password. Gibt die erhaltene gepunktete IP-Adresse zurück; leer, falls das Einloggen misslingt
mqtt.broker(host, port = 1883, user = "", password = "", keepalive = 0)	legt die Eigenschften des Brokers fest (IP-Adresse, IP-Port und falls nötig Authentifizierungsangaben). keepalive legt fest, wie lange die Verbindung ohne Datenaustausch offen bleibt (in sec) (default ist abhängig vom Broker). Es wird noch keine Verbindung zum Broker hergestellt
mqtt.connect(cleanSession = True)	erstellt eine Verbindung zum Broker. Für cleanSession = True, werden alle früheren Daten gelöscht. Gibt True zurück, falls die Verbindung zustande gekommen ist
mqtt.ping()	sendet einen Ping-Request an den Server, damit dieser die Verbindung offen hält
mqtt.publish(topic, payload, retain = False, qos = 0)	sendet zum gegebenen Topic eine Message (payload). Falls *retain = True* wird diese Message als die letzte good/retain Message betrachtet. qos ist der Quality of Service level (nur 0, 1 unterstützt). Gibt True zurück, falls erfolgreich; andernfalls False
mqtt.subscribe(topic, qos = 0	abonniert das gegebene Topic mit dem gegegeben qos level (nur 0, 1 unterstützt). Es werden maximal 50 erhaltende Message-Tupels (topic, payload) in einem Messagebuffer der Reihe nach gespeichert (maximale Längen topic: 50, payload: 200 bytes)
topic, payload = mqtt.receive()	holt das erste Element des Messagebuffers (das "älteste") als Tupel (topic, payload) zurück und entfernt das Element aus dem Buffer. Falls keine Daten im Buffer sind, wird (None, None) zurückgegeben. Die Pollperiode sollte mindestens 1 sec betragen
mqtt.disconnect()	schliesst die Verbindung

Modul ntptime
(using Micro:LinkUp ESP32 coprocessor as I2C slave)
(**Modul import: import ntptime**)

ntptime.getTimeRaw(server = "pool.ntp.org"	gibt die aktuelle Datumzeit zurück, die vom gegebenen Server abgegeben wurde. Format: Tupel (yyyy, mm, dd, h, m, s, week_day, day_of_year) alles Ints. Zeit ist GMT
ntptime.getTime(server = "pool.ntp.org")	dasselbe, aber es wird ein formatierter String zurückgegeben (Beispiel: "Tu 2019-06-11 13:04:44 GMT")

Modul sht
Sensirion Temperatur- und Luftfeuchtesensor am I2C-Port
Modul import: import sht

sht.getValues()	gibt ein Tupel mit Temperatur (in Grad Celsius) und Luftfeuchtigkeit (in Prozent) zurück

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

sgp.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)

Modul mcp9808
(Microchip Temperatursensor am I2C-Port)
(**Modul import: from mcp9808 import MCP9808**)

mcp = MCP9808()	erzeugt eine Sensorinstanz
mcp.temperature()	gibt die Temperatur (in Grad Celsius) zurück

Realtime Clock (RTC)
(RTC Clock Modul DS3231 am I2C-Port (Adresse 0x68))
Modul import: import rtc

Funktion	Aktion
rtc.set(yy, mm, dd, h, m, s, w)	setzt Jahr, Monat, Tag, Stunde, Minute, Sekunde, Wochentag (üblich 0: Montag)
rtc.set([yy, mm, dd, h, m, s, w])	dasselbe mit Liste (oder Tupel)
rtc.get()	gibt ein Tupel (yy, mm, dd, h, m , s, w) zurück (alles ints)

Modul bme280
(Temperatur-, Luftfeuchtesensor und Luftdrucksensor von Bosch am I2C-Port)
Modul import: from bme280 import BME280

bme = BME280()	erzeugt eine Sensorinstanz
bme.temperature()	gibt die Temperatur (in Grad Celsius) zurück
bme.humidity()	gibt die Luftfeuchtigkeit (in Prozent) zurück
bme.pressure()	gibt den Luftdruck (in hPa) zurück
bme.altitude()	gibt die Höhe über Meer (mit Luftdruck 1013.25 hPa auf Meereshöhe) zurück
bme.all()	liefert Temperatur, Luftdruck, Luftfeuchtigkeit, Höhe über Meer gleichzeiig in einem Tupel
bme.set_qnh()	setzt den Luftdruck auf Meereshöhe (in hPa)

ÜBER DIE AUTOREN

Jarka Arnold war als Dozentin an der Pädagogischen Hochschule Bern für die Informatikausbildung angehender Lehrkräfte für die Sekundarstufe 1 tätig. Sie hat dabei Informatikgrundkonzepte und das Programmieren mit Java, PHP und Python vermittelt. Ihre langjährige Erfahrung in der Aus-und Weiterbildung von Informatiklehrpersonen und viele Musterbeispiele sind in diesen Lehrgang eingeflossen. Sie ist zudem verantwortlich für den Webauftritt dieses Lehrgangs.

Aegidius Plüss war an der Universität Bern Professor für Informatik und deren Didaktik und hat in dieser Tätigkeit viele Informatiklehrkräfte aus- und weitergebildet, die heute aktiv an den Schulen tätig sind. Auf der Grundlage seiner grossen Erfahrungen mit vielen Programmiersprachen, Computersystemen und Elektronik hat er die didaktischen Bibliotheken und die notwendigen Tools für diesen Lehrgang entwickelt.

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