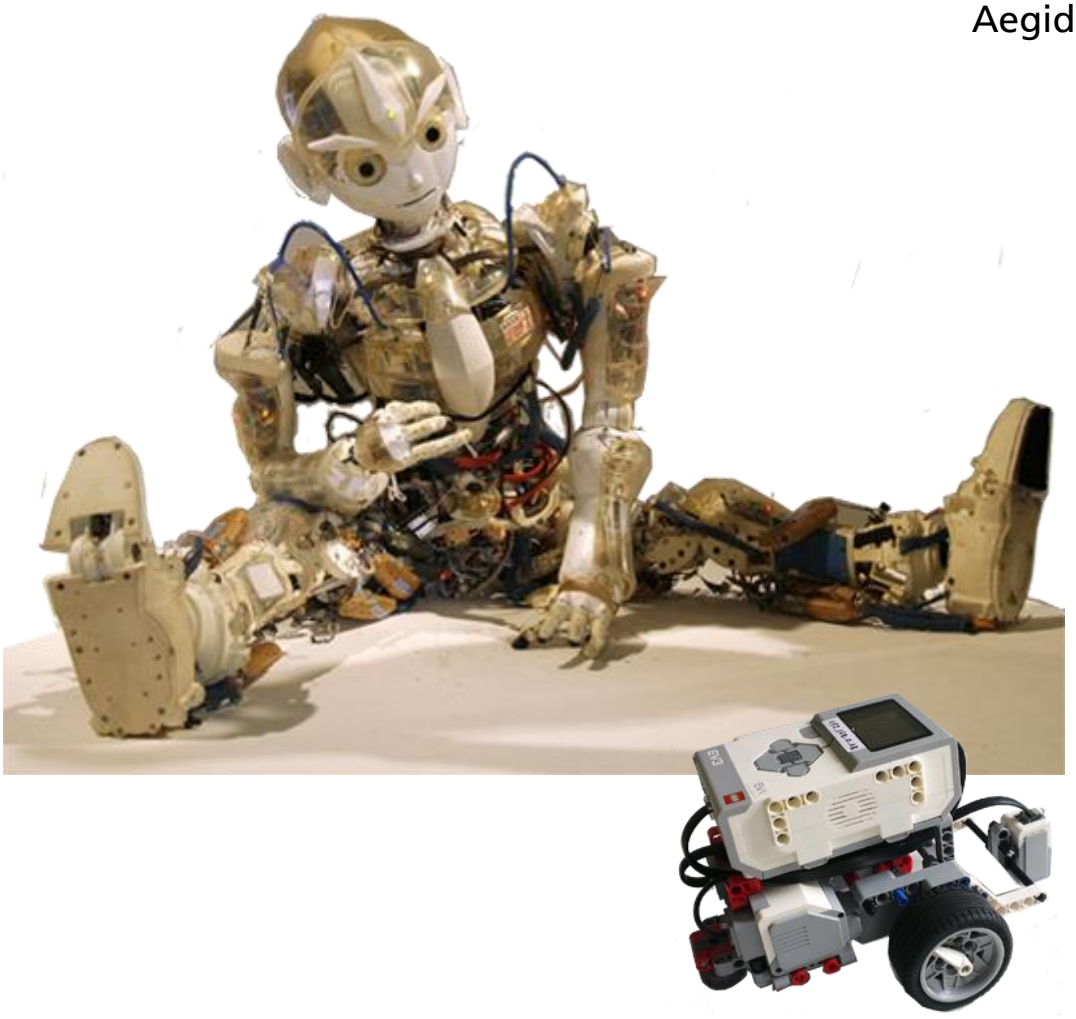
**TigerJython4Kids**

www.tigerjython4kids.ch

# Teil 2: ROBOTIK

Jarka Arnold
Aegidius Plüss



## mit TigerJython und Lego EV3

www.tigerjython4kids.ch

**INHALT**

**Teil 2: ROBOTIK**



**ANHANG**:






Version 2.1, Mai 2020



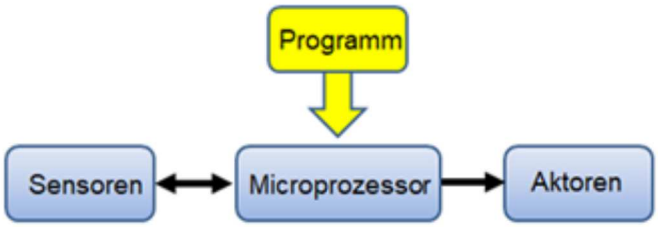
Autoren: Jarka Arnold, Aegidius Plüss
Kontakt: help@tigerjython.com

# 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.

# LEGO EV3 ROBOTER

Die Entwicklungsumgebung TigerJython zusammen mit der neuen Befehlsbibliothek ***grobot*** eignet sich sehr gut für die Einführung in die Robotik mit LEGO EV3. Die Bibliothek ***grobot*** ist eine einfachere Version der Bibliothek ***ev3robot***, die bereits seit mehreren Jahren für die EV3-Programmierung verwendet wird.

Obwohl sich die EV3-Roboter hervorragend für die Einführung in das objektorientierte Programmieren eignen (siehe Lernprogramm Robotik mit EV3), ist der Einstieg in die Robotik mit globalen Befehlen wesentlich einfacher. Darüber hinaus sind Programmcodes weitgehend identisch mit den Programmen für die mbRoboter (Roboter mit micro:bit). Im Klassenverband können daher die beiden Robotermodelle gleichzeitig eingesetzt werden.

Die Bibliothek *grobot* enthält auch die notwendigen Tools, mit welchen du Anwendungen in den hoch aktuellen Bereichen wie "Vernetzte Roboter" und "IoT (Internet of Things)" programmieren kannst.

## SIMULATION

Falls du dein Programm zuerst auf dem Computer testen möchtest oder keinen EV3 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

Im LEGO Roboterbausatz «Mindstorms EV3 Education» findest du alles, um einen fahrenden Roboter zusammenzubauen. Dabei gibt es viele Varianten. Wir schlagen dir aber vor, dass du ein mit zwei Motoren ausgerüstetes, fahrendes Modell wählst, bei dem sich die Sensoren leicht auf der Vorderseite befestigen lassen. Das Basis-Modell kannst du nach der Anleitung im Roboterbausatz bauen.

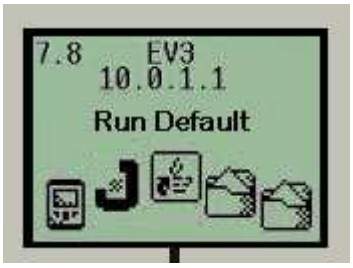
## SD-KARTE ERSTELLEN

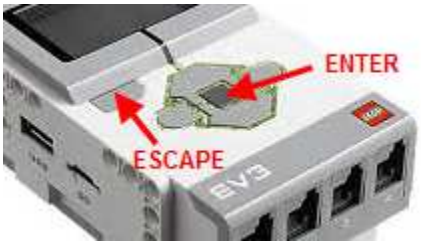
Beim Einschalten des Roboters wird als erstes das Betriebssystem des Mikroprozessors geladen und gestartet. Da wir den Roboter mit Python programmieren, benötigst du ein speziell konfiguriertes Linux-Betriebssystem, das von der SD-Karte geladen wird. Falls dir keine fertige SD-Karte zur Verfügung steht, kannst du sie selbst erstellen. Kopiere die img-Datei Ev3card.img (Download: tigerjython.ch/downlad/ev3sdcard.zip) auf eine leere micro-SD Card, mit 8 GB und maximal 32 GB Speicherkapazität (Typ class 10).

## EV3 STARTEN

Vergewissere dich, dass die SD-Karte bereits im EV3 eingeschoben ist und schalte ihn mit der ENTER-Taste ein. Nach kurzer Zeit siehst du auf dem Bildschirm das *"leJOS"*-Menü.

.




Für die Bedienung verwendest du die 4 Cursortasten, die zentrale ENTER-Taste und die ESCAPE-Taste oben links. Du kannst dich bereits ein bisschen daran gewöhnen. Zum Ausschalten drückst du beispielsweise zuerst ESCAPE, dann gehst du mit der Cursortaste RIGHT auf ✓ und bestätigst die Auswahl mit der ENTER-Taste.

## PC UND EV3 VERBINDEN

Die Programme für den EV3 werden auf einem Computer entwickelt und dann via WLAN oder via Bluetooth. Diese Verbindung musst du zuerst einrichten.

Um mit dem EV3 via WLAN zu kommunizieren, benötigst du einen USB-WLAN-Adapter. Wir empfehlen "**Edimax EW 7811UN**". Den Adapter musst du vor dem Einschalten des Roboters in den USB-Port des EV3 einstecken.

Der Computer und der EV3 müssen Zugang zum gleichen **WLAN Accesspoint** haben. Die **SSID** und **Passwort** des Accesspoints wählst du im EV3-Menü unter WiFi. Die **IP-Adresse**, die dein Roboter vom Accesspoint erhält, wird danach und jeweils beim Einschalten des EV3 auf dem Display angezeigt.

Um Bluetooth-Verbindung zu erstellen, musst du die beiden Geräte paaren (EV3 Key: 1234) und eine "Verbindung über Zugriffspunkt" erstellen. Eine genauere Beschreibung der Kommunikation findest du auf der Webseite.

## MEHRERE EV3 IM KLASSENVERBAND

Wenn man in einem Klassenverband mehrere EV3 verwendet und Programme via Bluetooth herunterladen will, muss man den Robotern verschiedene Bluetooth-Namen geben. Die Umbenennung wird auf dem EV3 unter dem Menüpunkt *System - Change name* vorgenommen. Nach dem Umbenennen muss der EV3 neu gestartet werden. Der neue Name erscheint dann auf dem EV3-Display.

Beim Programm-Download über das WLAN empfiehlt sich einen billigen WLAN-Router (muss nicht mit dem Internet verbunden sein) zu beschaffen und die Roboter und auch die EntwicklungsPCs mit diesem Router verbinden.

# 1. LOSLEGEN

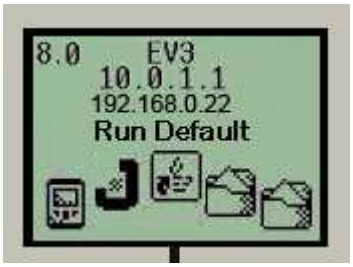
## DU LERNST HIER...

wie du den EV3 bedienst und ein erstes Roboterprogramm erstellst, dass du auf den EV3 hinunterlädst und dort ausführst.

## DAS ERSTE PROGRAMM AUSFÜHREN

Schiebe die SD-Karte in den Karten-Slot und starte den EV3 mit der ENTER-Taste. Nach kurzer Zeit erscheint das EV3-Startmenü. Die Bluetooth-Verbindung mit der IP-Adresse 10.0.1.1 ist standardmässig aktiviert. Falls du eine WLAN-Verbindung eingerichtet hast, erscheint darunter die IP-Adresse, die dein EV3 vom Accesspoint zugeteilt erhält.

Mit nochmaligen Drücken der ENTER-Taste startest du den *BrickGate-Server*, welcher für die Ausführung der Python-Programme auf dem EV3-Brick notwendig ist.





Du siehst jetzt das Menü des BrickGate-Servers. Der Server kann mehrere Programme speichern. Das Programm *HelloPython* ist bereits vorhanden. Falls mehrere Programme gespeichert sind, wählst du das auszuführende Programm mit den Cursor-Tasten.

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:

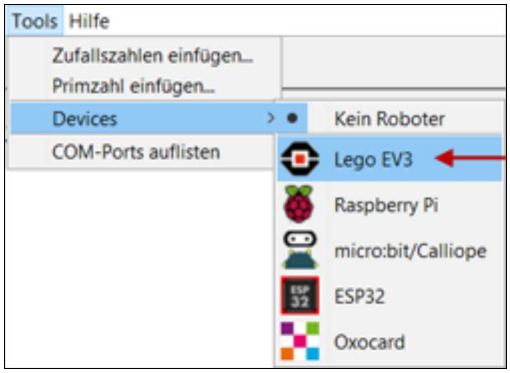
```
from grobot import *

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

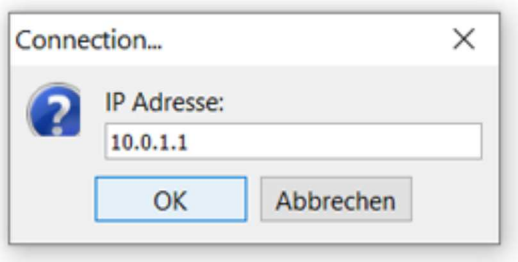
Mit der ersten Zeile  from grobot import * Importierst du die Robotik-Bibliothek *grobot*. Mit dem Befehl forward() wird der Roboter in die Vorwärtsbewegung versetzt und bleibt 2000 Millisekunden in diesem Zustand. Mit dem Befehl stop() wird er angehalten.

Um das Programm via Bluetooth oder WLAN auf den EV3 herunterladen muss du folgende Einstellungen vornehmen:

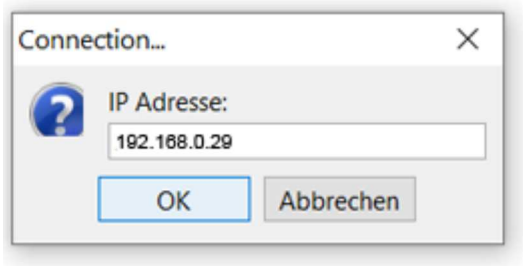
Wähle unter *Tools / Devices / Lego EV3*.

Danach erscheint eine Dialogbox, in der du die IP-Adresse deines Roboters eingeben musst.



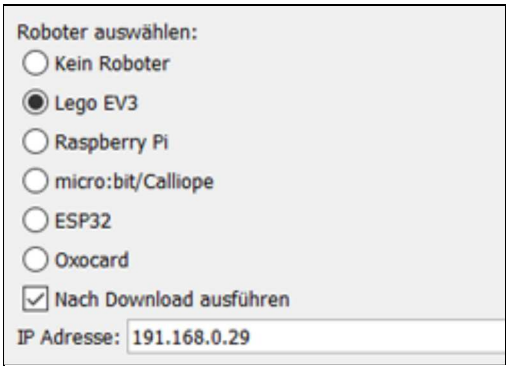




Für die Verbindung über **Bluetooth** wird standardmässig 10.0.1.1 für alle EV3 Roboter verwendet. Falls man im gleichen Raum mehrere Roboter verwendet, müssen sie verschiedene Namen haben.




Bei der Verbindung über das **WLAN** musst du die IP-Adresse, die deinem Roboter vom Accesspoint zugeteilt wurde (z.B. 192.168.0.29) eingeben.

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".

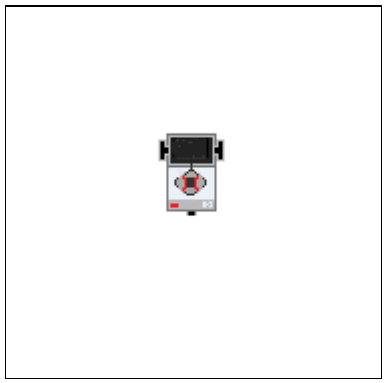
Der Roboter fährt eine kurze Strecke vorwärts. Der Programmname erscheint auch auf dem EV3-Display und kann jeder Zeit erneut ausgeführt werden.

Falls es nicht funktioniert kontrolliere die Verbindung zum Computer. Vergiss nicht, dass bei der Verbindung über das WLAN dein Computer am gleichen Accesspoint wie dein Roboter angemeldet werden muss.

Während der Programmausführung wird auf deinem Computer ein **Terminalfenster** angezeigt, in dem Fehlermeldungen vom EV3 angezeigt werden. Mit **Schliessen des Terminalfensters** kannst du ein Programm, jederzeit **abbrechen**.

## SIMULATIONSMODUS

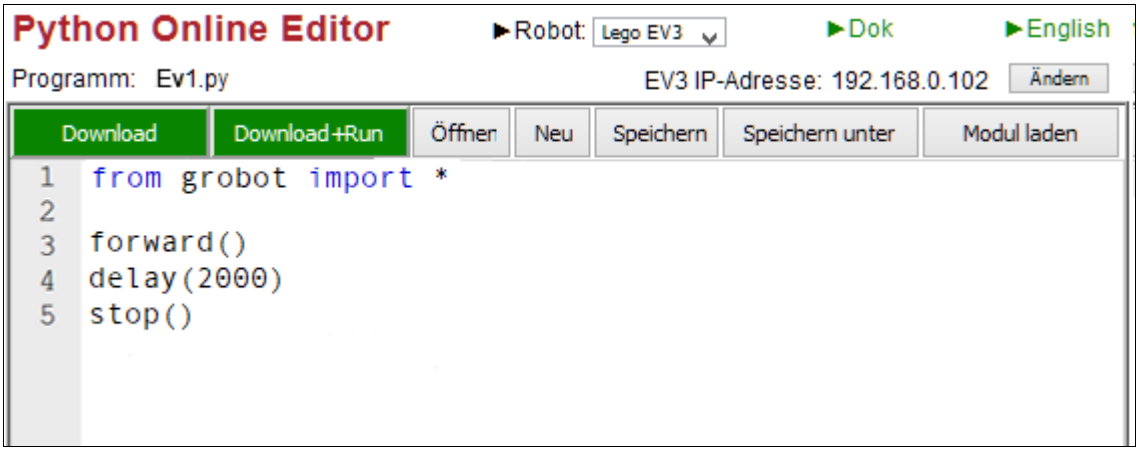
Die meisten Programme können auch im Simulationsmodus ausgeführt werden. Du verwendest das gleiche Programm, zum Ausführen klickst du aber anstelle des Robotik-Buttons auf den grünen Run-Pfeil.

## ONLINE-EDITOR

Als Alternative zum TigerJython-Editor kannst du die EV3-Roboter auch mit unserem Online-Editor programmieren, ohne eine lokale Entwicklungsumgebung installieren zu müssen. Die Programme werden in einem Browser-Fenster editiert und dann über das WLAN auf den Roboter heruntergeladen. Du kannst also sofort auf allen Plattformen, auch auf **Tablets** und **iPads**, mit Programmieren loslegen. Das setzt allerdings voraus, dass du eine Verbindung über das WLAN eingerichtet hast.

Mit Klick auf den Link [Online-Editor], der bei allen Beispielen angezeigt wird, wird der Programmcode in den Online-Editor kopiert.




Mit Klick auf "Ändern" gibst du die IP-Adresse deines Roboters ein. Diese bleibt für weitere Anwendungen auf deinem Computer gespeichert. Die Simulation wird mit Online-Editor nicht unterstützt.

## MERKE DIR...

Bevor du das erste Programm ausführen kannst, musst du nach dem Einschalten des EV3 noch einmal den ENTER-Button klicken, um den *BrickGate-Server* zu starten und eine Verbindung zwischen deinem Computer und EV3 erstellen.

Du gibst den Programmcode im TigerJython ein und klickst auf den Roboter-Button um das Programm auf den EV3 herunterzuladen und auszuführen. Bei der Simulation startest du die Programmausführung mit dem grünen Pfeil. Alle Beispiele können auch mit dem Online-Editor editiert und auf den EV3 heruntergeladen werden.

# 2. ROBOTER BEWEGEN (AKTOREN)

## DU LERNST HIER...

wie du die Motoren und LEDs deines Roboters programmieren kannst.

## MUSTERBEISPIELE

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

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

Für die Steuerung des Roboters kannst du auch sogenannte **blockierende** Befehle verwenden. Mit forward(2000) fährt der Roboter während 2000 Millisekunden vorwärts, mit left(550) dreht er während 550 Millisekunden nach links.

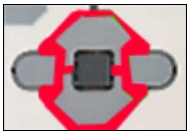
```
from grobot import *

forward(2000)
left(550)
forward(2000)
```

Ein Nachteil der blockierenden Befehle, wie z.B. *forward(2000)* ist, dass der Roboter während 2000 ms keine weitere Befehle entgegen nehmen kann. Mit *forward()* (ohne Zeitangabe) wird der Roboter in die Vorwärtsbewegung versetzt und bleibt in diesem Zustand so lange bis er einen Befehl erhält, den Zustand der Motoren zu ändern. Während dieser Zeit kann er andere Befehle entgegen nehmen, z. B. Sensorwerte zurückmelden.

**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(2) die LEDs auf rot, nach 3000 ms mit setLED(1) auf grün geschaltet und nach weiteren 1000 ms mit setLED(0) ausgechaltet. Danach erhalten die Motoren den Befehl *stop()* und der Roboter hält an.



```
from grobot import *

forward()
delay(4000)
backward()
setLED(2)
delay(3000)
setLED(1)
delay(1000)
setLED(0)
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 (0 bis 100, default speed ist 30).

```
from grobot 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 |

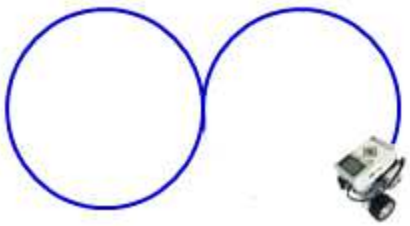
| | |
|---|---|
| stop() | hält an |
| setSpeed(speed) | setzt die Geschwindigkeit |
| delay(ms) | bleibt ms Millisekunden im gleichen Zustand |
| setLED(n) | LEDs einschalten (1 grün, 2 rot, 3 orange) |
| setLED(0) | LEDs ausschalten |
| setAlarm(1) | Schaltet einen Pipton ein |
| setAlarm(0) | Schaltet den Pipton aus |

**Programmausführung abbrechen**:
Am einfachsten geht es mit Schliessen des Terminalfensters. Du kannst auch die Buttons ENTER und DOWN gleichzeitig drücken, nur wird dabei auch das Brickgate-Programm abgebrochen und musst ihn danach mit ENTER-Buttons neu starten.

## 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.

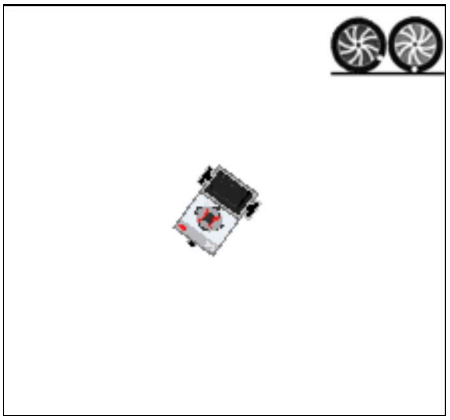


3. Mit **setAlarm(1)** wird ein Alarm eingeschaltet (ein Pipton). **setAlarm(0)** schaltet den Alarm aus. Der Roboter fährt zuerst 5000 ms vorwärts und danach 5000 ms rückwärts. Während des Rückwärtsfahrens soll er Alarm einschalten.

## ZUSATZSTOFF: MOTOREN 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 3 Sekunden mit der Geschwindigkeit 30 rückwärts. Anschliessend hält er an. Führe das Programm zuerst im Simulationsmodus aus und beobachte die Rotationen im Grafikfenster.



```
from grobot import *

motL.rotate(50)
delay(3000)
motL.rotate(-30)
delay(3000)
motL.rotate(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?

# 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.

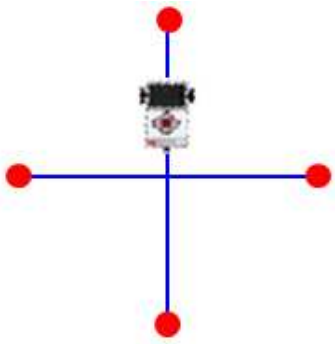
```
from grobot import *

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 grobot import *

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

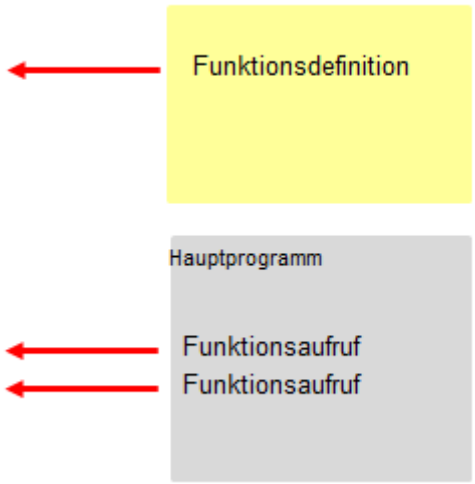
Hier ist es nützlich zu wissen, wie man ein laufendes Programm abbrechen kann: Am einfachsten geht es mit Schliessen des Terminalfensters. Du kannst ein Programm auch durch gleichzeitiges Drücken von ENTER und DOWN auf dem EV3-Brick abbrechen. Dabei wird aber auch das Brickgate-Programm beendet und muss erneut mit dem ENTER-Button gestartet werden.

**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 grobot import *

def blink():
    setLED(2)
    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:

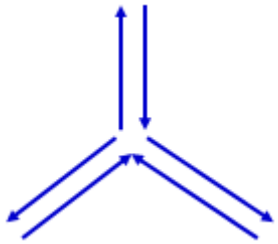
```
repeat:
    Anweisungen
```

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

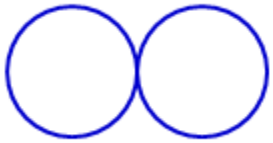
Mit **Funktionen** kannst du deine Programme besser strukturieren und Code-Dupplikation vermeiden.

## ZUM SELBST LÖSEN

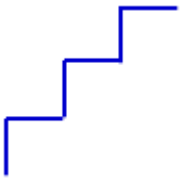
1. Der Roboter startet in der Mitte, fährt eine kurze Strecke vorwärts und dann rückwärts zurück zum Ausgangspunkt. Dann dreht er etwa um 120° nach rechts. Diese Befehlssequenz wiederholt er drei Mal.



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.



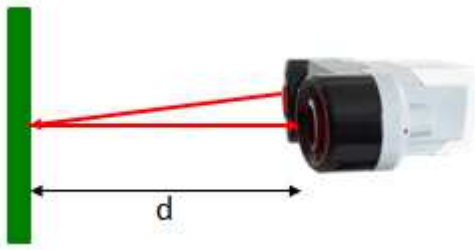
# 4. ULTRASCHALLSENSOR (DISTANZSENSOR)

## 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 Ultraschallwellen. Bei der 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 EV3-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 und anhalten**
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. Schliesse den Ultraschallsensor am Sensorport 1 an

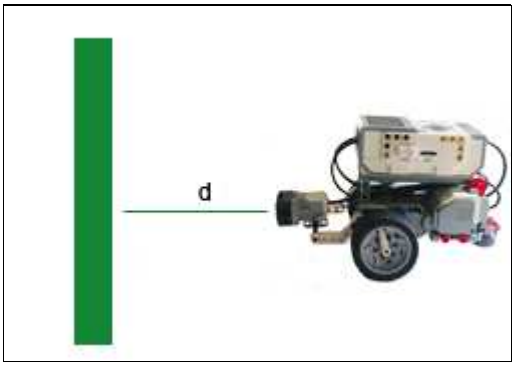
```
from grobot import *

forward()
repeat:
    d = us1.getDistance()
    print(d)
    if d < 20:
        stop()
    delay(200)
```

Ist der Sensor am Port 1 angeschlossen, gibt der Befehl us1.getDistance() 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 grobot import *

forward()
setSpeed(10)
repeat:
    d = us1.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 grobot import *

def searchTarget():
   repeat:
      right()
      delay(50)
      dist = us1.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.

## MERKE DIR...

Die Anweisung *us1.getDistance()* gibt den Wert des Ultraschalsensors, der am Sensorport 1 angeschlossen ist 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.

Mit ***break*** kannst du eine Schleife (auch Endlosschleife) abbrechen. Das Programm wird danach mit der nachfolgenden Zeile fortgesetzt.

## 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:
   ...
```

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

2b. Ein Roboter mit einem Ultraschallsensor soll ein Objekt finden, danach zu ihm fahren und wenige cm vor ihm anhalten

2c. Der Roboter soll nach einander drei Säulen finden und alle umstossen.

3. 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 der Objekte (target) im Grafik-Fenster angegeben. Für die Simulation musst du den Sensor immer am Port 1 anschliessen.

**Beispiel 4: Ultraschallsimulation**

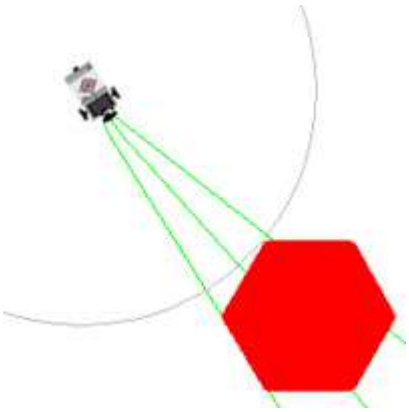
Der Roboter soll zum Hindernis fahren. Wenn die Distanz kleiner als 30 cm ist, fährt er eine kurze Strecke rückwärts und danach wieder vorwärts. Für die Simulation verwendest als Hindernis einen horizontalen Balken, den du mit folgendem Context erzeugst:

```
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 grobot import *

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

forward()
repeat:
    d = us1.getDistance()
    if d < 30:
        backward()
        delay(2000)
    else:
        forward()
    delay(100)
```

**Beispiel 5: Ultraschalsimulation** (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 grobot 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 = us1.getDistance()
      if dist != -1: #real != 255
         right()
         delay(600)
         break

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

repeat:
   dist = us1.getDistance()
   print(dist)
   if dist < 30:
      stop()
      break
```

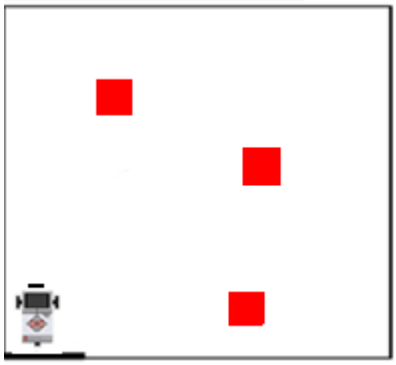
Das Objekt ist mit den Koordinaten eines Sechsecks mit dem Mittelpunkt (0, 0) beschrieben und mit dem Bild *redtarget.gif* an der Position (400, 400) dargestellt wird.

## MERKE DIR...

Die Beispiele mit einem 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 gibt 255 zurück). Sonst kannst du die Programme ohne Änderung im Realmodus ausführen. Die Zeilen mit *RobotContext* werden im Realmodus nicht berücksichtigt.

## ZUM SELBST LÖSEN

4. Löse die Aufgabe 2c) im Simulationsmodus. Ein Roboter soll mit seinem Ultraschallsensor soll drei Gegenstände nach einander finden. Er dreht am Ort und wenn er ein Objekt detektiert, fährt er los und bleibt kurz vor stehen. Danach sucht er das nächste Objekt. Im Simulationsmodus kannst du *squaretarget.gif* verwenden, um die Gegenstände darzustellen. Verwende die folgenden Vorlage:



```
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)
```

# 5. LICHTSENSOR

## DU LERNST HIER...

wie der EV3 mit seinen Lichtsensoren die Helligkeit der Unterlage messen kann.

## WIE FUNKTIONIERT EIN LICHTSENSOR?

Der Color-Sensor, der im EV3-Bausatz mitgeliefert wird, kann Farben erkennen und auch als Lichtsensor die Helligkeit der Unterlage messen.

Er ist mit einer roten Leuchtdiode (LED) und einer Fotodiode ausgestattet. Die LED beleuchtet die darunterliegende Fläche, die Fotodiode misst die Intensität des reflektierten Lichts. Die Sensorwerte liegen zwischen 0 und 1023 (je heller umso grösser ist der Wert).

## MUSTERBEISPIELE

### Beispiel 1: Kante folgen

Der Roboter mit einem Lichtsensor soll am Rand einer dunklen Fläche fahren. Je nachdem, ob er schwarz oder weiss "sieht", korrigiert er die Fahrtrichtung mit einem Links- oder Rechtsbogen. Die Sensorwerte werden in einer Endlos-Schleife mit der Messperiode von 100 ms abgefragt.

Mit der Anweisung v = ls3.getValue() wird die Intensität des Lichts in der Variablen v gespeichert (ls3 bedeutet, dass der Lichtsensor am Port 3 angeschlossen ist).

```
from grobot import *

repeat:
   v = ls3.getValue()
   if v < 500:
      rightArc(0.2)
   else:
      leftArc(0.2)
   delay(100)
```

Du kannst das Programm auch im Simulationsmodus ausführen. Dazu musst du gerade nach der Importzeite 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).

Im Realmodus werden die Zeilen mit *RobotContext* nicht berücksichtigt. Du kannst das gleiche Programm auch im Realmodus ausführen. 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, brauchst du mehrere Sensoren. In einer stark vereinfachten Version eines autonomen Fahrzeuges verwendest du zwei Lichtsensoren.

Die Messwerte des rechten und des linken Sensors speicherst du in den Variablen vR und vL
vR = ls1.getValue()
vL = ls2.getValue()

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

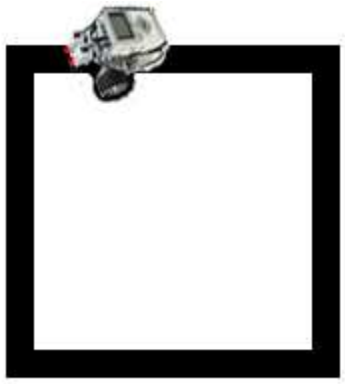
```
from grobot import *

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

setSpeed(20)
repeat:
    vR = ls1.getValue()
    vL = ls2.getValue()
    if vL < 500 and vR < 500:
        forward()
    elif vL > 500 and vR < 500:
        rightArc(0.1)
    elif vL < 500 and vR > 500:
        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. Auch der Schwellenwert 500 muss je nach Helligkeit der Unterlage angepasst werden. Mit *print(vR)* und *print(vL)* kannst du die Messwerte im Terminalfenster anzeigen.

**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 der Lichtsensoren 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 grobot import *

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

def keepOnTrack():
    if vL < 500 and vR < 500:
        forward()
    elif vL < 500 and vR > 500:
        leftArc(0.1)
    elif vL > 500 and vR < 500:
        rightArc(0.1)

repeat:
    vR = ls1.getValue()
    vL = ls2.getValue()
    if vL > 500 and vR > 500:
        left()
        delay(500)
    else:
        keepOnTrack()
    delay(100)
```

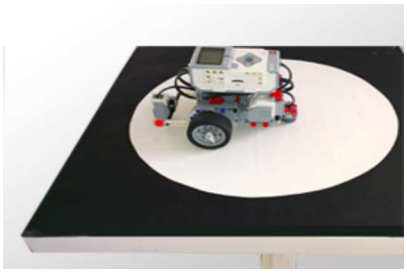
## MERKE DIR...

Mit einem Lichtsensor kannst du die Helligkeit der Unterlage messen. Der Befehl ls1.getValue() liefert den Wert des Lichtsensors, der am Sensorport 1 angeschlossen ist, als Wert zwischen 0 und 1023. Je grösser der Wert, umso heller ist die Unterlage. Mit zwei Lichtsensoren, lässt sich der Roboter beim Fahren auf den Tracks besser steuern.

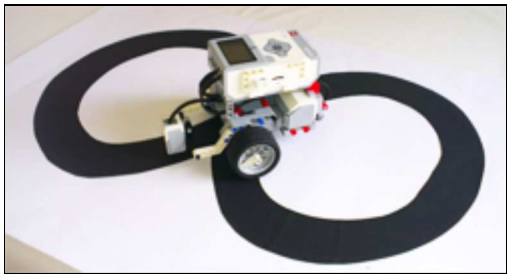
## ZUM SELBST LÖSEN

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 verwendest du

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



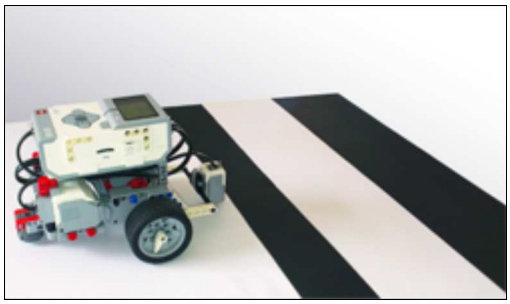
2. Bei Wegen mit Kreuzungen verliert der Roboter oft die Spur, insbesondere, wenn du ihn schneller fahren lässt. Ergänze das Programm aus dem Beispiel 2 für den Fall, dass der Roboter die Spur verliert (hell-hell "sieht"). Überlege, was er in diesem Fall tun kann.



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(490, 450)
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.

## ANMERKUNG

Die Bibliothek *grobot* enthält nur die Lichtsensor-Funktionen des Colorsensors. Falls du den Colorsensor auch für die Farberkennung verwenden willst, musst du die OOP-Bibliothek ***ev3robot*** verwenden (siehe www.jython.ch).

# 6. TOUCHSENSOR (BERÜHRUNGSSENSOR)

## DU LERNST HIER...

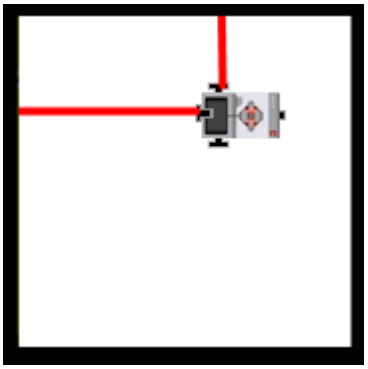
wie du ein einen Touchsensor dafür einsetzen kannst, Hindernisse zu erkennen und entsprechend zu reagieren.

## WIE FUNKTIONIERT EIN TOUCHSENSOR

Der EV3-Touchsensor ist wie ein Schalter aufgebaut, der beim Hineindrücken des roten Knopfs einen Kontakt macht. Gewöhnlich baust du ihn vorne in der Mitte des Roboters an. Der Touchsensor kennt zwei Zustände *gedrückt (isPressed)* und *nicht gedrückt*.

## MUSTERBEISPIELE

### Beispiel 1: Ein Hindernis registrieren und ausweichen



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

Ist der Touchsensor am Port 3 angeschlossen, gibt der Befehl ts3.isPressed()*True* zurück, falls der Sensorknopf Knopf gedrückt ist, sonst liefert er *False*.

```
from grobot import *

RobotContext.useObstacle("sprites/field1.gif", 250, 250)

forward()
repeat:
    if ts3.isPressed():
        backward()
        delay(1500)
        left()
        delay(550)
        forward()
```

Für die Simulation verwendest du ein Bild mit einem transparenten Hintergrund, als Gegenstand, welchen du im Grafikfenster mit dem *RobotContext useObstacle()* und der Angabe der Koordinaten positionieren musst.

**Beispiel 2: Kanalroboter**

Ein Roboter mit zwei Touchsensoren soll den Weg im Kanal finden. Mit zwei Touchsensoren kannst du den Roboter effizienter steuern. Je nachdem, ob er ein Hindernis mit dem linken oder mit dem rechten Sensor berührt, fährt er zuerst kurz zurück und dreht um einen kleinen Winkel in die Gegenrichtung. Die Sensoren sind an den Ports 1 und 2 angeschlossen.

Führe das Programm zuerst im Simulationsmodus aus und baue danach mit verschiedenen Gegenständen eine ähnliche Bahn für den realen Roboter.

```
from grobot import *

RobotContext.useObstacle("sprites/racetrack.gif", 250, 250)
RobotContext.setStartPosition(420, 460)

forward()
repeat:
    if ts1.isPressed():
        backward()
        delay(250)
        left()
        delay(200)
        forward()
    elif ts2.isPressed():
        backward()
        delay(250)
        right()
        delay(200)
        forward()
```

## MERKE DIR...

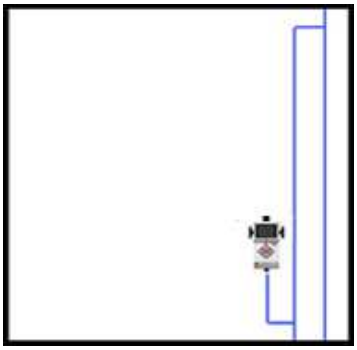
Der Touchsensor kann nur zwei Werte zurückgeben und zwar den Wahrheitswert *True* , wenn der Sensorbutton gedrückt ist, oder *False*, wenn er nicht gedrückt ist. Für die Simulation muss du Hintergrundbilder mit einem transparenten Hintergrund verwenden und diese mit *RobotContext.useObstackle()* hinzufügen.

## ZUM SELBST LÖSEN

1. Ein Roboter mit einem oder zwei Tousensoren bewegt sich im Käfig mit einem Ausgang. Er startet in der Mitte und soll möglichst rasch den Ausgang finden. Für die Simulation benötigst du den folgenden Context:

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

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

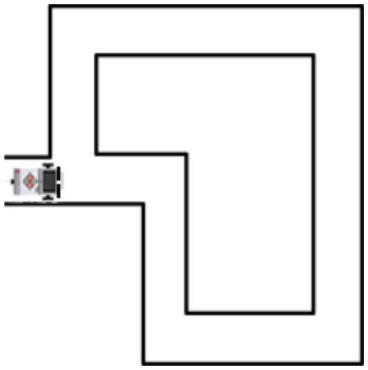


Beachte, dass der Rasenmäher immer einmal links und einmal rechts drehen muss. Um es so zu programmieren, gibt es ein einfacher Trick:

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

```
RobotContext.useObstacle("sprites/field1.gif", 250, 250)
RobotContext.setStartPosition(350, 350)
```

3. Kanalroboter.
Ein Roboter mit zwei Touchsensoren soll das ganze Rohr durchfahren und wieder zum Ausgangsort zurückkehren. Für die Simulation kannst du das Hintergrundbild *channel.gif* verwenden.



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

## ZUSATZSTOFF: BUTTON ESCAPE

### Beispiel 3: Programme mit Button escape abbrechen

Ähnlich wie du die Buttons des Touchsensorens programmieren kannst, können auch die Buttons auf dem EV3-Brick programmiert werden, um z. B. mit dem **Button escape** ein laufendes Programm abzubrechen.


Anstelle einer Endlosschleife *repeat* verwendest du eine *while-Schleife* mit einer Abbruchbedingung. So lange der Button *escape* nicht gedrückt wurde, läuft das Programm. Der Roboter bewegt sich gegen eine Wand. Wenn er sie mit seinem Touchsensor berührt, fährt er kurz zurück und dann wieder vorwärts. Endlos. Bis jetzt hast du solche Programme mit Schliessen des Terminalfensters abgebrochen. Diese **while**-Schleife ermöglicht dir, das Programm mit Drücken des Button *escape* abzubrechen. Dies ist insbesondere dann praktisch, wenn du ein Programm autonom, weit weg vom deinem Computer ausführst.

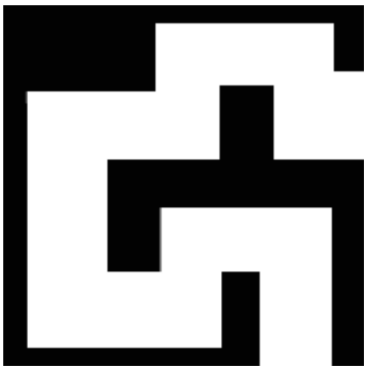
```
from grobot import *

forward()
while not button_escape.was_pressed():

    if ts3.isPressed():
        backward()
        delay(1500)
        forward()
exit()
```

## MERKE DIR...

Anstelle einer Endlosschleife *repeat* kannst du die Schleife
`while not button_escape.was_pressed():`
verwenden. Das Programm wird dann nur so lange ausgeführt, bis du den Button *escape* auf dem EV3 Brick drückst. Die Buttons *enter, left, right, up* und *down* kannst du zur Steuerung des Roboters verwenden.

## ZUM SELBST LÖSEN

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



Löse die Aufgabe zuerst im Simulationsmodus. Anstelle der Brick-Buttons kannst du die Cursortasten left, right und die Esc-Taste verwenden. Vorlage:

```
from grobot import *

RobotContext.useObstacle("sprites/bg2.gif", 250, 250)
RobotContext.setStartPosition(400, 490)

forward()
while not button_escape.was_pressed():
    if ts3.isPressed():
        ....
    if button_left.was_pressed():
        ...
    if button_right.was_pressed():
        ...
exit()
```

# 7. ROBOTER PER WLAN STEUERN

## DU LERNST HIER...

auf dem EV3 einen einfachen Webserver einrichten und mit interaktiven Webseiten den Roboter über das WLAN steuern.

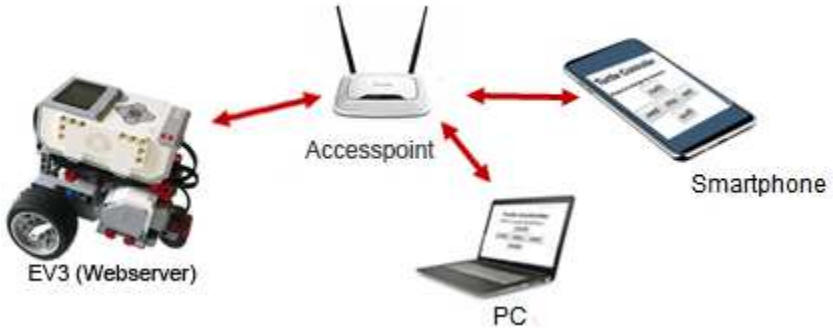
## VERNETZTE ROBOTER

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.

Auch der Microcontroller, der in deinem EV3 eingebaut ist, kann für die WLAN-Steuerung verwendet werden. Mit Tools und Befehlen, die in der Bibliothek *grobot* integriert sind, kannst du auf deinem EV3 sogar einen einfachen Webserver einrichten, so wie es auch bei vielen vernetzten Geräten der Fall ist.

Falls du bis jetzt deine Programme über das Bluetooth herunterladen hast, brauchst du zusätzlich einen USB-WLAN-Adapter (Edimax EW 7811UN), den du vor dem Einschalten des EV3 in die USB-Schnittstelle einstecken musst.


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. Die SSID und Passwort des Accesspoints stellst du im EV3-Menü unter WiFi ein.




Der EV3 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.

## MUSTERBEISPIELE

**Beispiel 1: LEDs ferngesteuert über das WLAN ein- und ausschalten**
Der EV3 erhält seine IP-Adresse vom Accesspoint. Sie ist beim Einschalten des EV3 auf dem Display sichtbar (z.B. 192.168.0.22).

Mit deinem Programm startest du auf dem EV3 einen Webserver und lädst mit dem Befehl saveHTML(html) den HTML-Code einer einfachen 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.

```
from grobot 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="on">Light ON</a></p>
     <p><a href="off">Light OFF</a></p>
  </body>
</html>
"""
print("Saving HTML")
saveHTML(html)

def onRequest(clientIP, state, params):
    if state == "/on":
        setLED(2)
    elif state == "/off":
        setLED(0)
startHTTPServer(onRequest)
while not button_escape.was_pressed():
    delay(100)
exit()
```

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. Der Zugriff auf die Webseite erfolgt Eventgesteuert. Wenn der Benutzer die IP-Adresse des Servers wählt, wird die Funktion onRequest() aufgerufen. Je nach dem, welchen Link der Benutzer wählt, wird als sogenannter URL-

Parameter state "/on" bzw. "/off" zurückgeschickt und der Server führt den entsprechenden Befehl aus.

Teste das Programm mit deinem Smartphone. Dazu musst du unter WLAN-Verbindung den gleichen Accesspoint wie bei EV3 einstellen und im Browser die IP-Adresse deines Roboters mit anschliessendem "**:81**" eingeben (z.B. 192.168.0.22:81).

**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 grobot 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)

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

startHTTPServer(onRequest)
while not button_escape.was_pressed():
    delay(100)
exit()
```

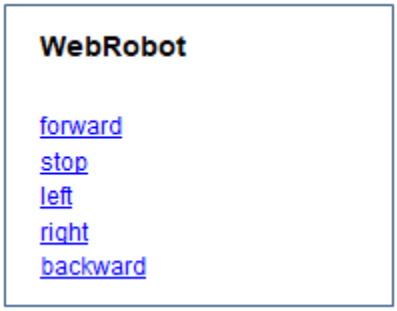
Wenn der Benutzer auf den Link "forward" klickt, erhält der Parameter state den Wert "/forward". In der Funktion onRequest() legst du fest, welchen Befehl der Roboter in diesem Fall ausführen muss.

## MERKE DIR...

Auf dem EV3 kann ein Webserver eingerichtet werden, der eine interaktive Webseite speichern und zur Verfügung stellen kann. Wählt ein Benutzer die IP-Adresse deines Roboters, erstellt er eine WLAN-Verbindung zum Webserver und die Webseite wird im Browser seines Smartphones/PC angezeigt. Mit Klick auf die Links werden als sogenannte URL-Parameter Informationen zurück zum Server geschickt. Im Programm, das auf dem EV3 läuft, legst du fest, welche Aktionen der Roboter ausführen soll.

## 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.




## 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 Button haben den Namen 'btn', mit der Beschriftung erhält jeder Button seinen Wert. Der ersten Button value "forward" , der zweite Button value "stop".

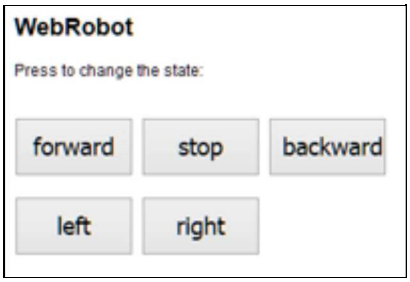

Die Informationen werden wieder im URL übermittelt. In der Funktion onRequest() speicherst du den Wert des Parameters "btn" in der Variablen state und legst mit *if-elif* fest, welche Aktion der Roboter ausführen soll. Der zweite Teil des Programms ist gleich wie im Beispiel 2.

```
from grobot 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)

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

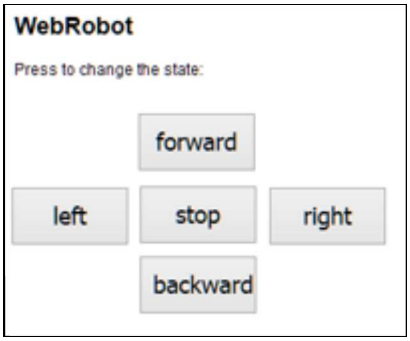
```
        if state == "forward":
            forward()
        elif state == "stop":
            stop()
state = "stop"
setSpeed(40)
startHTTPServer(onRequest)
while not button_escape.was_pressed():
    delay(100)
exit()
```

## 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 EV3-Roboters per WLAN zur Verfügung stellen, ähnlich wie Milliarden IoT-Geräten, die weltweit miteinander und mit übergeordneten Systemen kommunizieren.

## WAS IST IOT?

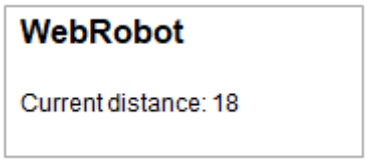
Das IoT ist eines der aktuellsten Themen der heutigen Informatik. Durch Einbau von kleinen, billigen aber WLAN-fähigen Microcontrollern können Geräte und Systeme (kurz Dinge) über ein Netzwerk Daten und Informationen untereinander und mit übergeordneten Systemen kommunizieren. Im Zusammenhang mit IoT 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 grobot 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)

def onRequest(clientIP, state, params):
   d = us3.getDistance()
   return [d]

startHTTPServer(onRequest)
print("Server starting")
while not button_escape.was_pressed():
    delay(100)
exit()
```

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.

Wenn ein Benutzer im Webbrowser die IP-Adresse des Roboters + :81 (z.B. 192.168.0.22:81) wird die Webseite mit der aktuell gemessenen Distanz angezeigt. Die Kommunikation erfolgt wieder über den Port 81, da der Port 80 für den Programmdownload reserviert ist.

## Hochpräzise Umweltsensoren

Da Lego EV3 das $I^2C$-Protokoll für die Kommunikation mit den Sensoren unterstützt, können auch hochpräzise Sensoren an den EV3-Sensorports angeschlossen werden. Die TigerJython-Robotik-Bibliothek unterstützt

- Sensirion SHT31 Temperatur- und Feuchtigkeitssensor
- ADXL345 Digitaler 3-Achsen-Beschleunigungssensor
- BME280 Temperatur- Feuchtigkeit und Luftdrucksensor

Bezugsquelle: www.mouser.ch . Diese Sensoren sind nicht so robust verpackt wie die LEGO-Sensoren, dafür viel günstiger.

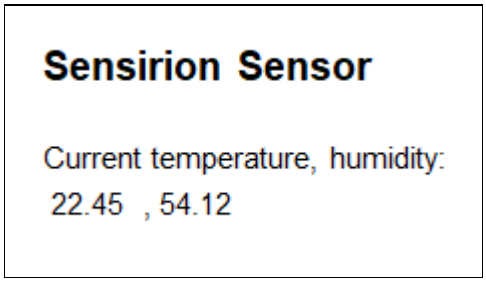
**Beispiel 2: Sensirion Temperatur- und Feuchtigkeitssensor**
Der Sensirion SHT31 ist ein billiger und vielseitig einsetzbarer digitaler Temperatur- und Feuchtigkeitssensor hoher Präzision. Interessant sind solche Messungen insbesondere zusammen mit Datenübertragung, wenn der EV3 die Temperatur und Luftfeuchtigkeit misst und diese Daten über das WLAN an einen Computer oder Smartphone liefert.

Um den Sensor am EV3-Sensorport anzuschliessen, schneidest du einen EV3-Verbindungskabel auf, isolierst die darin enthaltene Kabel ab und lötest das rote Kabel bei GND, das grüne bei VCC, das blaue bei SDA und das gelbe bei SCL an.

So präparierten Sensor kannst du danach z.B. am Sensorport 1 anschliessen.


Der HTML-Code im Kopf deines Programms enthält eine Überschrift und die Anzeige der Sensorwerte. %s, %s sind Platzhalter für die Sensorwerte *temp* und *humi*.

Um die Werte auf deinem Smartphone anzuzeigen, muss du im Browser die IP-Adresse des EV3 mit dem Zusatz ":81" eingeben.




```
from grobot 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)
```

```
def onRequest(clientIP, filename, params):
    temp, humi = sht1.getValues()
    return [temp, humi]

startHTTPServer(onRequest)
print("Server running")
while not button_escape.was_pressed():
    delay(100)
exit()
```

## 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 dem EV3 kannst du IoT-Anwendungen programmieren.

## ZUM SELBST LÖSEN

1. Programmiere eine IoT-Anwendung, bei der der EV3 die Werte des Lichtsensors 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 jeder Zeit mit deinem Smartphone den Zustand überprüfen. Wenn der Roboter ein Objekt in der Nähe detektiert, schaltet er Alarm ein und stellt auf einer Webseite, die aktuelle Distanz dar.

# Dokumentation EV3 Robotik mit globalen Funktionen

Autonomer und Simulationsmodus (im Simuationsmodus implementiert: [S])

**Modul import: from grobot import ***
**Roboter mit Fahrwerk (2 synchronisierte Motoren an MotorPort A und B):**

| Funktion | Aktion |
|---|---|
| forward() [S] | versetzt den Roboter in die Vorwärtsbewegung |
| forward(time) [S] | fährt während gegebener Zeit (in ms) vorwärts, blockierende Methode |
| backward() [S] | versetzt den Roboter in die Rückwärtsbewegung |
| backward(time) [S] | fährt während gegebener Zeit (in ms) rückwärts, blockierende Methode |
| left() [S] | versetzt den Roboter in die Linksdrehung |
| left(time) [S] | dreht während gegebener Zeit (in ms) nach links, blockierende Methode |
| right() [S] | versetzt den Roboter in die die Rechtsdrehung |
| right(time) [S] | dreht während gegebener Zeit (in ms) nach rechts, blockierende Methode |
| leftArc(radius) [S] | startet die Linkskurvenbewegung (Radius in m) |
| leftArc(radius, time) [S] | fährt während der gegebener Zeit (ms) auf dem Linksbogen (Radius in m), blockierend |
| rightArc(radius) [S] | startet die Rechtskurvenbewegung (Radius in m) |
| rightArc(radius, time) [S] | fährt während der gegebener Zeit (ms) auf dem Rechtsbogen (Radius in m), blockierend |
| stop() [S] | stoppt die Bewegung |
| setSpeed(speed) [S] | setzt die Geschwindigkeit (0..100). Wird erst beim Aufruf der nächsten Bewegungsfunktion wirksam (Standardgeschwinkigkeit speed=50) |
| delay(time) [S] | hält das Programm während der gegebenen Zeit (in ms) an |
| sleep(time) [S] | dasselbe wie delay(time) |
| setLED(n) [S] | schaltet LEDs: 0: aus, 1: grün, 2: rot, 3: rot hell, 4: grün blinkend, 5: rot blinkend, 6: rot blinkend hell, 7: grün doppelblinkend, 8: rot doppelblinkend, 9: rot doppelblinkend hell |
| setAlarm(n) [S] | schaltet Alarm: 0: aus, 1:ein |
| playTone(freq, duration) [S] | spielt einen Ton mit gegebener Frequenz (in Hertz) und Dauer (in ms) |
| setVolume(volume) | setzt die Lautstärke (0..100). Dieser Wert wird dauerhaft gespeichert |
| drawString(text, x, y) | schreibt einzeiligen Text an Position (x: horizontal, y: vertikal) |
| clearDisplay() | löscht den Bildschirm |
| getIPAddresses() | gibt eine Liste mit allen aktiven IP Adressen zurück |
| getRobot() | liefert die Instanz des globalen Robot zurück (zur Verwendung der OOP-Klassen) |
| reset() [nur S] | setzt den Roboter an die Startposition/Startrichtung |

**Motoren (EV3 Motoren einzeln steuern):**

| Instanzen | Port |
|---|---|
| motA , motB, motC, motD | Motor am entprechenden MotorPort |
| motL = motA, motR = motB | Linker und rechter Motor |

| Methoden | Aktion |
|---|---|
| motX.rotate(speed) [S] | dreht den Motor motX (speed > 0: vorwärts, speed < 0: rückwärts, speed = 0: stop |
| motX.getMotorCount() [S] | gibt den momentanen Stand des Zählers zurück |
| motX.resetMotorCount() [S] | setzt den Zähler auf 0 |
| motX.rotateTo(deg) [S] | bewegt Motor bis Umdrehungswinkel deg (0-360) und stoppt (blockierend). Setzt Zähler auf 0 |
| motX.rotateTo(deg, blocking) [S] | wie rotateTo(deg). Nicht blockierend, falls blocking = False |
| motX.continueTo(deg) [S] | wie rotateTo(deg), aber Zähler wird nicht auf 0 gesetzt |
| motX.continueTo(deg, blocking) [S] | wie rotateTo(deg), blocking), aber Zähler wird nicht auf 0 gesetzt. Nicht blockierend, falls blocking = False |
| motX.continueRelativeTo(deg [S] | wie continueTo(deg), aber count ist Inkrement |
| motX.continueRelativeTo(deg, blocking) [S] | wie continueTo(deg, blocking), aber count ist Inkrement. Nicht blockierend, falls blocking = False |
| motX.isMoving() [S] | gibt True zurück, falls der Motor motX rotiert |

**Buttons:**

| Instanzen | Position |
|---|---|
| button_escape [S] | oben links |
| button_enter [S] | Mitte |
| button_left [S] | Cursor links |
| button_right [S] | Cursor rechts |
| button_up [S] | Cursor aufwärts |
| button_down [S] | Cursor abwärts |

| Methode | Aktion |
|---|---|
| was_pressed() [S] | gibt True zurück, falls der Button seit dem letzten Aufruf gedrückt wurde |
| isEscapeHit() [S] | gibt True zurück, falls der Escape-Button seit dem letzten Aufruf gedrückt wurde |
| isEnterHit() [S] | dasselbe für den Enter-Button |
| isLeftHit() [S] | dasselbe für den Left-Button |
| isRightHit() [S] | dasselbe für den Right-Button |
| isUpHit() [S] | dasselbe für den Up-Button |
| isDownHit() [S] | dasselbe für den Down-Button |

**Lichtsensor (EV3 Colorsensor als Helligkeitssensor):**

| Instanzen | Port |
|---|---|
| ls1, ls2, ls3, ls4 [S] | Lichtsensor am entsprechenden SensorPort |

| Methode | Aktion |
|---|---|
| getValue() [S] | liefert die Lichtintensität (0..1023) |

**TouchSensor**

| Instanzen | Port |
|---|---|
| ts1, ts2, ts3, ts4 [S] | Touchsensor am entsprechenden SensorPort |

| Methode | Aktion |
|---|---|
| isPressed() [S] | liefert True, falls der Touchsensor gedrückt ist |

**Distanzsensor (EV3 Ultrasonicsensor):**

| Instanzen | Port |
|---|---|
| us1, us2, us3, us4 [S] | Sensor am entsprechenden SensorPort |

| Methoden | Aktion |
|---|---|
| getDistance() [S] | liefert Distanz in cm (255, falls keine reflektierendes Objekt gefunden wird, im Simulationsmodus: -1) |
| setProximityCircleColor(color) [S] | *Nur Simulationsmodus:* setzt die Farbe des Suchkreises |
| setMeshTriangleColor(color) [S] | *Nur Simulationsmodus:* setzt die Füllfarbe der Maschen |
| setBeamAreaColor(color) [S] | *Nur Simulationsmodus:* setzt die Farbe der Strahlbereichsgrenzen |
| eraseBeamArea() [S] | *Nur Simulationsmodus:* löscht die Strahlbereichsgrenzen |

Bemerkung:
Diese Methoden (mit SensorPort.S1) gibt es auch als globale Funktionen (Kompatibilität zu mbrobot)

**Temperatursensor (NXT Temperatursensor):**

| Instanzen | Port |
|---|---|
| tmp1, tmp2, tmp3, tmp4 | Sensor am entsprechenden SensorPort |

| Methode | Aktion |
|---|---|
| getTemperature() | liefert die Temperatur in Grad Celsius |

Bemerkung:
Diese Methode (mit SensorPort.S1) gibt es auch als globale Funktion (Kompatibilität zu mbrobot)

**Temperatur- Feuchtigkeitssensor (Sensirion SHT31 über I2C):**

| Instanzen | Port |
|---|---|
| sht1, sht2, sht3, sht4 | Sensor am entsprechenden SensorPort |

| Methode | Aktion |
|---|---|
| getValues() | liefert ein Float-Tupel mit Temperatur (Grad C) und Luftfeuchtigkeit (%). |

**Umweltsensor (Temperatur, Luftfeuchtigkeit, Luftdruck) (Bosch BME280 über I2C):**

| Instanzen | Port |
|---|---|
| bme1, bme2, bme3, bme4 | Sensor am entsprechenden SensorPort |

| Methode | Aktion |
|---|---|
| getValues() | liefert ein Float-Tupel mit Temperatur (Grad C) , Luftfeuchtigkeit (%) und Luftdruck (hPa) |

**3-Achsen Beschleunigungssensor (ADXL345 über I2C):**

| Instanzen | Port |
|---|---|
| adxl1, adxl2, adxl3, adxl4 | Sensor am entsprechenden SensorPort |

| Methode | Aktion |
|---|---|
| getValues() | liefert ein Float-Tupel mit den Beschleunigungen in x-, y- und z-Richtung (m/s^2 im Bereich -2g .. 2g) |

**Infrarotsensor (EV3 Infrarotsensor mit Fernsteuermodul) am SensorPort S1:**

| Instanzen | Port |
|---|---|
| irs | Infrarotsensor an SensorPort S1 |

| Methode | Aktion |
|---|---|
| getCommand() | gibt die aktuelle Kommando-ID für die gedrückten Tasten zurück:<br><br>0: nichts gedrückt<br>1: oben-links<br>2: unten-links<br>3: oben-rechts<br>4: unten-rechts<br>5: oben-links+oben-rechts<br>6: oben-links+unten-rechts<br>7: unten-links+oben-rechts<br>8: unten-links+unten-rechts<br>9: Zentrum<br>10: unten-links+oben-links<br>11: oben-rechts+unten-rechts<br><br>Der rote Schiebeschalter muss ganz oben stehen (für SensorPort S1) |

**Internet (EV3 über WLAN-Adapter mit Accesspoint verbunden):**

| Funktion | Aktion |
|---|---|
| httpGet(url) | führt einen HTTP GET Request durch und liefert den Response zurück. url in der Form "http://<server>?key=value&key=value&..." oder für SSL "https://..." |
| 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&..." oder für SSL "https://..." |
| httpDelete(url) | führt einen HTTP DELETE Request mit der gegebenen Ressource aus |
| startHTTPServer(handler) | startet einen HTTP Server (Webserver auf Port 80), der auf HTTP GET Requests hört. Bei einem GET Request wird der benutzerdefinierte Callback handler(clientIP, filename, params) aufgerufen:<br><br>clientIP: gepunktete IP-Adresse des Clients |

| | |
|---|---|
| | filename: Dateiname der URL mit vorgestelltem "/". Fehlt der Dateiname: "/"<br>params: Dictionary mit den GET Request Parametern {key: value}.<br><br>Beispiel http://192.168.0.101/on?a=ok&b=3<br><br>clientIP = "192.168.0.101"<br>filename = "/on"<br>params = {"a" : "ok", "b" : "3"}<br><br>Die Rückgabe besteht aus kommagetrennten Werten, die in die %s-Formatangaben der gespeicherten HTML-Datei eingebaut werden, bevor sie dem Browser zurückgesendet wird. (Die Anzahl und Reihenfolge muss übereinstimmen.) Fehlt der Rückgabewert, so wird die HTML-Datei unverändert zurückgesendet<br><br>(Der Aufruf ist nicht blockierend, da der Server in einem eigenen Thread läuft.) |
| saveHTML(text) | speichert den Text als HTML-Datei, die dem Browser bei einem GET Request zurück gesendet wird. Sie kann %-Formatangaben enthalten, die mit den Rückgabewerten des Callbacks handler(clientIP, filename, params) ersetzt werden |

**Wetterdaten**

| Funktion | Aktion |
|---|---|
| getWeather(city, key) | macht einen Wetterdaten-Request auf **http://openweathermap.org** für die gewählte Stadt. Dabei wird der gegebene Authorisierungsschlüssel verwendet. Diesen kann man man kostenfrei auf diesem Host beziehen. (Wird der Parameter weggelassen, so wird ein Standardschlüssel verwendet, der maximal 60 Anfragen / min für alle Nutzer erlaubt.)<br><br>Rückgabewert: Dictionary mit den Feldern:<br><br>"status" "OK" oder Fehlerstring, der den Fehler beschreibt, z.B. "City not found" (string)<br>"temp" Temperatur in °C (float)<br>"pressure" Luftdruck in hPa (mbar) (float)<br>"humidity" Luftfeuchtigkeit in % (int)<br>"temp_min" Tagesminimum der Temperatur in °C (float)<br>"temp_max" Tagesmaximum der Temperatur in °C (float)<br>"description" Wetterlage in WPorten (deutsch) (string)<br>"sunrise" Sonnenaufgang in Universal Time (UTC) (string)<br>"sunset" Sonnenuntergang in Universal Time (UTC) (string)<br>"datetime" Datum - Uhrzeit (UTC) der Wettererfassung (string) |

**Lokales Datum/Zeit** für gegebene Stadt

| Funktion | Aktion |
|---|---|
| getHTTPDateTime(city, timezone, key) | macht einen Zeit/Datum-Request auf **https://api.timezonedb.com** für die gewählte Stadt.. Dabei wird der gegebene Authorisierungsschlüssel verwendet. Diesen kann man man kostenfrei |

| | |
|---|---|
| | auf diesem Host beziehen. (Wird der Parameter weggelassen, so wird ein Standardschlüssel verwendet, der maximal 1 Anfrage/sec für alle Nutzer erlaubt.) timezone ist entweder der Country Code oder der Landesname (englisch).<br><br>Rückgabewert: String im Format: ww yyyy-nn-dd hh:mm:ss<br>ww: Wochentag, yyyy: Jahr, nn: Monat, dd: Tag, hh: Stunden, mm: Minuten, ss: Sekunden<br>Beispiel: Mo 2018-09-10 20:27:33<br><br>Falls ein Fehler auftritt, wird None zurückgegeben |
| getHTTPTime(city, timezone, key) | dasselbe, aber es wird nur der Zeitteil im Format hh:mm:ss zurückgegeben |
| getCountryCode(country) | gibt den Country Code (2 Buchstaben) zum gegebenen Land (Englisch geschrieben) zurück |
| getCountry(countryCode) | gibt das Land zum gegebenen Country Code zurück |
| countryCodes | Dictionary mit code:country |

# 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