Erste Schritte mit LEGO® MINDSTORMS® Education EV3 MicroPython

Version 1.0.0

INHALTSVERZEICHNIS

In dieser Anleitung wird erklärt, wie man MicroPython-Programme für LEGO® MINDSTORMS® Education EV3 Roboter schreibt. Dazu sind zwei Schritte nötig:

- Installation: Zuerst müssen der Computer und der EV3 Stein vorbereitet werden, indem alle erforderlichen Tools installiert werden. Zudem wird erklärt, wie man den EV3 Stein ein- bzw. ausschaltet und wie man durch das Menü auf dem Bildschirm navigiert.
- Erstellen und Ausführen von Programmen: Als Nächstes wird gezeigt, wie man ein Programm erstellt und es auf den EV3 Stein herunterlädt. Außerdem erfährst du, wie man das Programm über den Computer oder den EV3 Stein ausführt.

Nachdem du das erste Demo-Programm ausgeführt hast, kannst du die Beispielprogramme ausprobieren und eigene Programme erstellen.

KAPITEL
EINS

INSTALLATION

Auf dieser Seite ist beschrieben, welche Vorbereitungen getroffen werden müssen und was installiert werden muss, um mit dem Programmieren beginnen zu können.

1.1 Was wird benötigt?

Um loszulegen, wird Folgendes benötigt:

- Ein Computer mit Windows 10 oder Mac OS
- Internetverbindung und Administratorrechte

  Dies ist nur für die Installation erforderlich. Danach sind keine speziellen Zugriffsrechte zum Schreiben und Ausführen von Programmen erforderlich.
- Eine microSD-Karte

  Es wird eine Karte mit mindestens 4 GB Speicherplatz benötigt. Maximal darf die Karte 32 GB Speicherplatz aufweisen. Diese Art von microSD-Karte wird auch als microSDHC bezeichnet. Wir empfehlen Karten mit dem A1-Siegel (Application Performance Class A1).
- Ein microSD-Kartensteckplatz oder -Kartenleser im Computer

  Wenn dein Computer keinen microSD-Kartensteckplatz aufweist, kannst du einen externen microSD-Kartenleser verwenden, der über einen USB-Stecker angeschlossen wird.
- Ein Mini-USB-Kabel (im Lieferumfang des EV3 Sets enthalten)

Abbildung 1.1 zeigt, wie diese Komponenten miteinander verbunden werden.

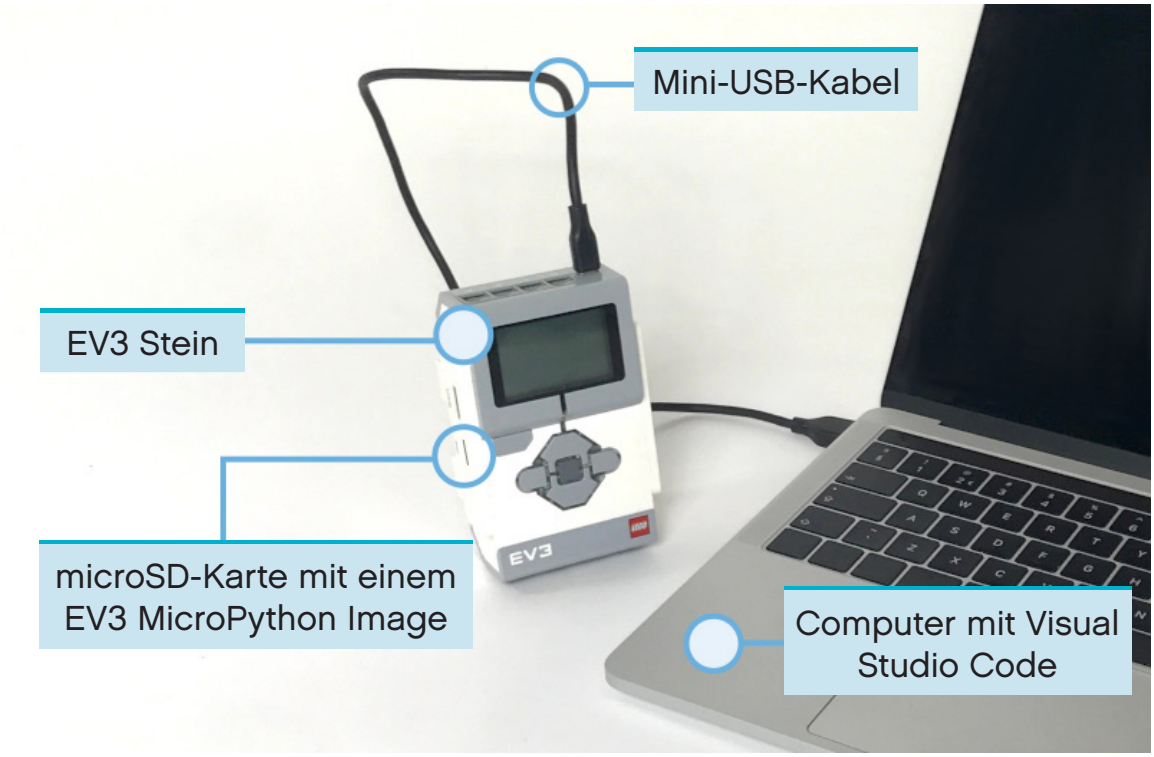


Abbildung 1.1: Überblick über die Komponenten

1.2 Vorbereiten des Computers

Die MicroPython-Programme werden mit Visual Studio Code geschrieben. Befolge diese Schritte, um die Anwendung herunterzuladen, zu installieren und einzurichten:

1. Visual Studio Code herunterladen.
2. Den Anweisungen auf dem Bildschirm folgen, um die Anwendung zu installieren.
3. Visual Studio Code starten.
4. „Extensions“ (Erweiterungen) öffnen.
5. Die Erweiterung „EV3 MicroPython“ wie in Abbildung 1.2 dargestellt installieren.

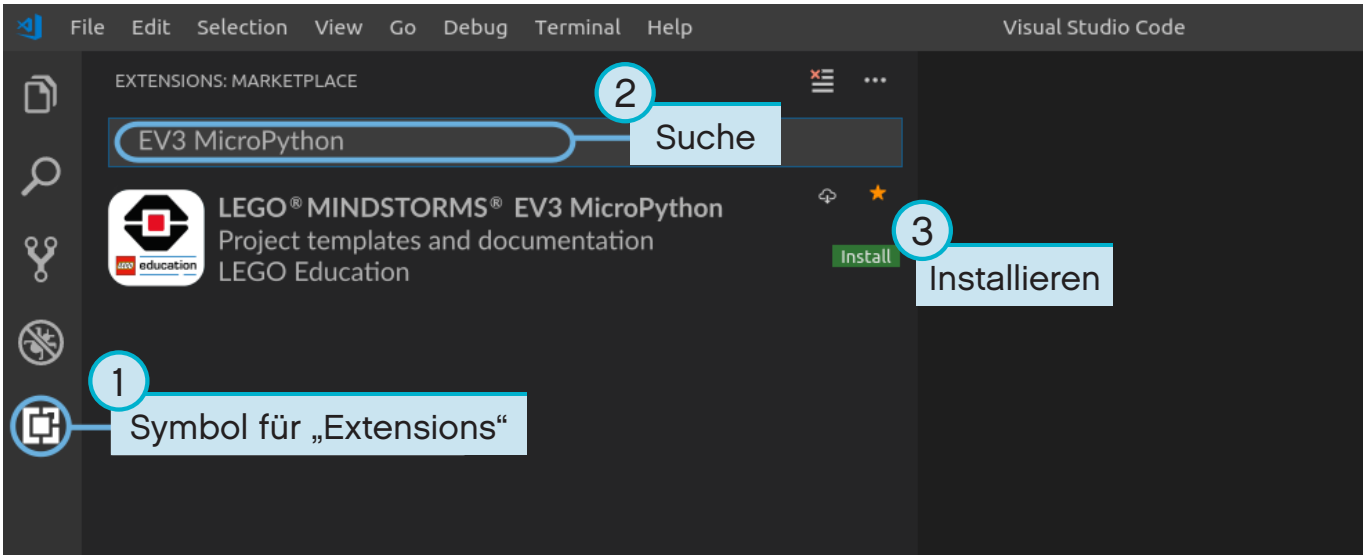


Abbildung 1.2: Installieren der Erweiterung über den Marketplace in Visual Studio Code

1.3 Vorbereiten der microSD-Karte

Um MicroPython-Programme auf dem EV3 Stein ausführen zu können, müssen zunächst alle erforderlichen Tools auf der microSD-Karte installiert werden. Im Folgenden findest du die Anleitung dazu.

Wenn auf der microSD-Karte Dateien gespeichert sind, sollten diese Dateien zunächst gesichert werden. Wie du MicroPython-Programme sicherst, wird im Abschnitt 2.6 *Verwalten von Dateien auf dem EV3 Stein* beschrieben.

Dieser Vorgang löscht alles, was auf der microSD-Karte gespeichert ist – auch bereits vorhandene MicroPython-Programme.

Installieren der MicroPython-Tools auf deiner microSD-Karte:

1. Das EV3 MicroPython Image für die microSD-Karte herunterladen und an einem geeigneten Ort speichern. Die Datei ist etwa 360 MB groß. Die Datei muss nicht extrahiert/entpackt werden.
2. Ein Flashing-Tool für microSD-Karten, wie z. B. das kostenfreie Tool Etcher, herunterladen und installieren.
3. Die microSD-Karte in den Computer oder Kartenleser einstecken.
4. Das Flashing-Tool starten und den Anweisungen auf dem Bildschirm folgen, um die gerade heruntergeladene Datei zu installieren. Wenn du Etcher verwendest, findest du im Folgenden eine Installationsanleitung (siehe auch Abbildung 1.3).
    a. Auf der microSD-Karte die Datei mit dem EV3 MicroPython Image auswählen, welches du gerade heruntergeladen haben.
    b. Deine microSD-Karte auswählen. Achte darauf, dass das Gerät und die Größe mit deiner microSD-Karte übereinstimmen.
    c. Den Flashing-Prozess starten. Dies kann einige Minuten dauern. Die Karte nicht entfernen, bis der Flashing-Prozess abgeschlossen ist.

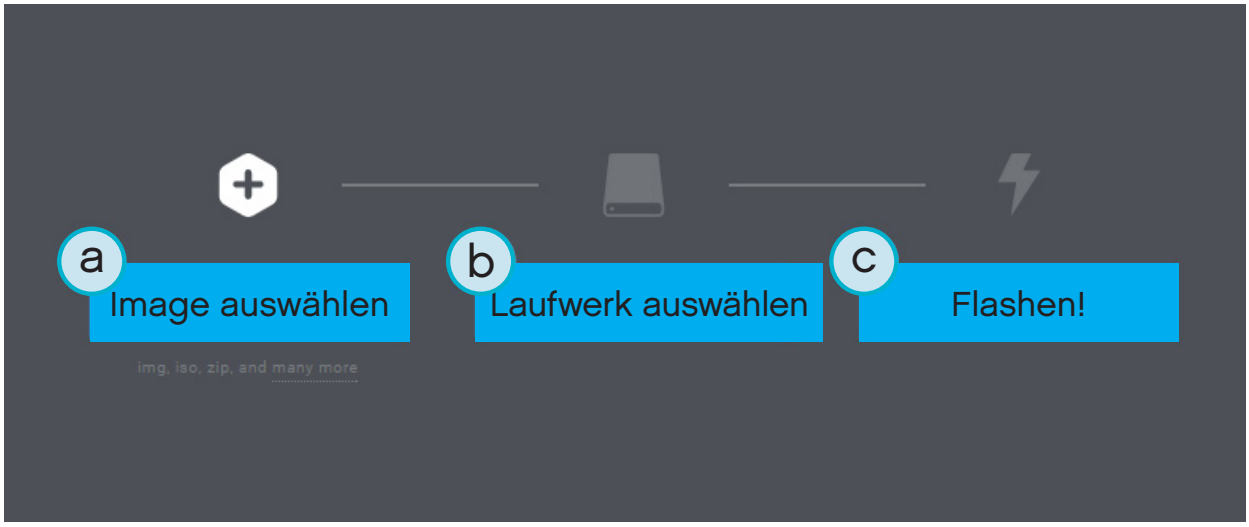


Abbildung 1.3: Flashen eines EV3 MicroPython Image auf einer microSD-Karte mithilfe von Etcher

1.4 Aktualisieren der microSD-Karte

Um die microSD-Karte zu aktualisieren, muss zunächst eine neue Image-Datei über den Link oben heruntergeladen werden. Anschließend wird erneut der oben beschriebene Flashing-Prozess durchgeführt. Vorher müssen alle MicroPython-Programme gesichert werden, die du behalten möchtest.

Es ist vor der Aktualisierung nicht notwendig, die Dateien auf der microSD-Karte zu löschen. Die Dateien werden automatisch während des Flashing-Prozesses gelöscht.

1.5 Verwenden des EV3 Steins

Vergewissere dich, dass der EV3 Stein ausgeschaltet ist. Setze dann die vorbereitete microSD-Karte in den microSD-Kartensteckplatz am EV3 Stein ein, wie in Abbildung 1.4 dargestellt.

Abbildung 1.4: Einsetzen der vorbereiteten microSD-Karte in den EV3 Stein

1.5.1 Ein- und Ausschalten des EV3 Steins

EV3 Stein einschalten, indem man auf die dunkelgraue mittlere Taste drückt.

Das Hochfahren kann einige Minuten dauern. Beim Hochfahren blinkt die Stein-Statusleuchte orange. Gleichzeitig läuft viel Text über den EV3 Bildschirm. Der EV3 Stein ist einsatzbereit, sobald die Stein-Statusleuchte grün leuchtet.

Um den EV3 Stein auszuschalten, Ausschaltmenü über die Zurück-Taste öffnen und dann mit der mittleren Taste „Power Off“ auswählen, wie in Abbildung 1.5 dargestellt.

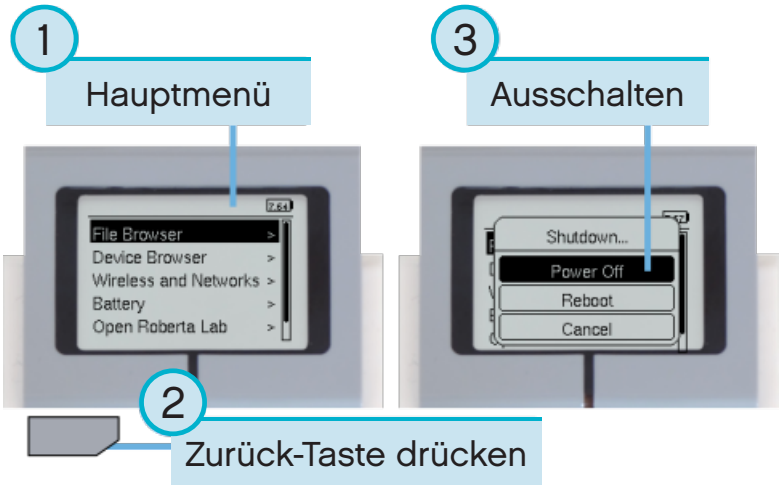


Abbildung 1.5: Ausschalten des EV3 Steins

1.5.2 Anzeigen der Motor- und Sensorwerte

Wenn gerade kein Programm ausgeführt wird, kannst du dir die Motor- und Sensorwerte über den Geräte-Browser anzeigen lassen, wie in Abbildung 1.6 dargestellt.

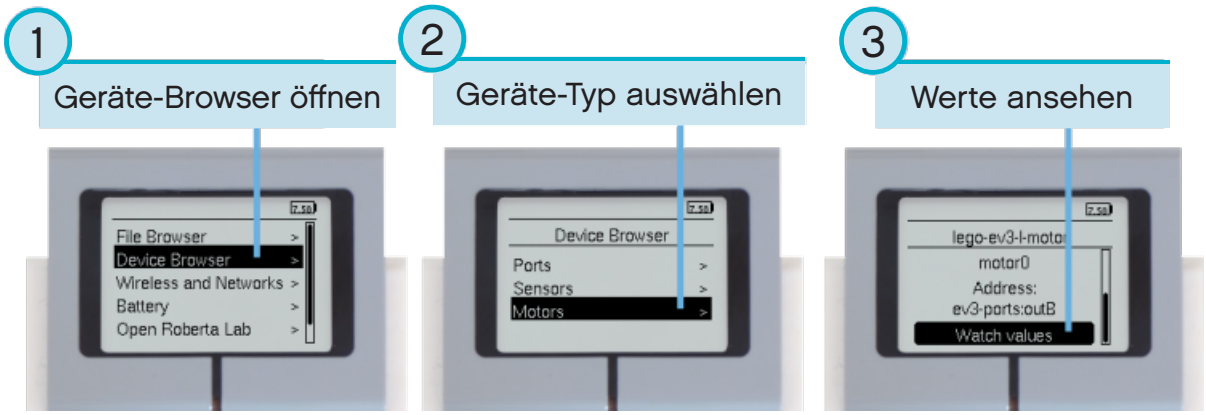


Abbildung 1.6: Anzeigen der Motor- und Sensorwerte

1.5.3 Zurückkehren zur ursprünglichen EV3 Software

Es ist jederzeit möglich, von MicroPython zur EV3 Software und den darin erstellten Programmen zurückzukehren. Gehe dazu wie folgt vor:

1. Den EV3 Stein wie oben beschrieben ausschalten.
2. Warten, bis der Bildschirm und die Stein-Statusleuchte ausgeschaltet sind.
3. MicroSD-Karte entfernen.
4. EV3 Stein einschalten.

KAPITEL

ZWEI

ERSTELLEN UND AUSFÜHREN VON PROGRAMMEN

Der Computer und der EV3 Stein sind nun eingerichtet. Jetzt kannst du beginnen, Programme zu schreiben.

Um das Erstellen und Verwalten von Programmen zu erleichtern, sollte man wissen, wie MicroPython-Projekte und -Programme geordnet werden.

Die Programme werden in Projektordnern abgelegt, wie in Abbildung 2.1 dargestellt. Ein Projektordner ist ein Verzeichnis auf einem Computer, das das Hauptprogramm (**main.py**) und weitere optionale Skripte oder Dateien enthält. Dieser Projektordner und sein gesamter Inhalt werden auf den EV3 Stein kopiert, von dem aus das Hauptprogramm ausgeführt wird.

Auf dieser Seite wird erklärt, wie ein solches Projekt angelegt und auf den EV3 Stein übertragen wird.

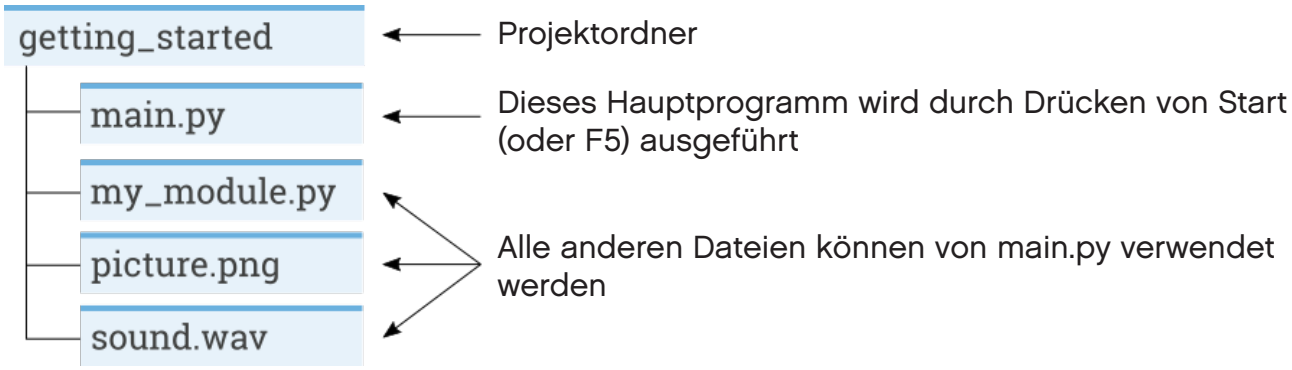


Abbildung 2.1: Ein Projekt enthält ein Programm namens **main.py** sowie optionale Ressourcen wie Klänge und MicroPython-Module.

2.1 Anlegen eines neuen Projekts

Um ein neues Projekt anzulegen, klicke auf das Symbol „EV3 MicroPython“ und anschließend auf *Create a new project*, wie in Abbildung 2.2 dargestellt. Daraufhin öffnet sich ein Textfeld. Gib einen Projektnamen in das Feld ein und klicke auf *Enter*. Wenn du dazu aufgefordert wirst, wähle einen Speicherort für dieses Programm aus und bestätige deine Auswahl durch Klicken auf *choose folder*.

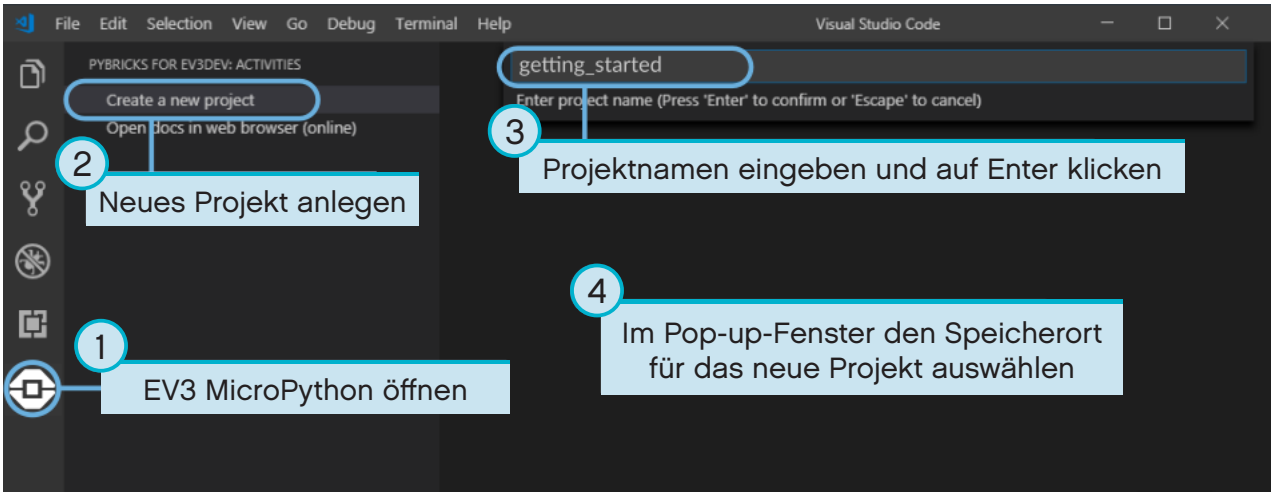


Abbildung 2.2: Anlegen eines neuen Projekts. In diesem Beispiel heißt das Projekt *getting_started*. Man kann jedoch einen beliebigen Namen festlegen.

Wenn du ein neues Projekt anlegst, enthält dieses Projekt automatisch eine Datei namens *main.py*. Um den Inhalt dieser Datei anzusehen und zu ändern, öffne diese über den Browser, wie in Abbildung 2.3 dargestellt. Hier schreibst du deine Programme.

Wenn du noch nicht mit MicroPython programmiert hast, empfehlen wir, das vorhandene Programm nicht zu löschen, sondern nur zu bearbeiten und zu ergänzen.

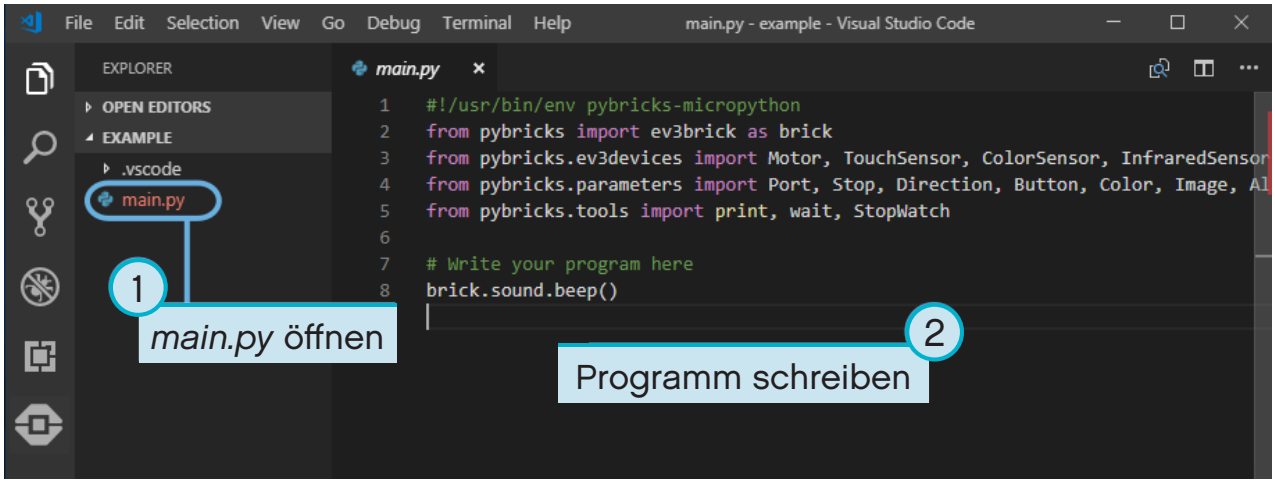


Abbildung 2.3: Öffnen des Standardprogramms main.py

2.2 Öffnen eines vorhandenen Projekts

Um ein bereits angelegtes Projekt zu öffnen, klicke auf *File* und anschließend auf *Open Folder*, wie in Abbildung 2.4 dargestellt. Suche dann den bereits angelegten Projektordner heraus und klicke auf *OK*. Zudem kannst du über den Menüpunkt *Open Recent* die kürzlich verwendeten Projekte öffnen.

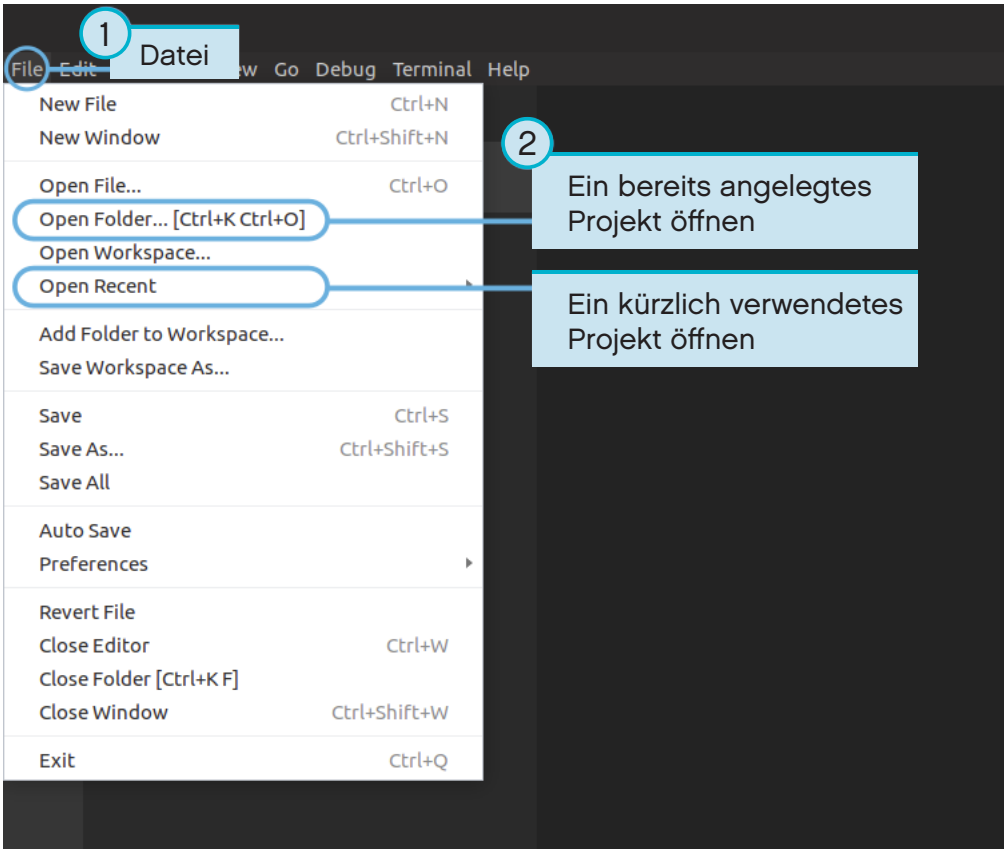


Abbildung 2.4: Öffnen eines bereits angelegten Projekts

2.3 Verbinden des EV3 Steins mit Visual Studio Code

Um das Programm auf den EV3 Stein zu übertragen, muss zuerst der EV3 Stein über das Mini-USB-Kabel am Computer angeschlossen und die Verbindung mit Visual Studio Code eingerichtet werden. Gehe dazu wie folgt vor:

- Den EV3 Stein einschalten.
- Den EV3 Stein über das Mini-USB-Kabel an den Computer anschließen.
- Die USB-Verbindung wie in Abbildung 2.5 dargestellt einrichten.

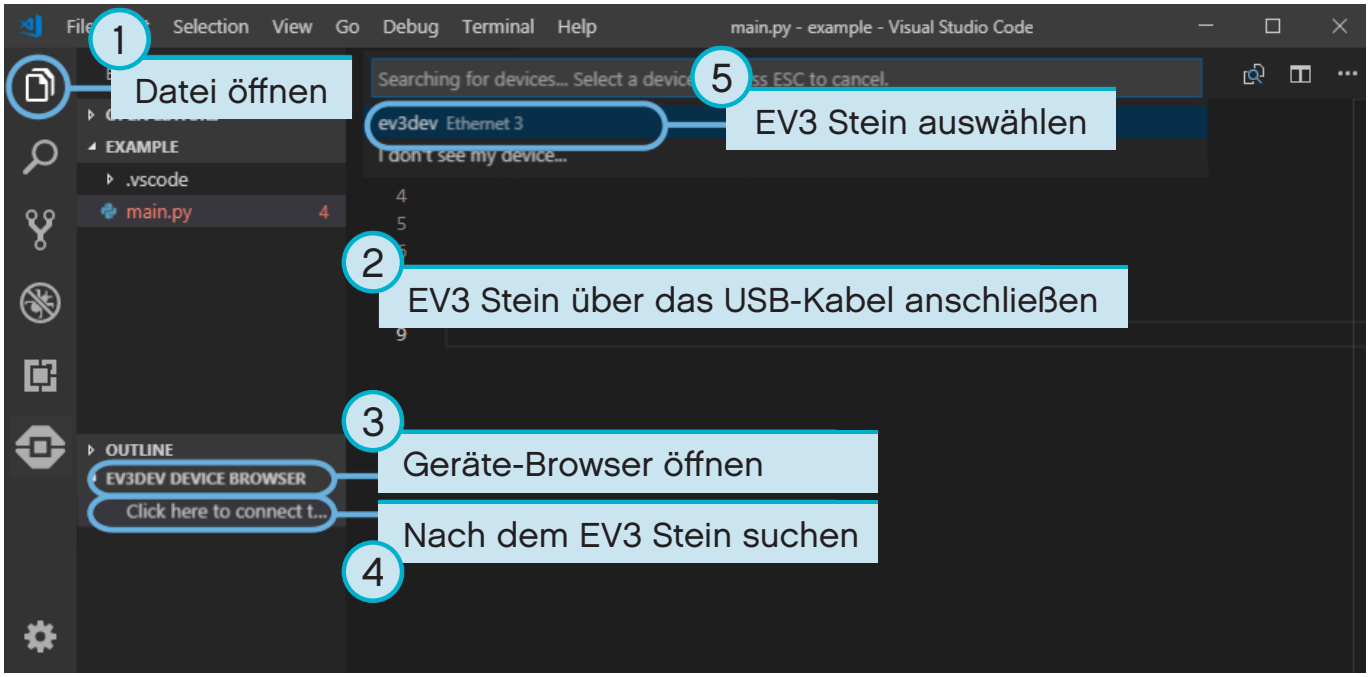


Abbildung 2.5: Einrichten der USB-Verbindung zwischen Computer und EV3 Stein

2.4 Herunterladen und Ausführen eines Programms

Drücke die F5-Taste, um ein Programm auszuführen. Alternativ das „Debug"-Fenster öffnen und auf den grünen Play-Pfeil klicken, um ein Programm manuell auszuführen. Siehe Abbildung 2.6.

Beim Starten öffnet sich eine Werkzeugleiste, über die das Programm bei Bedarf angehalten werden kann. Alternativ kann das Programm jederzeit gestoppt werden, indem man auf die Zurück-Taste des EV3 Steins drückt.

Wenn das Programm irgendeine Ausgabe mit dem Befehl *print* erzeugt, wird diese im Ausgabefenster angezeigt.

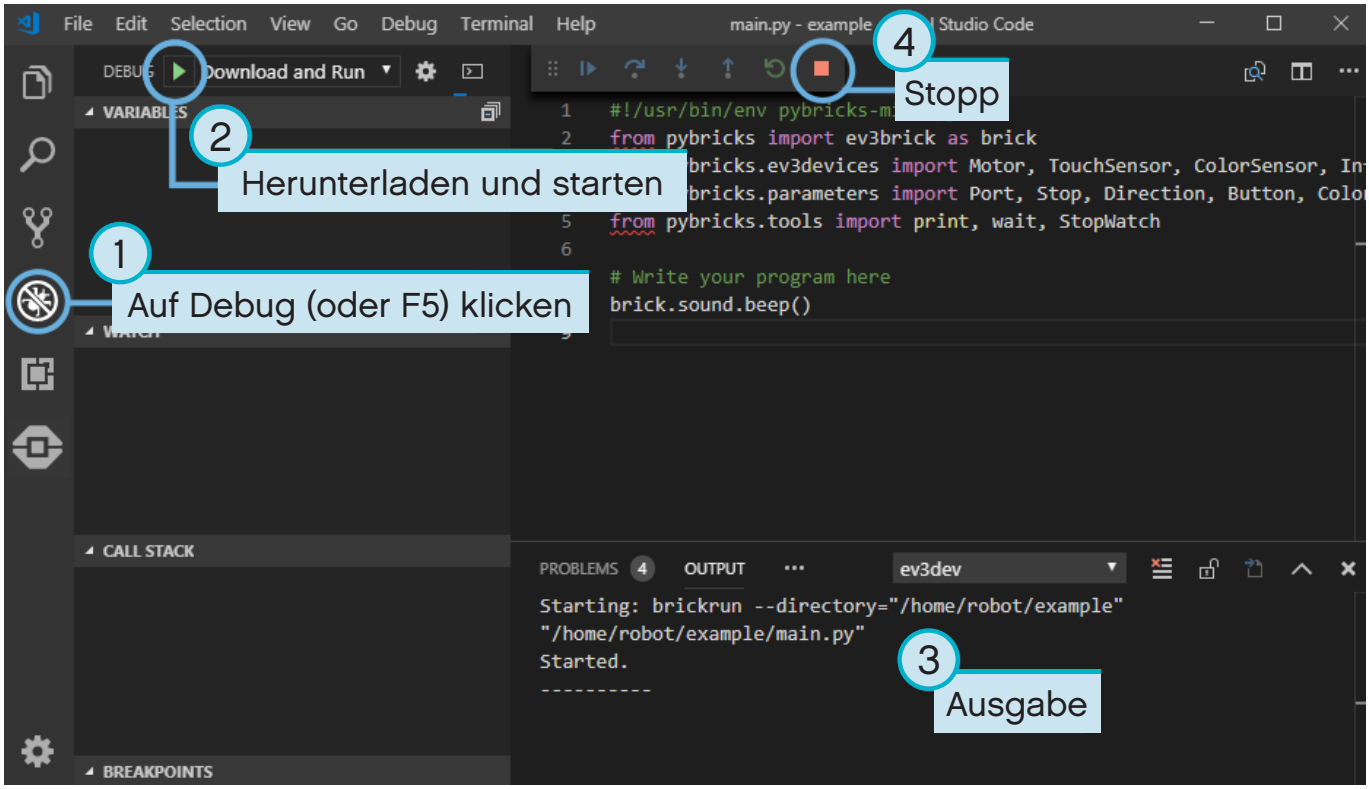


Abbildung 2.6: Ausführen eines Programms

2.5 Ergänzen des Beispielprogramms

Du hast jetzt bereits ein einfaches Beispielprogramm ausgeführt. Nun kannst du dieses Programm ergänzen, um damit den Motor zu steuern. Zunächst muss ein großer Motor an Anschluss B des EV3 Steins angeschlossen werden, wie in Abbildung 2.7 dargestellt.

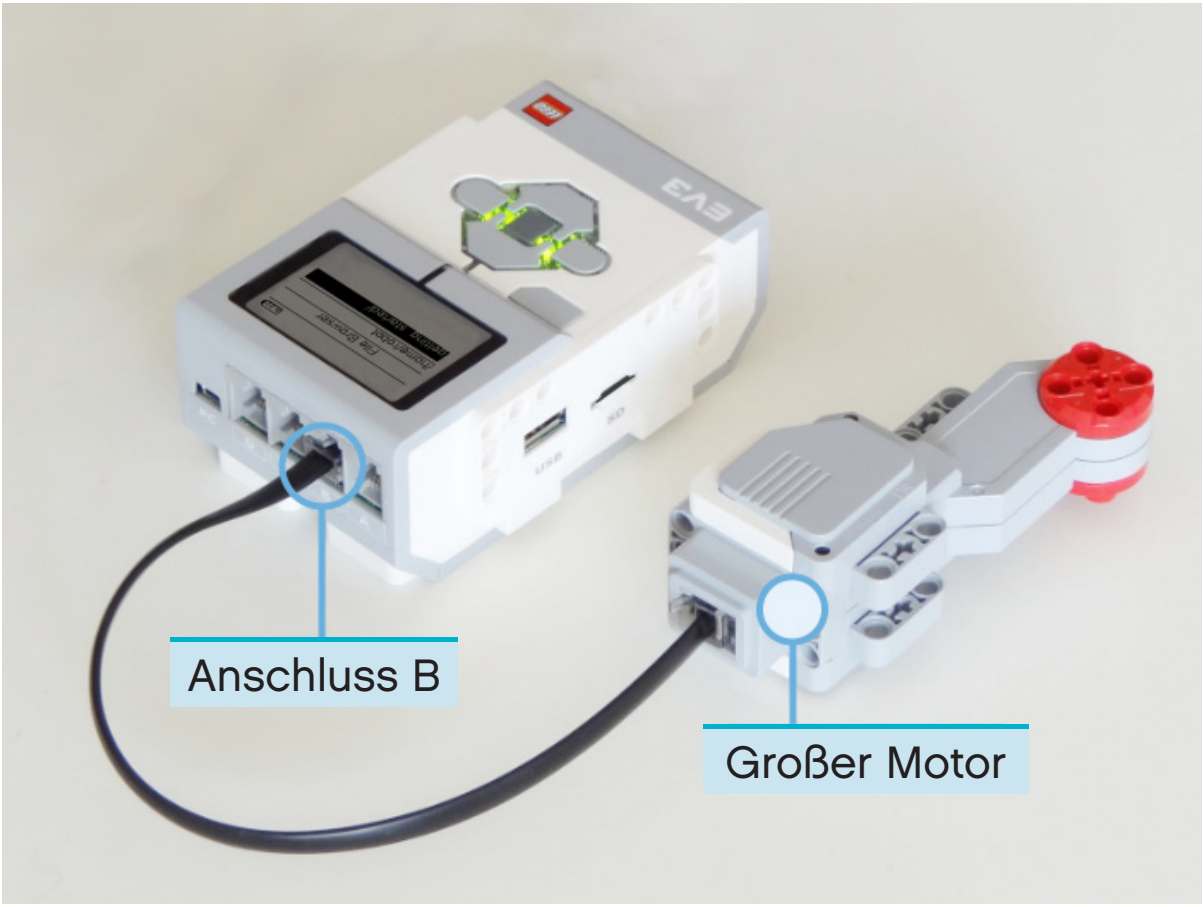


Abbildung 2.7: EV3 Stein mit einem an Anschluss B angeschlossenen großen Motor

Bearbeite nun das Programm *main.py* so, dass es am Ende wie folgt aussieht:

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port

# Play a sound.
brick.sound.beep()

# Initialize a motor at port B.
test_motor = Motor(Port.B)

# Run the motor up to 500 degrees per second. To a target angle of 90 degrees.
test_motor.run_target(500, 90)

# Play another beep sound.
# This time with a higher pitch(1000 Hz) and longer duration(500 ms).
brick.sound.beep(1000, 500)
```

Mit diesem Programm geschieht Folgendes: Piepton, Motor dreht sich, Piepton in einer höheren Tonlage. Führe das Programm aus, um zu überprüfen, ob es wie geplant funktioniert.

2.6 Verwalten von Dateien auf dem EV3 Stein

Nachdem du ein Projekt auf den EV3 Stein heruntergeladen hast, kannst du darin gespeicherte Programme ausführen, löschen oder sichern. Verwende dazu den Geräte-Browser, wie in Abbildung 2.8 dargestellt.

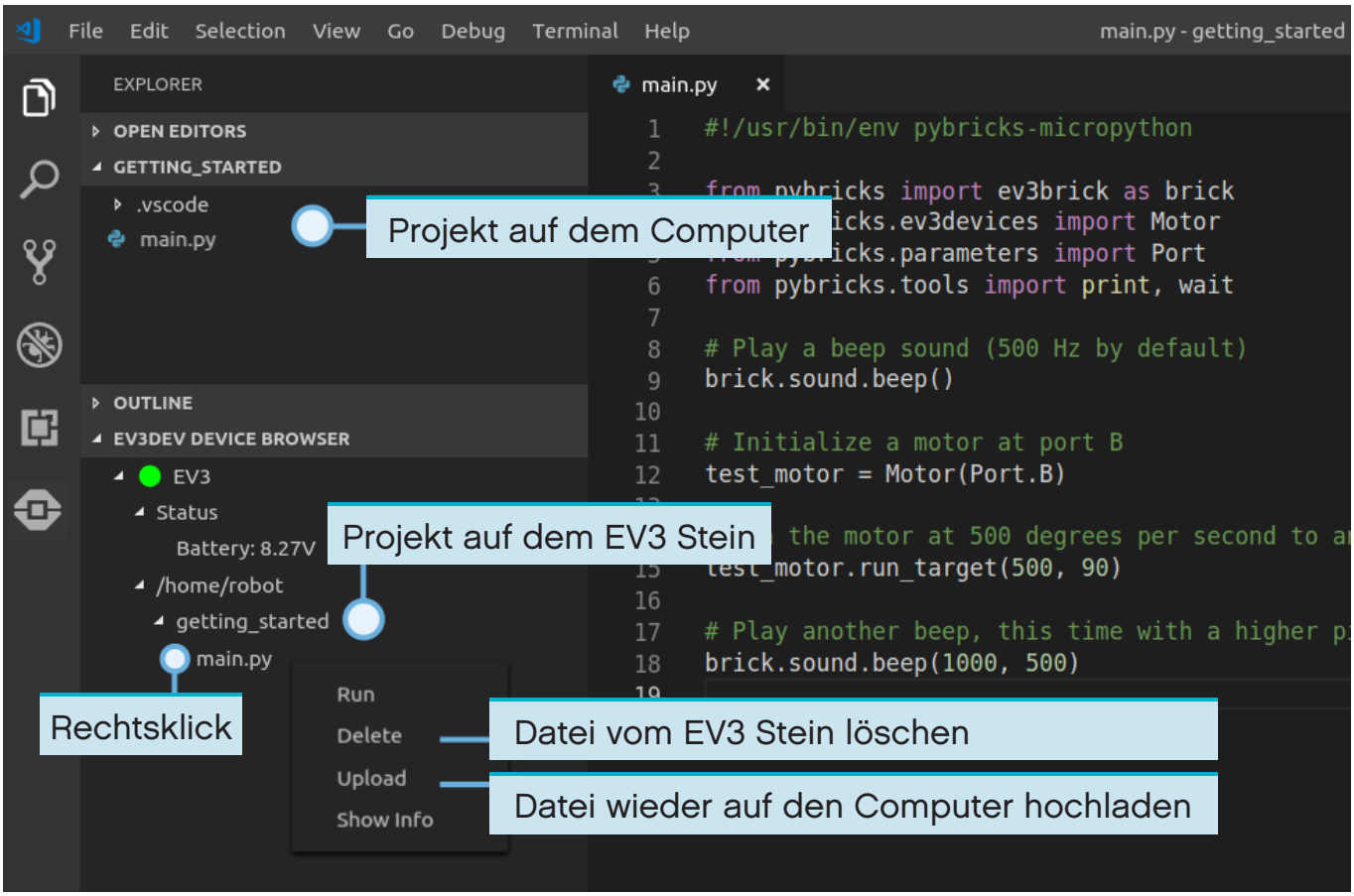


Abbildung 2.8: Verwalten von Dateien auf dem EV3 Stein über den EV3 Geräte-Browser

2.7 Ausführen eines Programms ohne Computer

Bereits heruntergeladene Programme können auch direkt über den EV3 Stein ausgeführt werden.

Suche dazu mit dem *file browser* (Datei-Browser) auf dem EV3 Stein das gewünschte Programm heraus. Drücke anschließend auf die mittlere Taste, um das Programm auszuführen. Siehe Abbildung 2.9.

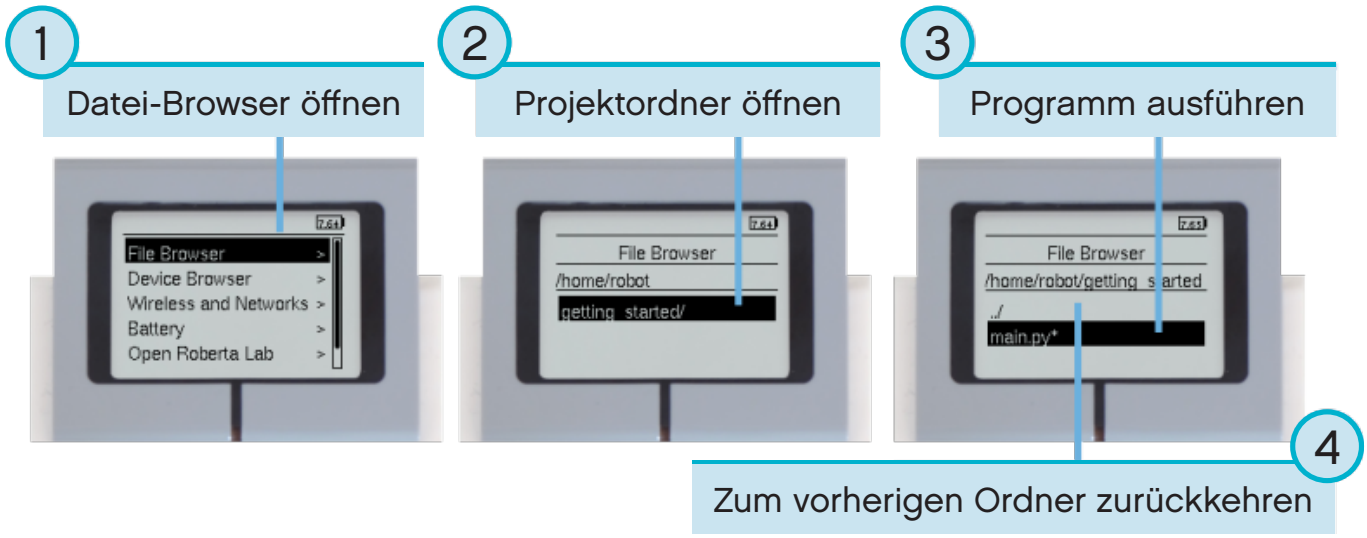


Abbildung 2.9: Ausführen eines Programms über die Tasten auf dem EV3 Stein

KAPITEL

DREI

EV3 STEIN – DER PROGRAMMIERBARE EV3 STEIN

LEGO® MINDSTORMS® Education EV3 Stein

3.1 Tasten

buttons ()

Hiermit wird überprüft, welche Tasten auf dem EV3 Stein derzeit gedrückt werden.

Returns Liste der gedrückten Tasten.

Return type Liste der *Buttons*

Beispiele:

```
# Do something if the left button is pressed
if Button.LEFT in brick.buttons():
    print("The left button is pressed.")
```

```
# Wait until any of the buttons are pressed
while not any(brick.buttons()):
    wait(10)

# Wait until all buttons are released
while any(brick.buttons()):
    wait(10)
```

3.2 Licht

light(*color*)

Hiermit wird die Farbe der EV3 Stein-Leuchte eingestellt.

Parameters **color**(Color) – Farbe des Lichts: Wähle `Color.BLACK` oder `None` aus, um das Licht auszuschalten.

Beispiel:

```
# Make the light red
brick.light(Color.RED)

# Turn the light off
brick.light(None)
```

3.3 Klang

classmethod sound.**beep**(*frequency=500, duration=100, volume=30*)

Hiermit wird ein Piepton abgespielt.

Parameters

- **frequency** (*frequency: Hz*) – Frequenz des Pieptons (*Standardeinstellung*: 500)
- **duration** (*time: ms*) – Dauer des Pieptons (*Standardeinstellung*: 100)
- **volume** (*percentage: %*) – Lautstärke des Pieptons (*Standardeinstellung*: 30)

Beispiel:

```
# A simple beep
brick.sound.beep()

# A high pitch(1500 Hz) for one second(1000 ms) at 50% volume
brick.sound.beep(1500, 1000, 50)
```

classmethod sound.**beeps**(*number*)

Hiermit wird eine bestimmte Anzahl an Tönen abgespielt. Zwischen den Tönen erfolgt eine kurze Pause.

Parameters number(*int*) – Anzahl an Pieptönen

Beispiel:

```
# Make 5 simple beeps
brick.sound.beeps(5)
```

classmethod sound.**file**(*file_name, volume=100*)

Hiermit wird eine Klang-Datei abgespielt.

Parameters

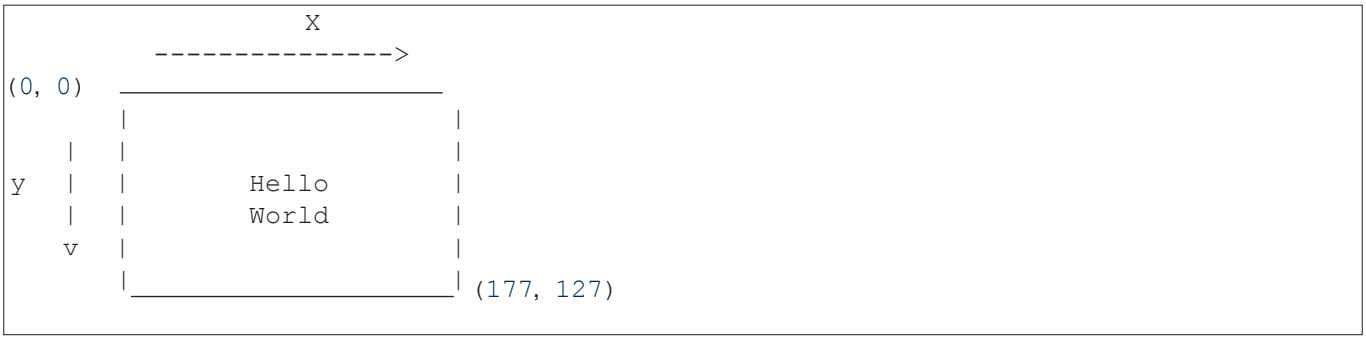
- **file_name** (*str*) – Dateipfad zur Klangdatei, einschließlich Erweiterung
- **volume** (*percentage: %*) – Lautstärke des Klangs (*Standardeinstellung*: 100)

Beispiel:

```
# Play one of the built-in sounds
brick.sound.file(SoundFile.HELLO)

# Play a sound file from your project folder
brick.sound.file('mysound.wav')
```

3.4 Display



classmethod display.**clear**()
Hiermit wird die Anzeige zurückgesetzt.

classmethod display.**text**(*text, coordinate=None*)
Hiermit wird ein Text angezeigt.

Parameters

- **text**(*str*) – der anzuzeigende Text
- **coordinate**(*tuple*) –(x, y) Koordinaten-Tupel: Es handelt sich dabei um die obere linke Ecke des ersten Zeichens. Wenn keine Koordinate festgelegt ist, wird der Text auf der nächsten Zeile angezeigt.

Beispiel:

```
# Clear the display
brick.display.clear()

# Print ``Hello`` near the middle of the screen
brick.display.text("Hello",(60, 50))

# Print ``World`` directly underneath it
brick.display.text("World")
```

classmethod display.**image**(*file_name, alignment=Align.CENTER, coordinate=None, clear=True)*
Hiermit wird eine Bilddatei angezeigt.

Die Platzierung der Bilddatei kann mit `alignment` oder durch Festlegen einer `coordinate` bestimmt werden. Es kann aber nicht beides gleichzeitig verwendet werden.

Parameters

- **file_name**(*str*) – Dateipfad zur Bilddatei: Die Dateipfade können absolut sein oder relativ zum Projektordner.
- **alignment**(Align) – Stelle, an der das Bild platziert werden soll (*Standardeinstellung*: `Align.CENTER`)
- **coordinate**(*tuple*) –(x, y) Koordinaten-Tupel: Es handelt sich dabei um die obere linke Ecke des Bildes (*Standardeinstellung*: `None`).
- **clear**(*bool*) – legt fest, ob die Anzeige zurückgesetzt werden soll, bevor das Bild angezeigt wird (*Standardeinstellung*: `True`)

Beispiel:

```
# Show a built-in image of two eyes looking upward
brick.display.image(ImageFile.UP)

# Display a custom image from your project folder
brick.display.image('pybricks.png')

# Display a custom image at the top right of the screen, without clearing
# the screen first
brick.display.image('arrow.png', Align.TOP_RIGHT, clear=False)
```

3.5 Akku

classmethod battery.**voltage**()

Hiermit wird die Akkuspannung abgerufen.

Returns Akkuspannung.

Return type *voltage: mV*

Beispiele:

```
# Play a warning sound when the battery voltage
# is below 7 Volt(7000 mV = 7V)
if brick.battery.voltage() < 7000:
    brick.sound.beep()
```

classmethod battery.**current**()

Hiermit wird der Strom abgerufen, den der Akku aktuell liefert.

Returns Akkustrom.

Return type *current: mA*

KAPITEL

VIER

EV3 GERÄTE – EV3 MOTOREN UND SENSOREN

LEGO® MINDSTORMS® Education EV3 Motoren und Sensoren

4.1 Motoren

class Motor(*port, direction=Direction.CLOCKWISE, gears=None*)

LEGO® MINDSTORMS® Education EV3 mittlerer oder großer Motor

Element 99455/6148292 oder 95658/6148278, enthalten in:

- 31313: LEGO MINDSTORMS Education EV3 (2013)
- 45544: LEGO MINDSTORMS Education EV3 Set (2013)
- 45503 oder 45502: Separates Teil (2013)

Parameters

- **port** (`Port`) – Anschluss, an den der Motor angeschlossen ist
- **direction** (`Direction`) – positive Drehrichtung (*Standardeinstellung*: `Direction.CLOCKWISE`)
- **gears** (*`list`*) – Liste der Zahnräder, die mit dem Motor verbunden sind (*Standardeinstellung*: `None`).

  Beispiel: [`12`, `36`] steht für ein Zahnradgetriebe, das aus einem Zahnrad mit 12 Zähnen und einem Zahnrad mit 36 Zähnen besteht. Siehe das Beispiel zum Übersetzungsverhältnis unten.

  Verwende eine Liste, wenn mehrere Zahnradgetriebe verwendet werden sollen, wie zum Beispiel: `[[12, 36], [20, 16, 40]]`.

  Wenn du ein Zahnradgetriebe spezifizierst, werden alle Motorbefehle und Einstellungen automatisch an das jeweilige Übersetzungsverhältnis angepasst. Die Drehrichtung des Motors bleibt unabhängig von der Anzahl der ausgewählten Zahnräder unverändert.

  Bei `gears=[12, 36]` ist das Übersetzungsverhältnis gleich 3. Das bedeutet, dass die Leistung um den Faktor 3 mechanisch verlangsamt wird. Um diese Verlangsamung auszugleichen, läuft der Motor dreimal so schnell und dreimal so weit, wenn er einen Befehl erhält. Wenn du also `run_angle(200, 90)` auswählst, entspricht die Ausgabe für den Mechanismus ganz einfach einer Umdrehung von 90 Grad mit eine Geschwindigkeit von 200 Grad/Sekunde.

  Gleiches gilt für die Dokumentation unten: Wenn „Drehwinkel des Motors“ oder „Motorgeschwindigkeit“ angegeben ist, ist dies gleichbedeutend mit „Ausgabewinkel für den Mechanismus“ bzw. „Ausgabegeschwindigkeit für den Mechanismus“ usw., da das Übersetzungsverhältnis automatisch berücksichtigt wird.

  Die Einstellung `gears` ist nur für Motoren mit Drehsensor verfügbar.

Beispiel:

```
# Initialize a motor(by default this means clockwise, without any gears).
example_motor = Motor(Port.A)

# Initialize a motor where positive speed values should go counterclockwise
right_motor = Motor(Port.B, Direction.COUNTERCLOCKWISE)

# Initialize a motor with a gear train
robot_arm = Motor(Port.C, Direction.CLOCKWISE, [12, 36])
```

Funktionen für Motoren ohne Drehsensor

dc(*duty*)
Hiermit wird der Arbeitszyklus des Motors festgelegt.

Parameters **duty**(*percentage: %*) – Arbeitszyklus (-100 bis 100)

Beispiel:

```
# Set the motor duty cycle to 75%.
example_motor.duty(75)
```

Funktionen für Motoren mit Drehsensor

angle()
Hiermit wird der Drehwinkel des Motors abgerufen.

Returns Drehwinkel des Motors

Return type *angle: deg*

reset_angle(*angle*)
Hiermit wird der kumulierte Drehwinkel des Motors zurückgesetzt.

Parameters **angle**(*angle: deg*) – Wert, auf den der Winkel zurückgesetzt werden soll

speed()
Hiermit wird die Geschwindigkeit (Winkelgeschwindigkeit) des Motors abgerufen.

Returns Motorgeschwindigkeit

Return type *rotational speed: deg/s*

stop(*stop_type=Stop.COAST*)
Hiermit wird der Motor angehalten.

Parameters **stop_type**(`Stop`) – legt fest, ob der Motor auslaufen, abbremsen oder die Position beibehalten soll (*Standardeinstellung*: `Stop.COAST`)

run(*speed*)
Hiermit läuft der Motor mit einer konstanten Geschwindigkeit (Winkelgeschwindigkeit).

Der Motor beschleunigt bis zur festgelegten Geschwindigkeit und der Arbeitszyklus wird automatisch angepasst, um die Geschwindigkeit selbst bei einer gewissen Last konstant zu halten. Dieser Befehl wird im Hintergrund fortgesetzt, bis der Motor einen neuen Befehl empfängt oder das Programm anhält.

Parameters **speed**(*rotational speed: deg/s*) – Geschwindigkeit des Motors

run_time(*speed, time, stop_type=Stop.COAST, wait=True*)
Hiermit läuft der Motor eine bestimmte Zeit lang mit einer konstanten Geschwindigkeit (Winkelgeschwindigkeit).

Der Motor beschleunigt bis zur festgelegten Geschwindigkeit und der Arbeitszyklus wird automatisch angepasst, um die Geschwindigkeit selbst bei einer gewissen Last konstant zu halten. Der Motor bremst gerade rechtzeitig ab, um nach Ablauf der festgelegten Zeit anzuhalten.

Parameters

- **speed** (*rotational speed: deg/s*) – Geschwindigkeit des Motors
- **time** (*time: ms*) – Dauer des Fahrmanövers
- **stop_type** (`Stop`) – legt fest, ob der Motor nach dem Anhalten auslaufen, abbremsen oder die Position beibehalten soll (*Standardeinstellung*: *`Stop.COAST`*)
- **wait** (*`bool`*) – legt fest, ob das Fahrmanöver zuerst beendet sein muss, bevor der Rest des Programms ausgeführt wird (*Standardeinstellung*: `True`); Bedeutet, dass das Programm die unter `time` festgelegte Dauer wartet.

run_angle (*speed, rotation_angle, stop_type=Stop.COAST, wait=True*)
Hiermit läuft der Motor bis zu einem bestimmten Winkel mit einer konstanten Geschwindigkeit (Winkelgeschwindigkeit).

Der Motor beschleunigt bis zur festgelegten Geschwindigkeit und der Arbeitszyklus wird automatisch angepasst, um die Geschwindigkeit selbst bei einer gewissen Last konstant zu halten. Der Motor bremst gerade rechtzeitig ab, um zum Stillstand zu kommen, nachdem der festgelegte Winkel durchfahren wurde.

Parameters

- **speed** (*rotational speed: deg/s*) – Geschwindigkeit des Motors
- **rotation_angle** (*angle: deg*) – Winkel, um den sich der Motor drehen sollte
- **stop_type** (`Stop`) – Hiermit wird festgelegt, ob der Motor nach dem Anhalten auslaufen, abbremsen oder die Position beibehalten soll (*Standardeinstellung*: *`Stop.COAST`*).
- **wait** (*`bool`*) – legt fest, ob das Fahrmanöver zuerst beendet sein muss, bevor der Rest des Programms ausgeführt wird (*Standardeinstellung*: `True`); Bedeutet, dass das Programm wartet, bis sich der Motor exakt um den eingegebenen Winkel gedreht hat.

run_target (*speed, target_angle, stop_type=Stop.COAST, wait=True*)
Hiermit läuft der Motor bis zu einem bestimmten Zielwinkel mit einer konstanten Geschwindigkeit (Winkelgeschwindigkeit).

Der Motor beschleunigt bis zur festgelegten Geschwindigkeit und der Arbeitszyklus wird automatisch angepasst, um die Geschwindigkeit selbst bei einer gewissen Last konstant zu halten. Der Motor bremst gerade rechtzeitig ab, um beim festgelegten Zielwinkel zum Stillstand zu kommen.

Die Drehrichtung wird auf Grundlage des Zielwinkels automatisch ausgewählt.

Parameters

- **speed** (*rotational speed: deg/s*) – absolute Geschwindigkeit des Motors: Die Drehrichtung wird auf Grundlage des Zielwinkels automatisch ausgewählt – unabhängig davon, ob ein positiver oder ein negativer Wert für die Geschwindigkeit eingegeben wurde.
- **target_angle** (*angle: deg*) – Zielwinkel, bis zu dem sich der Motor drehen soll (unabhängig vom aktuellen Winkel)
- **stop_type** (`Stop`) – Hiermit wird festgelegt, ob der Motor nach dem Anhalten auslaufen, abbremsen oder die Position beibehalten soll (*Standardeinstellung*: *`Stop.COAST`*).
- **wait** (*`bool`*) – legt fest, ob das Fahrmanöver zuerst beendet sein muss, bevor der Rest des Programms ausgeführt wird (*Standardeinstellung*: `True`); Bedeutet, dass das Programm wartet, bis der Motor den Zielwinkel erreicht hat.

Erweiterte Funktionen für Motoren mit Drehsensor

track_target (*target_angle*)

Hiermit wird ein Zielwinkel nachverfolgt, der sich im Verlauf der Zeit ändert.

Diese Funktion ähnelt der Funktion *run_target()*, ignoriert aber die Einstellungen für Geschwindigkeit und Beschleunigung: Der Motor bewegt sich so schnell wie möglich zum Zielwinkel. Geschwindigkeit und Beschleunigung werden stattdessen festgelegt, indem man auswählt, wie schnell oder langsam sich der `target_angle` verändern soll.

Diese Methode ist für schnelle Schleifen gut geeignet, bei denen sich das Motorziel ständig ändert.

Parameters **target_angle** (*angle: deg*) – Zielwinkel, bis zu dem sich der Motor drehen soll

Beispiel:

```
# Initialize motor and timer
from math import sin
motor = Motor(Port.A)
watch = StopWatch()
amplitude = 90

# In a fast loop, compute a reference angle
# and make the motor track it.
while True:

    # Get the time in seconds
    seconds = watch.time()/1000
    # Compute a reference angle. This produces
    # a sine wave that makes the motor move
    # smoothly between -90 and +90 degrees.
    angle_now = sin(seconds)*amplitude
    # Make the motor track the given angle
    motor.track_target(angle_now)
```

stalled()

Hiermit wird überprüft, ob der Motor derzeit blockiert ist.

Ein Motor gilt als blockiert, wenn er sich selbst bei maximalem Drehmoment nicht bewegen kann. Dies ist beispielsweise der Fall, wenn der Motor durch etwas blockiert wird oder wenn sich der Mechanismus einfach nicht weiter drehen kann.

Genauer gesagt, ist der Motor blockiert, wenn der im PID-Regler programmierte Arbeitszyklus das Maximum erreicht hat (`duty = duty_limit`) und der Motor es in der mit `stall_time` festgelegten Zeit nicht schafft, die Mindestgeschwindigkeit (`speed < stall_speed`) zu erreichen.

Die Einstellungen können für `duty_limit`, `stall_speed` und `stall_time` mit *set_dc_settings()* und *set_pid_settings()* angepasst werden, um die Anfälligkeit für Blockierungen zu verändern.

Returns `True`, wenn der Motor blockiert ist, bzw. `False`, wenn er nicht blockiert ist

Return type bool

run_until_stalled (*speed, stop_type=Stop.COAST, duty_limit=default*)

Hiermit läuft der Motor mit einer konstanten Geschwindigkeit (Winkelgeschwindigkeit), bis er blockiert wird. Der Motor gilt als blockiert, wenn er sich selbst bei maximalem Drehmoment nicht bewegen kann. Für eine detailliertere Definition siehe *stalled()*.

Mit `duty_limit` lässt sich das Drehmoment des Motors während dieses Fahrmanövers kurzzeitig begrenzen. Diese Funktion ist nützlich, um zu verhindern, dass ein Getriebe- oder Hebelmechanismus mit dem vollen Motordrehmoment angetrieben wird.

Parameters

- **speed** (*rotational speed: deg/s*) – Geschwindigkeit des Motors
- **stop_type** (`Stop`) – Hiermit wird festgelegt, ob der Motor nach dem Anhalten auslaufen, abbremsen oder die Position beibehalten soll (*Standardeinstellung*: `Stop.COAST`).
- **duty_limit** (*percentage: %*) – relative Begrenzung des Drehmoments: Diese Begrenzung funktioniert auf dieselbe Weise wie `set_dc_settings()`. Allerdings ist die Begrenzung hier nur kurzfristig: Nachdem der Befehl ausgeführt wurde, kehrt das Programm zum ursprünglichen Wert zurück.

set_dc_settings (*duty_limit, duty_offset*)

Hiermit werden die Einstellungen vorgenommen, um das Verhalten des `dc()`-Befehls anzupassen. Das beeinflusst auch alle `run`-Befehle, die im Hintergrund mit der `dc()`-Methode laufen.

Parameters

- **duty_limit** (*percentage: %*) – relative Begrenzung des Drehmoments während der darauffolgenden Motorbefehle: Hiermit wird der maximale Arbeitszyklus für die darauffolgenden Motorbefehle eingestellt. Dadurch wird die maximale Drehmomentausgabe auf einen Prozentwert des maximalen Kippmoments reduziert. Mit dieser Funktion kann vermieden werden, dass ein Getriebe- oder Hebelmechanismus mit dem vollen Motordrehmoment angetrieben wird oder dass dein LEGO® Zahnradgetriebe aus Versehen mit voller Geschwindigkeit läuft (*Standardeinstellung*: 100).
- **duty_offset** (*percentage: %*) – Mindest-Arbeitszyklus für die Verwendung von `dc()`: Diese Funktion fügt ein kleines Vorwärtsdrehmoment hinzu, sodass sich der Motor auch bei sehr niedrigen Arbeitszykluswerten bewegt. Dies kann beim Entwickeln eigener Feedback-Regler nützlich sein (*Standardeinstellung*: 0).

set_run_settings (*max_speed, acceleration*)

Hiermit wird die maximale Geschwindigkeit und Beschleunigung/Abbremsung des Motors für alle `run`-Befehle eingerichtet.

Dies gilt für die Motorbefehle `run`, `run_time`, `run_angle`, `run_target` und `run_until_stalled`. Siehe auch die Standardparameter für jeden Motor.

Parameters

- **max_speed** (*rotational speed: deg/s*) – maximale Geschwindigkeit des Motors beim Ausführen eines Motorbefehls
- **acceleration** (*rotational acceleration: deg/s/s*) – Beschleunigung auf die Zielgeschwindigkeit und Abbremsung bis zum Stillstand. Dieser Wert sollte positiv sein. Bei Bedarf verändert der Motor das Vorzeichen automatisch, um abzubremsen.

Beispiel:

```
# Set the maximum speed to 200 deg/s and acceleration to 400 deg/s/s.
example_motor.set_run_settings(200, 400)

# Make the motor run for 5 seconds. Even though the speed argument is 300
# deg/s in this example, the motor will move at only 200 deg/s because of
# the settings above.
example_motor.run_time(300, 5000)
```

set_pid_settings (*kp, ki, kd, tight_loop_limit, angle_tolerance, speed_tolerance, stall_speed, stall_time*)

Hiermit werden die Einstellungen für die Positions- und Geschwindigkeitsregler eingerichtet. Siehe auch die PID- und Standardparameter für jeden Motor.

Parameters

- **kp** (*int*) – Regelkonstante für die proportionale Position (und die integrale Geschwindigkeit)
- **ki** (*int*) – Regelkonstante für die integrale Position
- **kd** (*int*) – Regelkonstante für die abgeleitete Position (und die proportionale Geschwindigkeit)
- **tight_loop_limit** (*time: ms*) – Wenn du irgendeinen **run**-Befehl innerhalb dieses Intervalls nach Beginn des vorherigen Befehls ausführst, nimmt der Regler an, dass du die Geschwindigkeit direkt steuern willst. Das bedeutet, dass der Regler die Beschleunigungseinstellung ignoriert und sofort damit beginnt, die Geschwindigkeit zu verfolgen, die du in dem **run**-Befehl eingegeben hast. Diese Funktion eignet sich gut für schnelle Schleifen, bei denen eine schnelle Reaktion des Motors erwünscht ist statt einer sanften Beschleunigung (z. B. bei einem Roboter, der einer Linie folgt).
- **angle_tolerance** (*angle: deg*) – erlaubte Abweichung vom Zielwinkel, bevor eine Bewegung als vollständig abgeschlossen erachtet wird
- **speed_tolerance** (*rotational speed: deg/s*) – erlaubte Abweichung von der Geschwindigkeit „Null“, bevor eine Bewegung als vollständig abgeschlossen erachtet wird
- **stall_speed** (*rotational speed: deg/s*) – siehe *stalled()*.
- **stall_time** (*time: ms*) – siehe *stalled()*.

4.2 Sensoren

4.2.1 Berührungssensor

class TouchSensor (*port*)

LEGO® MINDSTORMS® Education EV3 Berührungssensor

Element 95648/6138404, enthalten in:

- 31313: LEGO MINDSTORMS Education EV3 (2013)
- 45544: LEGO MINDSTORMS Education EV3 Set (2013)
- 45507: Separates Teil (2013)

Parameters **port** (*Port*) – Anschluss, an den der Sensor angeschlossen ist

pressed ()

Hiermit wird überprüft, ob der Sensor gedrückt wird.

Returns `True`, wenn der Sensor gedrückt wird, bzw. `False`, wenn er nicht gedrückt wird

Return type `bool`

4.2.2 Farbsensor

class ColorSensor (*port*)

LEGO® MINDSTORMS® Education EV3 Farbsensor

Element 95650/6128869, enthalten in:

- 31313: LEGO MINDSTORMS Education EV3 (2013)
- 45544: LEGO MINDSTORMS Education EV3 Set (2013)
- 45506: Separates Teil (2013)

Parameters **port** (`Port`) – Anschluss, an den der Sensor angeschlossen ist

color ()
Hiermit wird die Farbe einer Oberfläche erfasst.

Returns `Color.BLACK`, `Color.BLUE`, `Color.GREEN`, `Color.YELLOW`, `Color.RED`, `Color.WHITE`, `Color.BROWN` oder `None`

Return type `Color` oder `None`, falls keine Farbe erkannt wird

ambient ()
Hiermit wird die Stärke des Umgebungslichts gemessen.

Returns Stärke des Umgebungslichts von 0 (dunkel) bis 100 (hell)

Return type *percentage: %*

reflection ()
Hiermit wird die Reflexion eines roten Lichts auf einer Oberfläche gemessen.

Returns Reflexion von 0 (keine Reflexion) bis 100 (starke Reflexion)

Return type *percentage: %*

rgb ()
Hiermit wird die Reflexion eines roten, grünen und dann blauen Lichts auf einer Oberfläche gemessen.

Returns Reflexion des roten, grünen und blauen Lichts jeweils von 0,0 (keine Reflexion) bis 100,0 (starke Reflexion)

Return type Tupel mit drei *percentages*

4.2.3 Infrarotsensor und -sender

class InfraredSensor (*port*)
LEGO® MINDSTORMS® Education EV3 Infrarotsensor und Infrarotsender

Elemente 95654/6132629 und 72156/6127283, enthalten in:

- 31313: LEGO MINDSTORMS Education EV3 (2013)
- 45509 und 45508: Separates Teil (2013)

Parameters **port** (`Port`) – Anschluss, an den der Sensor angeschlossen ist

distance ()
Hiermit wird der relative Abstand zwischen dem Sensor und einem Objekt mithilfe von Infrarotlicht gemessen.

Returns relativen Abstand von 0 (nah) bis 100 (weit entfernt)

Return type *relative distance: %*

beacon (*channel*)
Hiermit wird der relative Abstand und der Winkel zwischen der Fernsteuerung und dem Infrarotsensor gemessen.

Parameters **channel** (*int*) – Kanalnummer der Fernsteuerung

Returns Tupel des relativen Abstands (0 bis 100) und den ungefähren Winkel (-75 bis 75 Grad) zwischen der Fernsteuerung und dem Infrarotsensor

Return type (*relative distance: %, angle: deg*) oder (`None`, `None`), wenn keine Fernsteuerung erkannt wird

buttons (*channel*)
Hiermit wird überprüft, welche Tasten auf der Infrarot-Fernsteuerung gedrückt werden.

Parameters **channel** (*int*) – Kanalnummer der Fernsteuerung

Returns Liste der gedrückten Tasten auf der Fernsteuerung für den jeweiligen Kanal

Return type Liste von *Buttons*

4.2.4 Ultraschallsensor

class UltrasonicSensor (*port*)
LEGO® MINDSTORMS® Education EV3 Ultraschallsensor

Element 95652/6138403, enthalten in:

- 45544: LEGO MINDSTORMS Education EV3 Set (2013)
- 45504: Separates Teil (2013)

Parameters **port** (`Port`) – Anschluss, an den der Sensor angeschlossen ist

distance (*silent=False*)
Hiermit wird mithilfe von Ultraschallwellen der Abstand zwischen dem Sensor und einem Objekt gemessen.

Parameters **silent** (*bool*) – Wähle `True` aus, um den Sensor nach dem Messen des Abstands auszuschalten.

Wähle `False` aus, um den Sensor eingeschaltet zu lassen (*Standardeinstellung*).

Wenn du `silent=True` auswählst, sendet der Sensor nur dann Schallwellen aus, wenn die Messung durchgeführt wird. Dadurch wird verhindert, dass andere Ultraschallsensoren gestört werden. Allerdings dauert das Abschalten des Sensors jeweils etwa 300 ms.

Returns Abstand (Millimeter)

Return type *distance: mm*

presence ()
Hiermit wird nach Ultraschallwellen gesucht, um zu überprüfen, ob sich andere Ultraschallsensoren in der Nähe befinden.

Wenn sich der andere Ultraschallsensor im Modus „silent“ befindet, kannst du diesen Sensor nur dann erfassen, wenn er gerade eine Messung durchführt.

Returns `True`, falls Ultraschall erfasst wird, bzw. `False`, falls kein Ultraschall erkannt wird

Return type `bool`

4.2.5 Gyrosensor

class GyroSensor (*port, direction=Direction.CLOCKWISE*)

LEGO® MINDSTORMS® Education EV3 Gyrosensor

Element 99380/6138411, enthalten in:

- 45544: LEGO MINDSTORMS Education EV3 Set (2013)
- 45505: Separates Teil (2013)

Parameters

- **port** (`Port`) – Anschluss, an den der Sensor angeschlossen ist
- **direction** (`Direction`) – positive Drehrichtung, wenn man auf den roten Punkt auf dem Sensor blickt (*Standardeinstellung*: `Direction.CLOCKWISE`)

speed ()

Hiermit wird die Geschwindigkeit (Winkelgeschwindigkeit) des Sensors abgerufen.

Returns Winkelgeschwindigkeit des Sensors

Return type *rotational speed: deg/s*

angle ()

Hiermit wird der kumulierte Winkel des Sensors abgerufen.

Returns Umdrehungswinkel

Return type *angle: deg*

reset_angle (*angle*)

Hiermit wird der Drehwinkel des Sensors auf einen Sollwert eingestellt.

Parameters **angle** (*angle: deg*) – Wert, auf den der Winkel zurückgesetzt werden soll

KAPITEL

FÜNF

`PARAMETER` – PARAMETER UND KONSTANTEN

Konstante Parameter/Argumente für Pybricks-Programmierschnittstelle

`class Port`

Anschluss am programmierbaren EV3 Stein

Motor-Anschlüsse:

`A`

`B`

`C`

`D`

Sensor-Anschlüsse:

`S1`

`S2`

`S3`

`S4`

`class Direction`

Drehrichtung für positive Geschwindigkeitswerte: im Uhrzeigersinn oder entgegen dem Uhrzeigersinn

`CLOCKWISE`

Bei einem positiven Geschwindigkeitswert sollte sich der Motor im Uhrzeigersinn drehen.

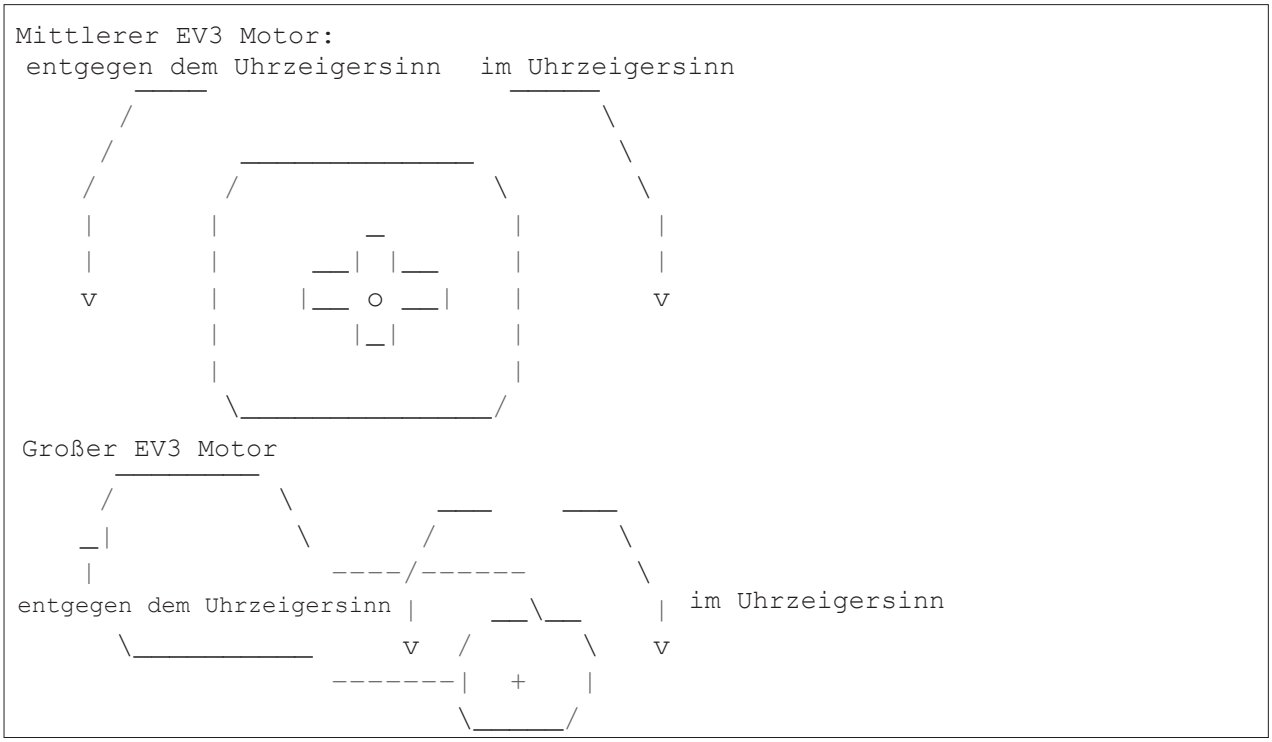
`COUNTERCLOCKWISE`

Bei einem positiven Geschwindigkeitswert sollte sich der Motor entgegen dem Uhrzeigersinn drehen.

Für alle Motoren gilt: Die Drehrichtung ist diejenige, in die sich der Motor dreht, wenn man auf die Welle blickt (wie bei einer Uhr).

Bei den NXT und EV3 Motoren solltest du darauf achten, den Motor so anzusehen, dass sich die rote/orange Welle unten rechts befindet.

Parameter	Positive Geschwindigkeit	Negative Geschwindigkeit
Direction.CLOCKWISE	im Uhrzeigersinn	entgegen dem Uhrzeigersinn
Direction.COUNTERCLOCKWISE	entgegen dem Uhrzeigersinn	im Uhrzeigersinn



class Stop

Hiermit wird festgelegt, was nach dem Anhalten des Motors passieren soll: auslaufen, abbremsen oder Winkel beibehalten.

COAST

Der Motor kann sich frei bewegen.

BRAKE

Der Motor hält kleinen äußeren Kräften passiv stand.

HOLD

Der Motor wird weiterhin gesteuert, um den eingegebenen Winkel beizubehalten. Diese Funktion ist nur bei Motoren mit Drehsensor verfügbar.

Die Art des Anhaltens legt fest, wie groß der Bewegungswiderstand des Motors nach dem Anhalten ist:

Parameter	Widerstand	Physikalische Bedeutung
Stop.COAST	niedrig	Reibung
Stop.BRAKE	mittel	Reibung + Drehmoment entgegen der Bewegung
Stop.HOLD	hoch	Reibung + Drehmoment zum Beibehalten des festgelegten Winkels

class Color

Licht oder Oberflächenfarbe

BLACK

BLUE

GREEN

YELLOW

RED

WHITE

BROWN

ORANGE

PURPLE

class Button
Tasten auf dem Stein oder der Fernsteuerung

LEFT_DOWN

DOWN

RIGHT_DOWN

LEFT

CENTER

RIGHT

LEFT_UP

UP

BEACON

RIGHT_UP

LEFT__UP	UP/BEACON	RIGHT__UP
LEFT	CENTER	RIGHT
LEFT__DOWN	DOWN	RIGHT__DOWN

class Align
Anordnung eines Bildes auf dem Display

BOTTOM_LEFT

BOTTOM

BOTTOM_RIGHT

LEFT

CENTER

RIGHT

TOP_LEFT

TOP

TOP_RIGHT

class ImageFile
Pfad zu EV3 Standardbildern

Information

RIGHT

FORWARD

ACCEPT

QUESTION_MARK

STOP_1

LEFT

DECLINE

THUMBS_DOWN

BACKWARD

NO_GO

WARNING

STOP_2

THUMBS_UP

LEGO

EV3

EV3_ICON

Objekte

TARGET

Augen

BOTTOM_RIGHT

BOTTOM_LEFT

EVIL

CRAZY_2

KNOCKED_OUT

PINCHED_RIGHT

WINKING

DIZZY

DOWN

TIRED_MIDDLE

MIDDLE_RIGHT

SLEEPING

MIDDLE_LEFT

TIRED_RIGHT

PINCHED_LEFT

PINCHED_MIDDLE

CRAZY_1

NEUTRAL

AWAKE

UP

TIRED_LEFT

ANGRY

class SoundFile

Pfad zu EV3 Standardklängen

Ausdrücke

SHOUTING

CHEERING

CRYING

OUCH

LAUGHING_2

SNEEZING

SMACK

BOING

BOO

UH_OH

SNORING

KUNG_FU

FANFARE

CRUNCHING

MAGIC_WAND

LAUGHING_1

Information

LEFT

BACKWARDS

RIGHT

OBJECT

COLOR

FLASHING

ERROR

ERROR_ALARM

DOWN

FORWARD

ACTIVATE

SEARCHING

TOUCH

UP

ANALYZE

STOP

DETECTED

TURN

START

Kommunikation

MORNING

EV3

GO

GOOD_JOB

OKEY_DOKEY

GOOD

NO

THANK_YOU

YES

GAME_OVER

OKAY

SORRY

BRAVO

GOODBYE

HI

HELLO

MINDSTORMS

LEGO

FANTASTIC

Bewegungen

SPEED_IDLE

SPEED_DOWN

SPEED_UP

Farbe

BROWN

GREEN

BLACK

WHITE

RED

BLUE

YELLOW

Mechanisch

TICK_TACK

HORN_1

BACKING_ALERT

MOTOR_IDLE

AIR_RELEASE

AIRBRAKE

RATCHET

MOTOR_STOP

HORN_2

LASER

SONAR

MOTOR_START

Tiere

INSECT_BUZZ_2

ELEPHANT_CALL

SNAKE_HISS

DOG_BARK_2

DOG_WHINE

INSECT_BUZZ_1

DOG_SNIFF

T_REX_ROAR

INSECT_CHIRP

DOG_GROWL

SNAKE_RATTLE

DOG_BARK_1

CAT_PURR

Zahlen

ZERO

ONE

TWO

THREE

FOUR

FIVE

SIX

SEVEN

EIGHT

NINE

TEN

System

READY

CONFIRM

GENERAL_ALERT

CLICK

OVERPOWER

KAPITEL

SECHS

TOOLS – ZEIT- UND DATENERFASSUNG

Häufig verwendete Tools für die Zeit- und Datenerfassung

print (*value, ..., sep, end, file, flush*)

Hiermit werden die Werte auf einer Terminal- oder Stream-Anwendung angezeigt.

Beispiel:

```
# Print some text
print("Hello, world")

# Print some text and a number
print("Value:", 5)
```

wait (*time*)

Hiermit pausiert das Benutzerprogramm eine bestimmte Zeit lang.

Parameters **time** (*time: ms*) – Wartezeit

class StopWatch

Eine Stoppuhr zum Messen von Zeitintervallen. Sie funktioniert ähnlich wie die Stoppuhr-Funktion von Smartphones.

time ()

Hiermit wird die aktuelle Zeit auf der Stoppuhr abgerufen.

Returns verstrichene Zeit

Return type *time: ms*

pause ()

Hiermit wird die Stoppuhr angehalten.

resume ()

Hiermit wird die Stoppuhr wieder gestartet.

reset ()

Hiermit wird die Zeit auf der Stoppuhr auf 0 zurückgesetzt.

Der Betriebszustand bleibt unverändert:

- Wenn die Stoppuhr angehalten wurde, bleibt sie stehen (steht jetzt aber auf 0).
- Wenn die Stoppuhr lief, läuft sie weiter (beginnt aber erneut von 0).

KAPITEL

SIEBEN

ROBOTIK – ROBOTIKMODUL

Robotikmodul für die Pybricks-Programmierschnittstelle

class DriveBase (*left_motor, right_motor, wheel_diameter, axle_track*)

Roboterfahrzeug mit zwei Antriebsrädern und einem oder mehreren optionalen Laufrädern

Parameters

- **left_motor** (Motor) – der Motor, der das linke Rad antreibt
- **right_motor** (Motor) – der Motor, der das rechte Rad antreibt
- **wheel_diameter** (*dimension: mm*) – Durchmesser der Räder
- **axle_track** (*dimension: mm*) – Abstand zwischen den Mittelpunkten beider Räder

Beispiel:

```
# Initialize two motors and a drive base
left = Motor(Port.B)
right = Motor(Port.C)
robot = DriveBase(left, right, 56, 114)
```

drive (*speed, steering*)

Hiermit wird der Roboter mit der festgelegten Fahr- und Lenkgeschwindigkeit gestartet. Beides wird am Mittelpunkt zwischen den Rädern des Roboters gemessen.

Parameters

- **speed** (*speed: mm/s*) – Fahrgeschwindigkeit des Roboters
- **steering** (*rotational speed: deg/s*) – Lenkgeschwindigkeit des Roboters

Beispiel:

```
# Initialize two motors and a drive base
left = Motor(Port.B)
right = Motor(Port.C)
robot = DriveBase(left, right, 56, 114)

# Initialize a sensor
sensor = UltrasonicSensor(Port.S4)

# Drive forward until an object is detected
robot.drive(100, 0)
while sensor.distance() > 500:
    wait(10)
robot.stop()
```

drive_time(*speed, steering, time*)

Hiermit fährt der Roboter eine bestimmte Zeit lang mit der festgelegten Fahr- und Lenkgeschwindigkeit und hält dann an.

Parameters

- **speed**(*speed: mm/s*) – Fahrgeschwindigkeit des Roboters
- **steering**(*rotational speed: deg/s*) – Lenkgeschwindigkeit
- **time**(*time: ms*) – Dauer des Fahrmanövers

Beispiel:

```
# Drive forward at 100 mm/s for two seconds
robot.drive(100, 0, 2000)

# Turn at 45 deg/s for three seconds
robot.drive(0, 45, 3000)
```

stop(*stop_type=Stop.COAST*)

Hiermit wird der Roboter angehalten.

Parameters **stop_type**(`Stop`) – legt fest, ob der Motor auslaufen, abbremsen oder die Position beibehalten soll (*Standardeinstellung*: `Stop.COAST`)

KAPITEL

ACHT

SIGNALE UND EINHEITEN

Bei vielen Programmbefehlen musst du physikalische Größen in bestimmten Einheiten angeben. Diese Seite gibt einen Überblick über jede Größe und ihre Einheit.

8.1 Zeit: ms

Alle Werte, die die Zeit oder Dauer betreffen, werden in Millisekunden (ms) gemessen.

Zum Beispiel werden die mit `run_time` festgelegte Bewegungsdauer, die mit `wait` festgelegte Wartezeit oder die von der `StopWatch` gemeldeten Zeiten in Millisekunden angegeben.

8.2 Winkel: deg

Alle Winkel werden in Grad (deg) gemessen. Eine volle Umdrehung entspricht 360 Grad.

Zum Beispiel werden Winkelwerte für den `Motor` oder den `GyroSensor` in Grad angegeben.

8.3 Umdrehungsgeschwindigkeit: deg/s

Die Umdrehungsgeschwindigkeit oder Winkelgeschwindigkeit beschreibt, wie schnell sich etwas dreht. Sie wird in Grad pro Sekunde (deg/s) angegeben.

Zum Beispiel werden Umdrehungswerte für den `Motor` oder den `GyroSensor` in Grad pro Sekunde angegeben.

Wir empfehlen dir, in deinem Programm mit Grad pro Sekunde zu arbeiten. Wenn du andere Einheiten nutzen willst, verwende die folgende Umrechnungstabelle (z. B. Umdrehungen pro Minute in der englischen Einheit rpm, die der SI-Einheit min-1 entspricht).

	deg/s	rpm
1 deg/s =	1	1/6=0,167
1 rpm =	6	1

8.4 Abstand: mm

Abstände werden, wenn möglich, in Millimetern (mm) angegeben.

Beispielsweise wird der Abstandswert des `UltrasonicSensor` in Millimetern angegeben.

Wir empfehlen dir, in deinem Programm mit Millimetern zu arbeiten. Wenn du andere Einheiten nutzen willst, verwende die folgende Umrechnungstabelle.

	mm	cm	inch (Zoll)
1 mm =	1	0,1	0,0394
1 cm =	10	1	0,394
1 inch (Zoll) =	25,4	2,54	1

8.5 Abmessung: mm

Abmessungen werden wie Abstände möglichst in Millimetern (mm) angegeben.

Beispielsweise wird der Raddurchmesser in Millimetern gemessen.

8.6 Relativer Abstand: %

Bei einigen Abstandsmessungen gibt es keinen genauen Wert in einer bestimmten Einheit. Stattdessen gibt es einen Bereich von sehr nah (0 %) bis sehr weit entfernt (100 %). Dabei handelt es sich um relative Abstände.

Beispielsweise beschreibt der Abstandswert des `InfraredSensor` einen relativen Abstand.

8.7 Geschwindigkeit: mm/s

Lineare Geschwindigkeiten werden in Millimetern pro Sekunde (mm/s) angegeben.

Beispielsweise wird die Geschwindigkeit eines Roboterfahrzeugs in mm/s gemessen.

8.8 Umdrehungsbeschleunigung: deg/s/s

Die Umdrehungsbeschleunigung oder *Winkelbeschleunigung* beschreibt, wie schnell sich die Umdrehungsgeschwindigkeit verändert. Sie wird als Veränderung der Anzahl an Grad pro Sekunde innerhalb einer Sekunde (deg/s/s) angegeben. Dies wird häufig auch als deg/s^2 ausgedrückt.

Beispielsweise kann man die Umdrehungsbeschleunigung von einem `Motor` so einstellen, dass der Sollwert für die konstante Geschwindigkeit schnell oder langsam erreicht wird.

8.9 Prozent: %

Für einige Signale gibt es keine bestimmte Einheit. Stattdessen werden sie in einem Bereich vom Minimum (0 %) bis zum Maximum (100 %) angegeben. *Relative Abstände* gehören zu dieser Art von Signal, die in Prozent angegeben werden.

Beispielsweise gibt es für die *Lautstärke* einen Signalbereich von 0 bis 100 %.

8.10 Frequenz: Hz

Klangfrequenzen werden in Hertz (Hz) angegeben.

Beispielsweise kann man die Frequenz eines *Pieptons* festlegen, um die Tonhöhe zu ändern.

8.11 Spannung: mV

Spannungen werden in Millivolt (mV) angegeben.

Zum Beispiel die Spannung des *Akkus*.

8.12 Strom: mA

Elektrischer Strom wird in Milliampere (mA) angegeben.

Zum Beispiel der vom *Akku* gelieferte Strom.

KAPITEL

NEUN

ROBOT EDUCATOR

Mit diesem Beispielprogramm fährt der Robot Educator (Abbildung 9.1) so lange, bis er ein Hindernis erkennt. Daraufhin fährt er rückwärts, dreht sich und fährt dann weiter.

Die Bauanleitungen für den Robot Educator findest du auf der LEGO Education Website.

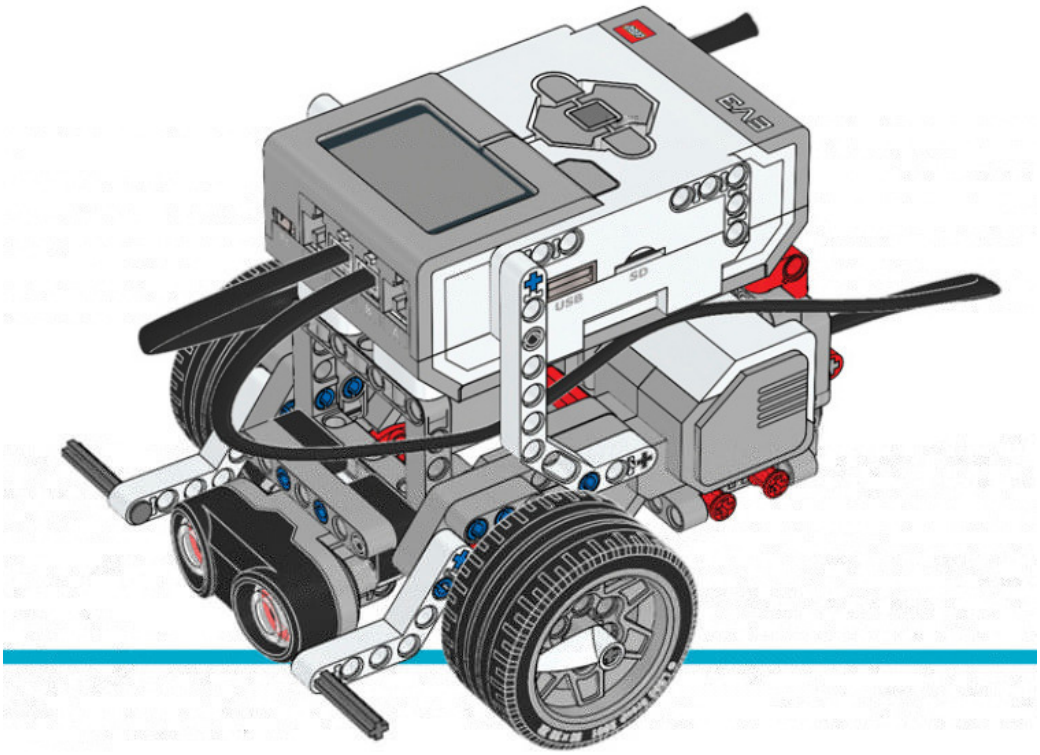


Abbildung 9.1: Robot Educator mit Ultraschallsensor

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import Motor, TouchSensor, ColorSensor
from pybricks.parameters import Port, Button, Color, ImageFile, SoundFile
from pybricks.tools import wait
from pybricks.robotics import DriveBase

# Play a sound.
brick.sound.beep()

# Initialize the Ultrasonic Sensor. It is used to detect
# obstacles as the robot drives around.
obstacle_sensor = UltrasonicSensor(Port.S4)

# Initialize two motors with default settings on Port B and Port C.
# These will be the left and right motors of the drive base.
left_motor = Motor(Port.B)
right_motor = Motor(Port.C)

# The wheel diameter of the Robot Educator is 56 millimeters.
wheel_diameter = 56

# The axle track is the distance between the centers of each of the wheels.
# For the Robot Educator this is 114 millimeters.
axle_track = 114

# The DriveBase is composed of two motors, with a wheel on each motor.
# The wheel_diameter and axle_track values are used to make the motors
# move at the correct speed when you give a motor command.
robot = DriveBase(left_motor, right_motor, wheel_diameter, axle_track)

# The following loop makes the robot drive forward until it detects an
# obstacle. Then it backs up and turns around. It keeps on doing this
# until you stop the program.
while True:
    # Begin driving forward at 200 millimeters per second.
    robot.drive(200, 0)

    # Wait until an obstacle is detected. This is done by repeatedly
    # doing nothing(waiting for 10 milliseconds) while the measured
    # distance is still greater than 300 mm.
    while obstacle_sensor.distance() > 300:
        wait(10)

    # Drive backward at 100 millimeters per second. Keep going for 2 seconds.
    robot.drive_time(-100, 0, 2000)

    # Turn around at 60 degrees per second, around the midpoint between
    # the wheels. Keep going for 2 seconds.
    robot.drive_time(0, 60, 2000)
```

KAPITEL

ZEHN

FARBSORTIERER

Mit diesem Beispielprogramm für den Farbsortierer (Abbildung 10.1) kannst du verschiedenfarbige LEGO Technic Steine mithilfe des Farbsensors sortieren.

Scanne einen Stein nach dem anderen und lege sie in die Halterung. Ein Piepton bestätigt, dass die Farbe erkannt wurde. Wenn die Halterung voll ist oder du auf die mittlere Taste drückst, beginnt der Roboter, die Elemente nach Farben zu sortieren.

Die Bauanleitungen für den Farbsortierer findest du auf der LEGO® Education Website.

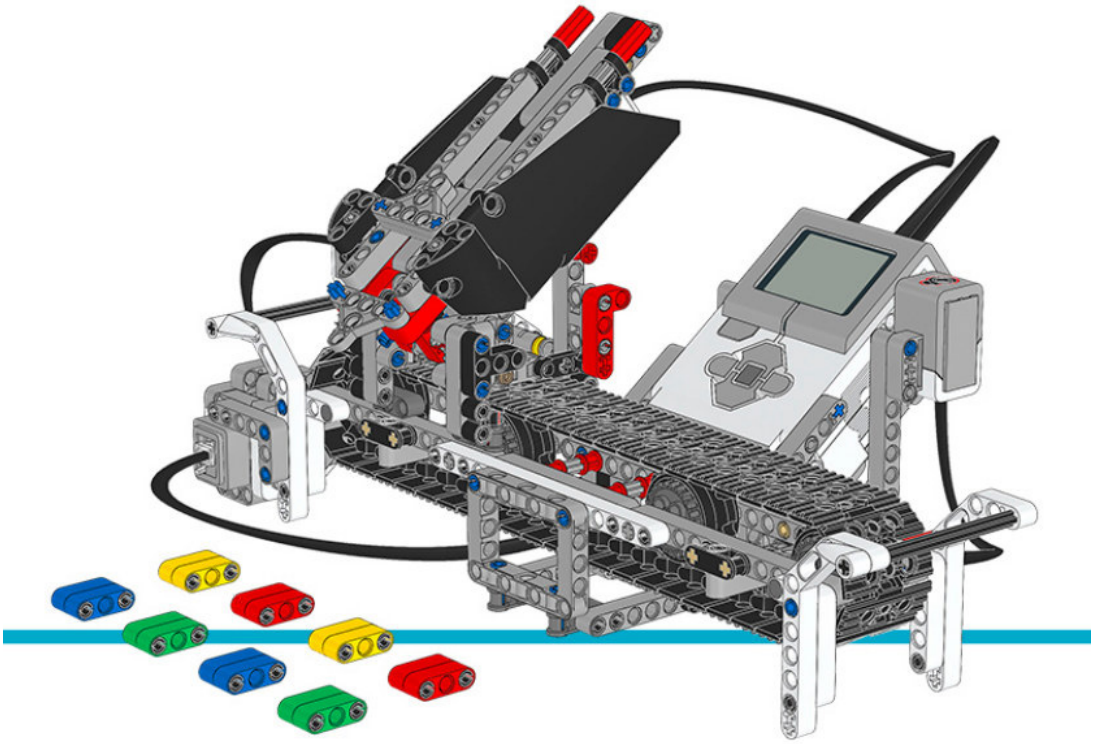


Abbildung 10.1: Farbsortierer

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import Motor, TouchSensor, ColorSensor
from pybricks.parameters import Port, Button, Color, ImageFile, SoundFile
from pybricks.tools import wait

# The colored objects are either red, green, blue, or yellow.
POSSIBLE_COLORS =(Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW)

# Initialize the motors that drive the conveyor belt and eject the objects.
belt_motor = Motor(Port.D)
feed_motor = Motor(Port.A)

# Initialize the Touch Sensor. It is used to detect when the belt motor
# has moved the sorter module all the way to the left.
touch_sensor = TouchSensor(Port.S1)

# Initialize the Color Sensor. It is used to detect the color of the objects.
color_sensor = ColorSensor(Port.S3)

# This is the main loop. It waits for you to scan and insert 8 colored objects.
# Then it sorts them by color. Then the process starts over and you can scan
# and insert the next set of colored objects.
while True:
    # Get the feed motor in the correct starting position.
    # This is done by running the motor forward until it stalls. This
    # means that it cannot move any further. From this end point, the motor
    # rotates backward by 180 degrees. Then it is in the starting position.
    feed_motor.run_until_stalled(120)
    feed_motor.run_angle(450, -180)

    # Get the conveyor belt motor in the correct starting position.
    # This is done by first running the belt motor backward until the
    # touch sensor becomes pressed. Then the motor stops, and the the angle is
    # reset to zero. This means that when it rotates backward to zero later
    # on, it returns to this starting position.
    belt_motor.run(-500)
    while not touch_sensor.pressed():
        pass
    belt_motor.stop()
    wait(1000)
    belt_motor.reset_angle(0)

    # Clear all the contents from the display.
    brick.display.clear()

    # When we scan the objects, we store all the color numbers in a list.
    # We start with an empty list. It will grow as we add colors to it.
    color_list = []

    # This loop scans the colors of the objects. It repeats until 8 objects
    # are scanned and placed in the chute. This is done by repeating the loop
    # while the length of the list is still less than 8.
    while len(color_list) < 8:
      # Show an arrow that points to the color sensor.
      brick.display.image(ImageFile.RIGHT)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung von der vorherigen Seite)

```
    # Show how many colored objects we have already scanned.
    brick.display.text(len(color_list))

    # Wait for the center button to be pressed or a color to be scanned.
    while True:
        # Store True if the center button is pressed or False if not.
        pressed = Button.CENTER in brick.buttons()
        # Store the color measured by the Color Sensor.
        color = color_sensor.color()
        # If the center button is pressed or a color is detected,
        # break out of the loop.
        if pressed or color in POSSIBLE_COLORS:
            break

    if pressed:
        # If the button was pressed, end the loop early.
        # We will no longer wait for any remaining objects
        # to be scanned and added to the chute.
        break
    else:
        # Otherwise, a color was scanned.
        # So we add(append) it to the list.
        brick.sound.beep(1000, 100, 100)
        color_list.append(color)

        # We don't want to register the same color once more if we're
        # still looking at the same object. So before we continue, we
        # wait until the sensor no longer sees the object.
        while color_sensor.color() in POSSIBLE_COLORS:
            pass
        brick.sound.beep(2000, 100, 100)

        # Show an arrow pointing to the center button,
        # to ask if we are done.
        brick.display.image(ImageFile.BACKWARD)
        wait(2000)

# Play a sound and show an image to indicate that we are done scanning.
brick.sound.file(SoundFile.READY)
brick.display.image(ImageFile.EV3)

# Now sort the bricks according the list of colors that we stored.
# We do this by going over each color in the list in a loop.
for color in color_list:

    # Wait for one second between each sorting action.
    wait(1000)

    # Run the conveyor belt motor to the right position based on the color.
    if color == Color.BLUE:
        brick.sound.file(SoundFile.BLUE)
        belt_motor.run_target(500, 10)
    elif color == Color.GREEN:
    brick.sound.file(SoundFile.GREEN)
        belt_motor.run_target(500, 132)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung von der vorherigen Seite)

```
    elif color == Color.YELLOW:
        brick.sound.file(SoundFile.YELLOW)
        belt_motor.run_target(500, 360)
    elif color == Color.RED:
        brick.sound.file(SoundFile.RED)
        belt_motor.run_target(500, 530)

    # Now that the conveyor belt is in the correct position,
    # eject the colored object.
    feed_motor.run_angle(1500, 90)
    feed_motor.run_angle(1500, -90)
```

KAPITEL

ELF

ROBOTERARM H25

Mit diesem Beispielprogramm bewegt der Roboter (Abbildung 11.1) die Stapel schwarzer Radnaben umher. Der Roboterarm wird zunächst initialisiert und beginnt dann, die Naben umherzubewegen.

Die Bauanleitungen für den Roboter findest du auf der LEGO Education Website.

Tipp: Wenn du den Roboter baust, richte den EV3 Stein so aus, dass die microSD-Karte leicht zugänglich ist.

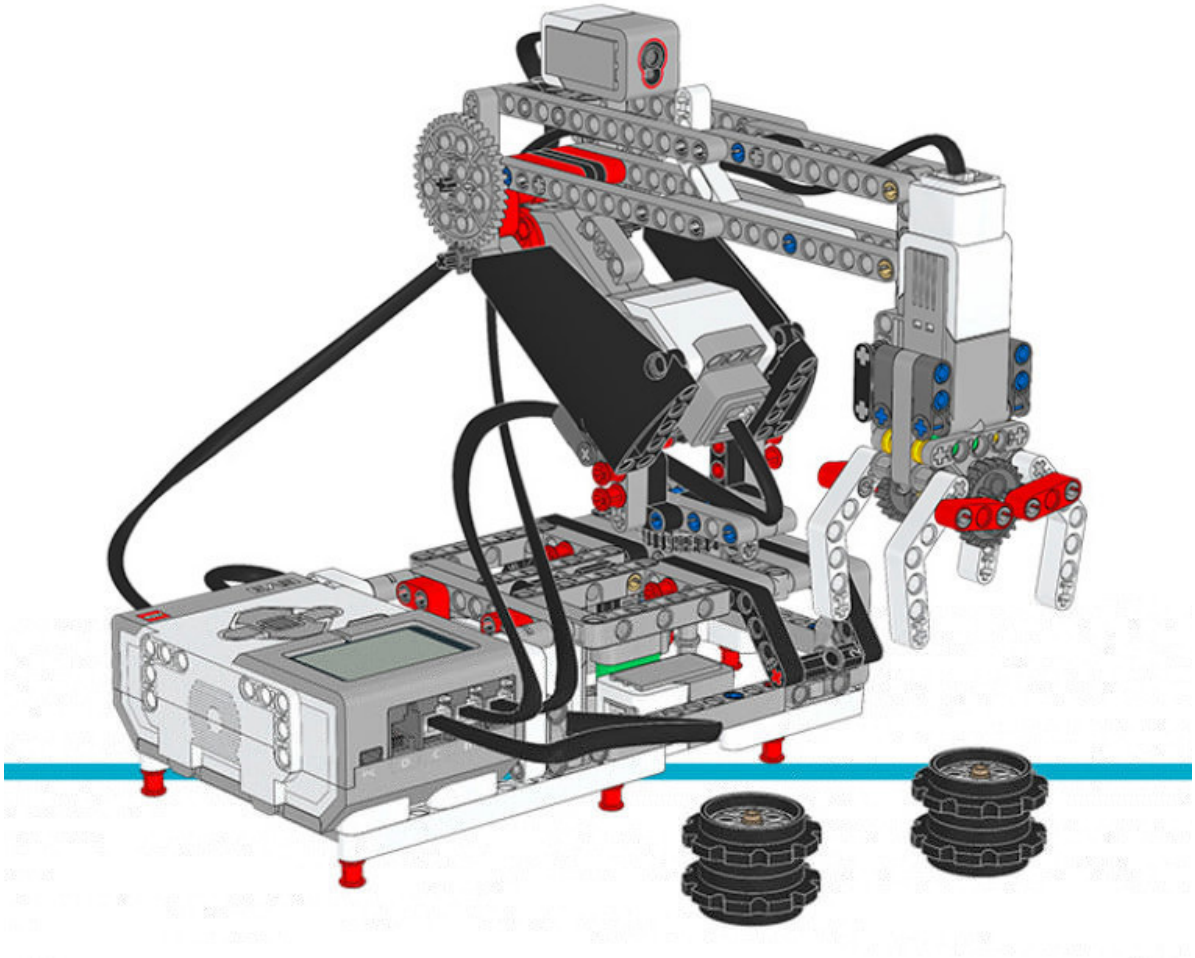


Abbildung 11.1: Roboterarm H25

(Fortsetzung von der vorherigen Seite)

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import Motor, TouchSensor, ColorSensor
from pybricks.parameters import Port, Stop, Direction
from pybricks.tools import wait

# Configure the gripper motor on Port A with default settings.
gripper_motor = Motor(Port.A)

# Configure the elbow motor. It has an 8-teeth and a 40-teeth gear
# connected to it. We would like positive speed values to make the
# arm go upward. This corresponds to counterclockwise rotation
# of the motor.
elbow_motor = Motor(Port.B, Direction.COUNTERCLOCKWISE, [8, 40])

# Configure the motor that rotates the base. It has a 12-teeth and a
# 36-teeth gear connected to it. We would like positive speed values
# to make the arm go away from the Touch Sensor. This corresponds
# to counterclockwise rotation of the motor.
base_motor = Motor(Port.C, Direction.COUNTERCLOCKWISE, [12, 36])

# Limit the elbow and base accelerations. This results in
# very smooth motion. Like an industrial robot.
elbow_motor.set_run_settings(60, 120)
base_motor.set_run_settings(60, 120)

# Set up the Touch Sensor. It acts as an end-switch in the base
# of the robot arm. It defines the starting point of the base.
base_switch = TouchSensor(Port.S1)

# Set up the Color Sensor. This sensor detects when the elbow
# is in the starting position. This is when the sensor sees the
# white beam up close.
elbow_sensor = ColorSensor(Port.S3)

# Initialize the elbow. First make it go down for one second.
# Then make it go upwards slowly(15 degrees per second) until
# the Color Sensor detects the white beam. Then reset the motor
# angle to make this the zero point. Finally, hold the motor
# in place so it does not move.
elbow_motor.run_time(-30, 1000)
elbow_motor.run(15)
while elbow_sensor.reflection() < 32:
    wait(10)
elbow_motor.reset_angle(0)
elbow_motor.stop(Stop.HOLD)

# Initialize the base. First rotate it until the Touch Sensor
# in the base is pressed. Reset the motor angle to make this
# the zero point. Then hold the motor in place so it does not move.
base_motor.run(-60)
while not base_switch.pressed():
    wait(10)
base_motor.reset_angle(0)
base_motor.stop(Stop.HOLD)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung von der vorherigen Seite)

```
# Initialize the gripper. First rotate the motor until it stalls.
# Stalling means that it cannot move any further. This position
# corresponds to the closed position. Then rotate the motor
# by 90 degrees such that the gripper is open.
gripper_motor.run_until_stalled(200, Stop.COAST, 50)
gripper_motor.reset_angle(0)
gripper_motor.run_target(200, -90)

def robot_pick(position):
    # This function makes the robot base rotate to the indicated
    # position. There it lowers the elbow, closes the gripper, and
    # raises the elbow to pick up the object.

    # Rotate to the pick-up position.
    base_motor.run_target(60, position, Stop.HOLD)
    # Lower the arm.
    elbow_motor.run_target(60, -40)
    # Close the gripper to grab the wheel stack.
    gripper_motor.run_until_stalled(200, Stop.HOLD, 50)
    # Raise the arm to lift the wheel stack.
    elbow_motor.run_target(60, 0, Stop.HOLD)

def robot_release(position):
    # This function makes the robot base rotate to the indicated
    # position. There it lowers the elbow, opens the gripper to
    # release the object. Then it raises its arm again.

    # Rotate to the drop-off position.
    base_motor.run_target(60, position, Stop.HOLD)
    # Lower the arm to put the wheel stack on the ground.
    elbow_motor.run_target(60, -40)
    # Open the gripper to release the wheel stack.
    gripper_motor.run_target(200, -90)
    # Raise the arm.
    elbow_motor.run_target(60, 0, Stop.HOLD)

# Play three beeps to indicate that the initialization is complete.
brick.sound.beeps(3)

# Define the three destinations for picking up and moving the wheel stacks.
LEFT = 160
MIDDLE = 100
RIGHT = 40

# This is the main part of the program. It is a loop that repeats endlessly.
#
# First, the robot moves the object on the left towards the middle.
# Second, the robot moves the object on the right towards the left.
# Finally, the robot moves the object that is now in the middle, to the right.
#
# Now we have a wheel stack on the left and on the right as before, but they
# have switched places. Then the loop repeats to do this over and over.
while True:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung von der vorherigen Seite)

```
    # Move a wheel stack from the left to the middle.
    robot_pick(LEFT)
    robot_release(MIDDLE)

    # Move a wheel stack from the right to the left.
    robot_pick(RIGHT)
    robot_release(LEFT)

    # Move a wheel stack from the middle to the right.
    robot_pick(MIDDLE)
    robot_release(RIGHT)
```

PYTHON-MODULVERZEICHNIS

VERZEICHNIS

L

M

P

R

S

T

U

V

W