3.1.5 Einführung in die Bedienung der Python-Programmierumgebung

Python ist eine textbasierte Programmiersprache, die der Kategorie der höheren Programmiersprachen zugeordnet wird. In den meisten Fällen werden Python-Programme interpretiert. Dies bedeutet, dass kein Compiler vorab das Programm in Maschinencode übersetzt, sondern erst zur Laufzeit die Übersetzung passiert. Dies hat den Vorteil, dass Skripte direkt ausgeführt werden können, was auch bei LEGO der Fall ist. Nachteilig an diesem Vorgehen ist, dass die korrekte Ausführbarkeit erst während der Laufzeit geprüft wird und somit Fehler im Programmcode nicht von der LEGO-Programmierumgebung beim Erstellen der Programme geprüft und gemeldet werden können (z. B. fehlerhaft genutzte Variablen). Python hat das Ziel, einen gut lesbaren, knappen Programmierstil zu unterstützen, was beispielsweise dazu führt, dass Einrückungen anstelle von geschweiften Klammern für die Strukturierung von Blöcken genutzt werden müssen.

Python ist eine objektorientierte Programmiersprache, unterstützt aber auch andere Programmierparadigmen wie die aspektorientierte und die funktionale Programmierung. Variablen erhalten in Python keinen festen Typ, vielmehr bietet Python eine dynamische Typisierung an.

Python bietet wie Java für die Behandlung von Fehlern die Ausnahmebehandlung (Englisch: exception handling), was kurz in Kapitel 6.6 aufgegriffen wird.

Für die Ansteuerung des Hubs, der Motoren und Sensoren liefert LEGO eine vollständige aus verschiedenen Modulen bestehende Python-Bibliothek mit. Damit lassen sich die verfügbaren Komponenten vollständig ansteuern und verwalten.

Die folgenden Klassen spiegeln dabei die wichtigsten Bereiche wider.

Python-Klassen	Kurzerläuterung
App	Wiedergabe von Musik auf dem verbundenen Computer/Tablet
Button	Abfrage des Status der linken oder rechten Taste des Hubs, Zugriff über die Hub-Klasse (MSHub bzw. PrimeHub)
ColorSensor	Zugriff auf den Farbsensor zum Auslesen der Sensorwerte und Warten auf Ereignisse
ForceSensor	Zugriff auf den Kraftsensor (nur LEGO Spike Prime) zum Auslesen der Sensorwerte und Warten auf Ereignisse
DistanceSensor	Zugriff auf den Ultraschallsensor zum Auslesen der Sensorwerte und Warten auf Ereignisse
LightMatrix	Ansteuerung der Lichtmatrix, Zugriff über die Hub-Klasse (MSHub bzw. PrimeHub)
MotionSensor	Abfrage des Status des Kreiselsensors (Drehwinkel, Orientierung, Gesten o. ä.), Zugriff über die Hub-Klasse (MSHub bzw. PrimeHub)
Motor	Zugriff auf einen einzelnen Motor zum Auslesen des Drehwinkels und zum Starten und Stoppen des Motors
MotorPair	Zugriff auf zwei Motoren zur gleichzeitigen Ansteuerung

Python-Klassen	Kurzerläuterung
PrimeHub/MSHub	Zugriff auf den Hub und die darin integrierten Komponenten (Lichtmatrix, Kreiselsensor, Tasten, Lautsprecher)
Speaker	Zugriff auf den Lautsprecher und Ausgabe von Tönen
StatusLight	Ansteuerung des Statuslichts des Hubs in der Zentraltaste, Zugriff über die Hub-Klasse (MSHub bzw. PrimeHub)
Timer	Zugriff auf den Zeitgeber zur Abfrage von Systemstartzeiten

Die Ansteuerung der Motoren und Sensoren wird durch Übergabe bestimmter Parameter geregelt, die den jeweiligen Port bestimmen. Dabei können der Klasse `MotorPair` auch mehrere Ports übergeben werden, um zum Beispiel zwei Motoren gleichzeitig zu starten.

Beispielcode	Beschreibung
`motor_pair = MotorPair('A', 'B')` `motor_pair.move(10, ,cm')`	Initialisiert ein Motor-Paar, welches an die Ports A und B angeschlossen ist, startet beide Motoren für eine Fahrt von 10 cm (bezogen auf die Größe der Standard-Räder des Sets)
`motor = Motor('A')` `motor.run_for_degrees(90)`	Initialisiert den Motor an Port A und lässt diesen um 90 Grad im Uhrzeigersinn drehen

Die Python-Programmierumgebung von LEGO erzeugt beim Anlegen eines neuen Python-Projekts automatisch ein Grundgerüst eines Python-Programms. Damit die Klassen für die Programmierung des Hubs genutzt werden können, müssen diese in das Programm importiert werden. Details zum Thema Import werden noch u. a. in Abschnitt 3.1.7 behandelt. Das Grundgerüst enthält bereits die wichtigsten Imports von Klassen, die für die gewöhnliche Programmierung benötigt werden. Für die Aufgaben in diesem Buch sind nicht alle Imports notwendig, können aber einfach bestehen bleiben. Wenn weitere Module, Klassen oder Funktionen benötigt werden, wird dies bei der jeweiligen Aufgabe behandelt. An dieser Stelle besteht allerdings ein Unterschied zwischen LEGO Mindstorms und LEGO Spike Prime. Die relevanten Python-Klassen sind bei LEGO Mindstorms im Modul `mindstorms` zu finden, bei LEGO Spike Prime im Modul `spike`. Außerdem ist die Klasse zum Ansteuern des Hubs in den beiden Modulen unterschiedlich benannt. Bei LEGO Mindstorms heißt die Klasse `MSHub` bei LEGO Spike Prime `PrimeHub`.

Bei LEGO Mindstorms sieht das Grundgerüst eines Python-Programms, welches die Programmierumgebung erzeugt, folgendermaßen aus:

```
from mindstorms import MSHub, Motor, MotorPair, ColorSensor, DistanceSensor, App
from mindstorms.control import wait_for_seconds, wait_until, Timer
from mindstorms.operator import greater_than, greater_than_or_equal_to, less_
than, less_than_or_equal_to, equal_to, not_equal_to
import math
```

```
# Create your objects here.
hub = MSHub()

# Write your program here.
hub.speaker.beep()
```

Bei LEGO Spike Prime sieht das Grundgerüst eines Python-Programms, welches die Programmierumgebung erzeugt, folgendermaßen aus:

```
from spike import PrimeHub, LightMatrix, Button, StatusLight, ForceSensor,
MotionSensor, Speaker, ColorSensor, App, DistanceSensor, Motor, MotorPair
from spike.control import wait_for_seconds, wait_until, Timer
from math import *

hub = PrimeHub()

hub.light_matrix.show_image('HAPPY')
```

Hinweis für die weitere Nutzung der Python-Beispielprogramme

Im weiteren Verlauf des Buchs werden bei den Beispielprogrammen die notwendigen Import-Anweisungen nicht mehr explizit mit abgedruckt, da von den Standard-Importen ausgegangen wird. Zusätzlich notwendige Importe werden bei den jeweiligen Aufgaben erläutert.

Außerdem wird bei Zugriff auf den Hub die Klasse des LEGO Mindstorms `MSHub` abgedruckt. Alle Nutzer des LEGO Spike Prime müssen diese durch `PrimeHub` in den Beispielprogrammen ersetzen.

Alle Beispielprogramme stehen sowohl für den LEGO Mindstorms als auch LEGO Spike Prime auf der Verlagsseite zum Download zur Verfügung.

Die Onlinehilfe für die Python-Programmierumgebung kann über das Symbol auf der rechten Bildschirmseite eingeblendet werden. Über dieses Symbol lässt sich die Hilfe auch vergrößern und verkleinern. Bei LEGO Mindstorms wird das Symbol  genutzt, bei LEGO Spike Prime  . Es öffnet sich daraufhin die Wissensdatenbank.

In der Software LEGO SPIKE 1.3.5 liegt diese in deutscher Sprache vor. In LEGO Mindstorms 10.1.0 wird sie nur in englischer Sprache angeboten und sieht folgendermaßen aus:

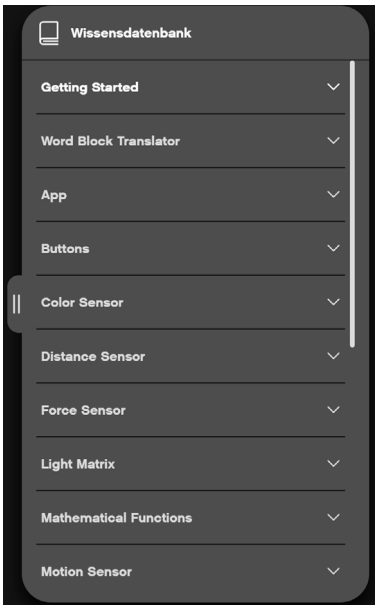


Abb. 3–10 // Python-Wissensdatenbank und Onlinehilfe beim LEGO Mindstorms

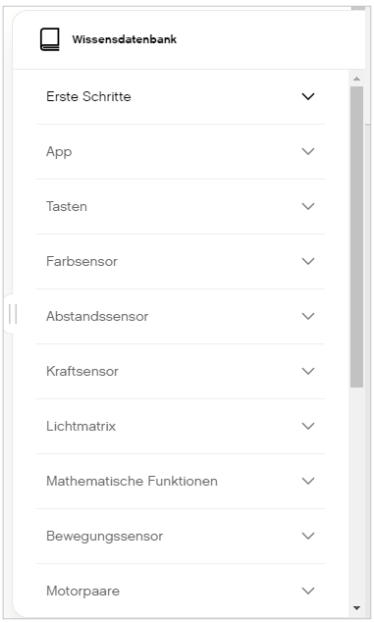


Abb. 3–11 // Python-Wissensdatenbank und Onlinehilfe beim LEGO Spike Prime

Einen sehr wichtigen Bereich der Programmierumgebung stellt die Konsolenausgabe dar. Bei einem Fehler im Programm werden dort die relevanten Fehler ausgegeben. Weiterhin kann das Programm auf der Konsole Statusinformationen oder Meldungen ausgeben, indem es die Funktion `print` nutzt. Ohne die Konsole sind aufgrund des fehlenden Displays im Hub Fehler in einem Python-Programm schwer bis gar nicht zu lokalisieren. Die Konsole wird durch das Symbol in der Mitte im unteren Bereich sichtbar gemacht.

Bei LEGO Mindstorms wird dabei das Symbol genutzt, bei LEGO Spike Prime das Symbol . Das folgende einfache Beispielprogramm gibt »Hallo Welt« auf der Konsole aus.

```
print('Hallo Welt')
```

In der Konsole wird die aktuelle Uhrzeit jeweils mit ausgegeben, was somit zu folgender Ausgabe führt:

Abb. 3–12 // Darstellung der Python-Konsole mit Ausgabe

Im Gegensatz zur Programmieroberfläche mit den Textblöcken ist ein Kopieren des Quellcodes mit Strg+C und das Einfügen mit Strg+V im Python-Editor möglich, wodurch Programmteile aus einem Programm in ein anderes Programm problemlos kopiert werden können.

Ein Kopieren der Konsolenausgabe ist zum Zeitpunkt der Drucklegung nur in der Programmierumgebung für den LEGO Spike Prime möglich.

3.1.6 Kurze Einführung in die Objektorientierung

In diesem Abschnitt wird eine kurze Einführung zur objektorientierten Programmierung gegeben. Dabei wird auf einige wenige der wichtigsten Begriffe eingegangen. Es sollen hier nur die wichtigsten Grundlagen vermittelt werden, die für die direkte Programmierung des LEGO Mindstorms bzw. LEGO Spike Prime notwendig sind. Es ist nicht das Ziel, eine vollständige Vermittlung der Prinzipien der Objektorientierung zu versuchen, da hierfür ganze Bücher gefüllt werden.

Bei der prozeduralen Programmierung, wie sie zum Beispiel bei Basic verwendet wird, wird das Programm inhaltlich durch Prozeduren (sprich Unterprogramme) strukturiert, die nacheinander gerufen werden können.

Im Gegensatz dazu geht die Objektorientierung davon aus, dass ein Gesamtsystem sich durch Objekte beschreiben lässt, die sich durch Eigenschaften und Aktivitäten/Funktionen auszeichnen. Durch die Verknüpfung von Objekten wird dabei ein Gesamtsystem aufgebaut.

Beispiel

Ein sehr beliebtes Beispiel ist dabei ein Auto. Ein Auto besteht aus verschiedenen Teilen, die Eigenschaften haben und durch Aktivitäten das Gesamtsystem beeinflussen. Das Auto hat dabei einen Motor und mehrere Räder. Ein Rad kann sich drehen und wird durch die Aktivität des Motors beeinflusst. Die Türen haben die Möglichkeit, sich zu öffnen, und haben als sichtbare Eigenschaften eine Farbe. Diese wenigen Beispiele sollen verdeutlichen, dass sich im Grunde alle Systeme als Verbindungen von Objekten darstellen lassen, die sich gegenseitig beeinflussen beziehungsweise sich gegenseitig bedingen.

Die Objektorientierung erfordert somit auch eine andere Art der Herangehensweise an die Erstellung eines Programms im Vergleich zu einer prozeduralen Programmiersprache.

In Python ist auch ein prozeduraler Programmierstil möglich. Bei der Umsetzung mit Python wird in diesem Buch sogar öfter explizit dieser Stil angewendet, um die Vergleichbarkeit der Lösungen in den beiden Programmiersprachen besser zu verdeutlichen. Erst in den letzten Kapiteln wird stärker auf die Objektorientierung eingegangen, dabei werden die vorgestellten Lösungen mit entsprechenden Diagrammen erläutert.

In der Objektorientierung sind einige wichtige Kernbegriffe vorhanden, die die Grundlagen bestimmen.

- Klasse

  Eine Klasse stellt die allgemeine Definition eines Objekts dar und beinhaltet die Eigenschaftsbeschreibungen und Aktivitäten in verallgemeinerter Form, ohne dass diesen konkrete Werte zugewiesen sein müssen. Diese sind als Programmcode vorhanden. Eine Klasse kann nicht direkt ausgeführt werden.

  Ein Auto stellt dabei eine Beschreibung und Definition dar und kann sowohl Eigenschaften (zum Beispiel Farbe) als auch Aktivitäten (zum Beispiel Starten des Motors) haben. Eine Klasse sollte in Python mit einem Großbuchstaben (»UpperCaseCamelCase«) beginnen.

- Objekt

  Ein Objekt ist eine konkrete Instanz einer Klasse und repräsentiert eine konkrete Ausprägung. Ein Objekt wird während der Ausführung des Programms benötigt. Dabei erhalten Eigenschaften konkrete Werte.

  Im Fall des Autos stellt dies ein fertig produziertes, fahrbereites, konkretes Fahrzeug in der Farbe Rot dar.

- Kapselung

  Ein Grundkonzept der Objektorientierung ist die Kapselung von Daten und Eigenschaften. Ein Objekt kapselt dabei den vollständigen Zugriff auf dessen Inhalt und erlaubt nur auf die Inhalte Zugriff, die von außen erreichbar sein sollen.

- Methode
  Eine Methode ist ein von außen erreichbarer Kommunikationspunkt, der entweder Zugriff auf Daten oder die Ausführung von Programmlogik auf Basis eines konkreten Objekts (= Instanz einer Klasse) erlaubt. Damit wird eine Aktivität oder Funktion auf Basis eines Objekts gerufen.
  Der Name einer Methode sollte in Python nur aus Kleinbuchstaben bestehen. Mehrere Wörter in einer Methode sollten dann per Unterstrich getrennt werden.

- Vererbung
  Die Objektorientierung definiert eine Vererbungshierarchie. Dabei können konkretere Klassen existieren, die eine definierte Klasse genauer beschreibt. Ein Auto ist beispielsweise ein Pkw. Dies bedeutet, dass eine Klasse Pkw die Oberklasse eines Autos darstellt. Ein Pkw kann bereits Eigenschaften haben (zum Beispiel eine Geschwindigkeit), kennt aber noch keine Definition bezüglich der Reifen, die erst durch das Auto dazukommen.

- Pakete und Module
  Pakete und Module werden in Python verwendet, um Klassen in einer Struktur abzulegen.

3.1.7 Weiterführende Informationen zu Python

In Python gibt es eine Vielzahl von Schlüsselwörtern beziehungsweise Ausdrücken, die unveränderlich feststehen. Diese können nicht für andere Zwecke (zum Beispiel als Variablennamen) verwendet werden, sondern dienen einem bestimmten Zweck und bilden damit die Grundlage der Programmierung. Eine Auflistung wird im Anhang 8.2 zur Verfügung gestellt.

Bibliotheken

Python verfügt neben den Schlüsselwörtern über eine sehr große Anzahl an Standardbibliotheken mit Zusatzfunktionen, die für die Programmierung genutzt werden können. Diese Bibliotheken sind entweder Bestandteil der Python-Umsetzung und lassen sich direkt verwenden (zum Beispiel Klassen für den Zugriff auf Dateien) oder können als Zusatzbibliotheken hinzugenommen werden (zum Beispiel Klassen für ein erweitertes Logging). Für viele Bibliotheken existieren meist ausführliche Dokumentationen. Im Normalfall ist diese Dokumentation auf Englisch. Eine Übersetzung in andere Sprachen existiert häufig nicht, die Inhalte werden auch eher über Bücher oder Tutorials vermittelt.

Programmierbefehle ausführen

Ein Befehl in Python entspricht einem Methodenaufruf oder einer mathematischen Operation. Eine Zeile stellt dabei einen Befehl dar.

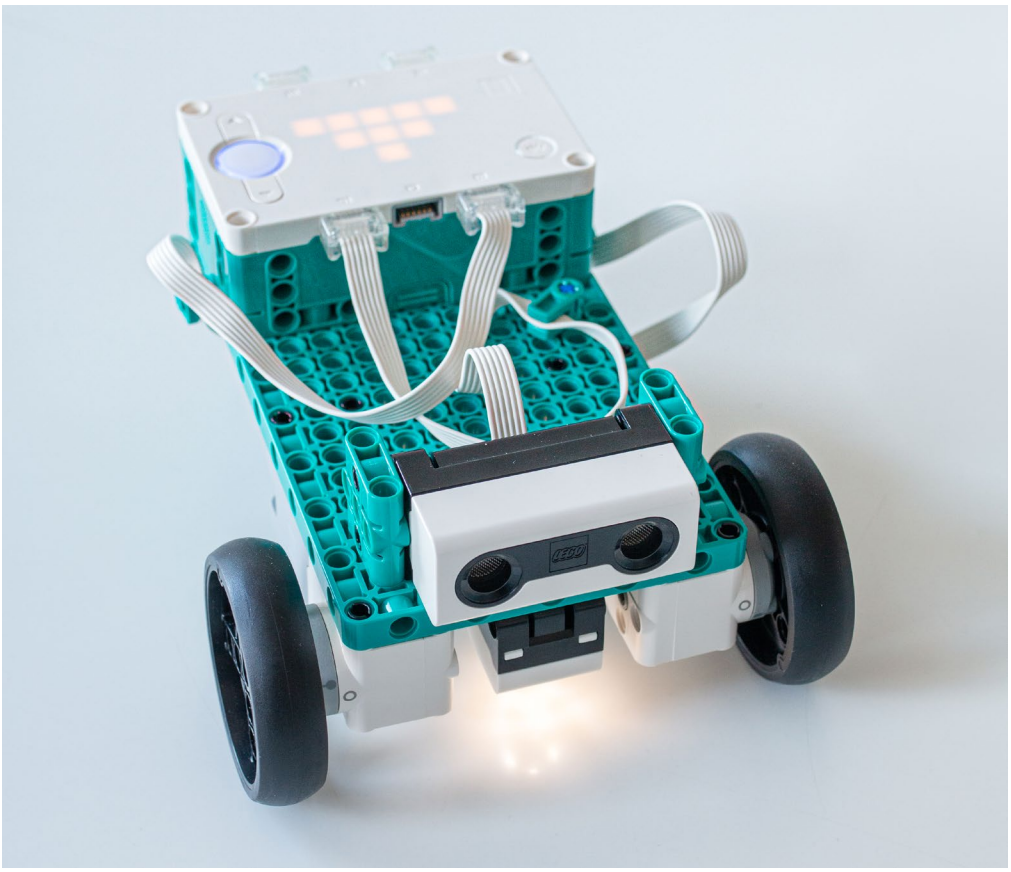
Bei Python ist hierbei wichtig, dass nicht alle Parameter einer Methode übergeben werden müssen (im Gegensatz zu Java). Zusätzlich kann ein Parameter mit seinem Namen übergeben werden, sodass z. B. nur der erste und dritte Parameter übergeben werden muss. Bei einer Übergabe ohne

3.3 Bau des Beispielroboters

Der Beispielroboter ist ein sehr einfaches Modell, das einen fahrenden Roboter mit ein oder zwei Farbsensoren und einem Ultraschallsensor kombiniert. Dieser Roboter besteht aus etwas mehr als 50 Teilen, ist in weniger als einer Viertelstunde aufgebaut und kann auch gut im Unterricht der Anleitung folgend gebaut werden. Solange nur ein Farbsensor verwendet wird, reicht die Ausstattung von LEGO Spike Prime oder LEGO Mindstorms 51515 vollständig aus.

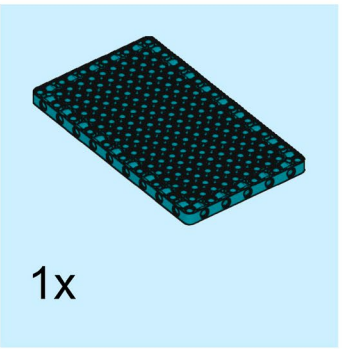
Dieser Roboter kann für typische Beispielszenarien wie Fahren, Hinderniserkennung, Ausweichen, Sound- und Displayanzeige genutzt werden und ist auch für erweiterte Problemlösungen wie die Linienverfolgung geeignet. Für die Teilnahme an Wettbewerben ist dieses Basismodell nur bedingt geeignet.

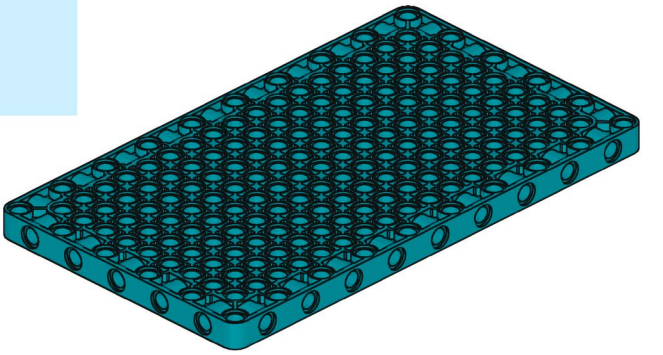
Eine Variante der folgenden Bauanleitung des Beispielroboters liegt im LDraw-Format (.ldr) vor und kann so direkt mit entsprechenden LEGO-CAD-Programmen (z. B. LDCad, MLCad, LDraw-Viewer) geladen. Diese Dateien stehen auf *dpunkt.de/mindspike* zum Download bereit.



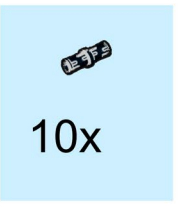
Die folgende Bauanleitung des Robotermodells zeigt die Teile in der Farbgebung der Mindstorms-Variante. Im Set Spike Prime 45678 sind die Teile – mit Ausnahme der Kugel im Heck – ebenfalls enthalten, nur in anderer Farbgebung. Die Verkabelung ist in der Anleitung nicht angezeigt, wird aber mit den anschließenden Bildern erläutert.

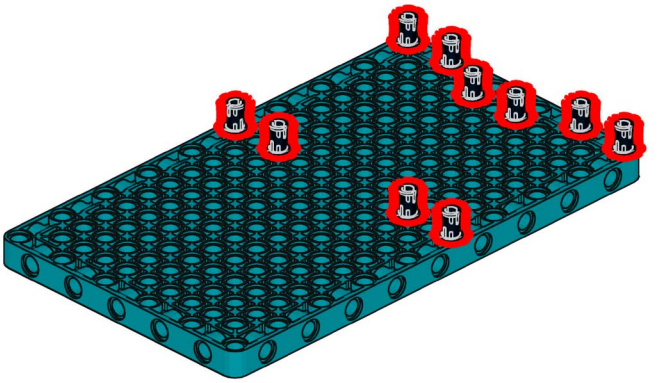
1



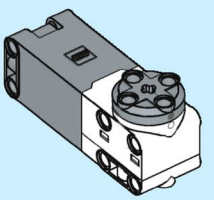


2

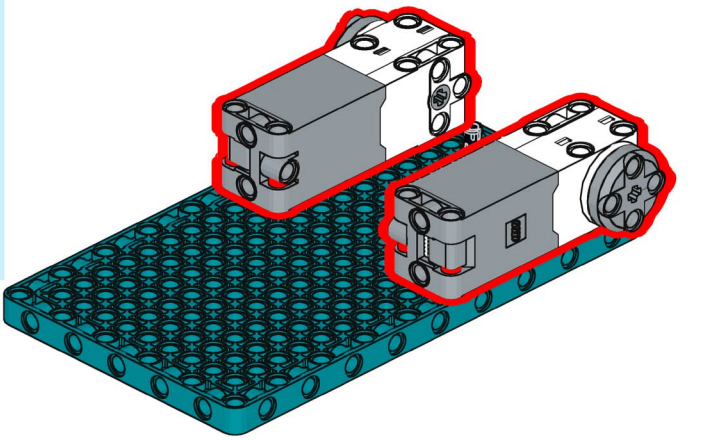




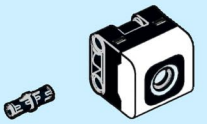
3



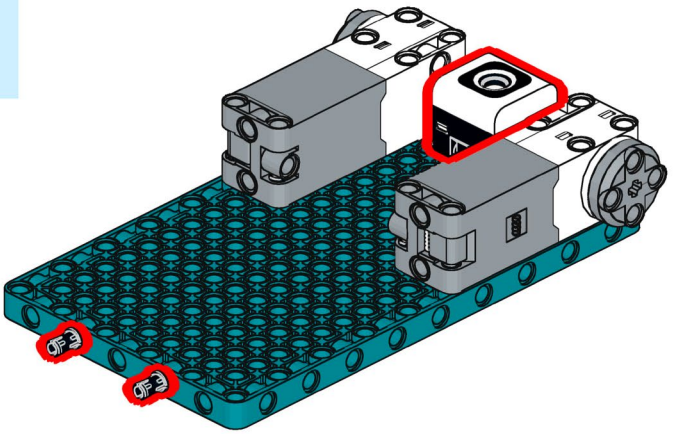
2x



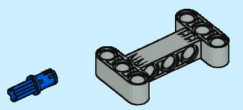
4



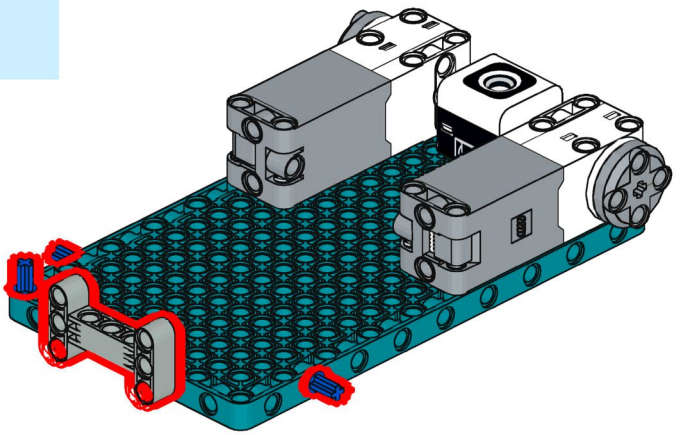
2x 1x



5

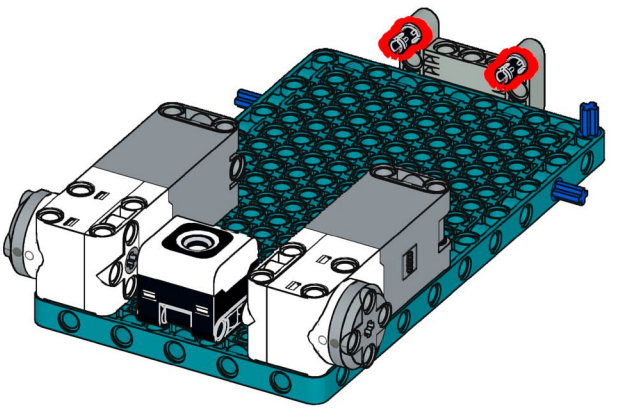


3x 1x



6

2x



```
wait_for_seconds(3)
hub.light_matrix.show_image('SURPRISED')
wait_for_seconds(3)
hub.light_matrix.show_image('SKULL')
wait_for_seconds(3)
```

4.3 Schleifen, Schleifenabbruch und Unterbrechung

Schleifen sind ein wichtiges Konstrukt in der Programmierung, um Programmschritte mehrmals auszuführen. Somit lässt sich eine Schleife folgendermaßen beschreiben:

Eine Schleife ...

- wiederholt die Aktionen in ihrem »Bauch« (Programmsequenzen)
- hat eine Abbruchbedingung
  - Abbruchbedingung führt bei erfolgreicher Prüfung zum Beenden der Schleife
  - Beispiele für mögliche Abbruchbedingungen: unendlich, Anzahl an Durchläufen, Sensorwerte
- kann eine weitere Schleife im »Bauch« beinhalten (Schachtelung von Schleifen)
- wird benötigt, um Aktionen (Programmsequenzen) mehrfach auszuführen, ohne diese im Programm kopieren zu müssen

Eine Schleife endet typischerweise mit einer Abbruchbedingung. In der Programmierung sind Schleifen mit vorgestellter oder nachgestellter Abbruchbedingung möglich, sodass entweder die zu wiederholende Programmsequenz mindestens einmal (bei nachgestellter Abbruchbedingung) oder gegebenenfalls gar nicht ausgeführt wird (bei vorgestellter Abbruchbedingung). Eine Abbruchbedingung wird immer erst geprüft, wenn das Programm den Prüfschritt erreicht. Somit kann man nicht davon ausgehen, dass eine Schleife sofort bei Erreichen des Abbruchereignisses beendet wird.

Es gibt auch die Möglichkeit, eine Schleife an jeder beliebigen Stelle abzubrechen. Dies ist allerdings von der jeweiligen Programmiersprache abhängig, da ein solcher Abbruch nicht in jeder Sprache ohne Probleme beziehungsweise ohne tiefere Kenntnis von nebenläufiger Programmierung (zum Beispiel Thread) möglich ist.

Diese verschiedenen Möglichkeiten und Einsatzgebiete für Schleifen werden bei der Lösung der folgenden Aufgaben deutlich:

ID	Aufgabenstellung
A431	Roboter fährt ein Viereck (Umsetzung mit einer Schleife) mit circa 20 cm Seitenlänge.
A432	Roboter fünfmal ein Stück (20 Grad Motordrehung) vorwärtsfahren und nach jeder Fahrsequenz 1 Sekunde warten lassen.
A433	Erweiterung des zweiten Programms, sodass nach dem fünften Mal ein Ton ausgegeben wird und dies alles solange wiederholt wird, bis der Ultraschallsensor eine Entfernung kleiner als 30 cm gemessen hat.
A434	Darstellung einer Animation von Punkten auf der Lichtmatrix, indem die Eckpunkte zum Zentrum animiert werden. Die Animation soll 5-mal durchlaufen werden.

Für die erste Aufgabe A431 kommen dabei das erste Mal die Überlegungen zum Tragen, wie der Roboter eine feste Strecke und eine konkrete Drehung vollziehen kann. Dies ist mit einfacher Physik oder mit erweiterten Sensoren zu beantworten. Deshalb gehen wir zunächst auf die Physik ein.

Die Strecke, die ein fahrendes Objekt zurücklegt, hängt von der Drehung und der Größe der Räder ab. Die gefahrene Strecke lässt sich einfach berechnen, indem die Anzahl der Drehungen mit dem Radumfang multipliziert wird. Der Radumfang berechnet sich aus Durchmesser × Kreiszahl PI (ungefähr 3,14). LEGO druckt die Reifengröße auf die Reifen auf, sodass diese wie beim Fahrrad oder Auto abgelesen werden können. Die Angabe ist dabei in Durchmesser × Breite in Millimeter angegeben. Das im Mindstorms/Spike-Standardkasten enthaltene Rad hat einen Durchmesser von 57 mm. Um somit eine Strecke von 20 cm zurückzulegen, sind 20 cm/(5,7 cm × π) Umdrehungen notwendig. Dies sind circa 1,1169 Umdrehungen oder 402 Grad. Grundsätzlich sind diese Überlegungen für weitere Varianten sinnvoll und notwendig, allerdings bietet der Mindstorms und Spike bereits die Möglichkeit, eine feste Fahrstrecke in Zentimeter oder Inch zurückzulegen. Dies funktioniert jedoch nur, wenn keine Übersetzung mittels Zahnräder verbaut ist.

Im Unterricht wird eher das Prinzip »Trial and Error« (sprich: Versuch und Irrtum) angewendet werden, da dies in der Praxis oftmals die detaillierteren Ergebnisse liefert. Denn die tatsächlich gefahrene Strecke hängt nicht nur von den berechneten Werten ab, sondern auch von der Reibung, dem Untergrund, dem Batterieladestand und mit wie viel Leistung die Motoren angesteuert werden. Die LEGO-Motoren sind zwar ziemlich genau, aber je schneller die Motoren drehen, umso ungenauer werden auch die Messwerte und die Ansteuerung. Somit ist eine große Geschwindigkeit bei genauen Aufgaben meist der falsche Ansatz.

Bei Drehungen eines Roboters können ähnliche Überlegungen angestellt werden, die dabei von der Physik des Roboters abhängig sind und auch davon, wie der Roboter für eine Drehung angesteuert wird. Der Beispielroboter kann auf der Stelle wie ein Panzer drehen. In dem Fall laufen die beiden Motoren gegenläufig (das heißt, das eine Rad dreht vorwärts, das andere Rad dreht rückwärts), oder aber der Roboter bleibt mit einem Rad stillstehen und dreht nur durch eine Radbewegung. Dabei ändert sich der Mittelpunkt der Drehung und damit auch die notwendige Drehbewegung je Rad. Dies wird in den folgenden Diagrammen dargestellt.

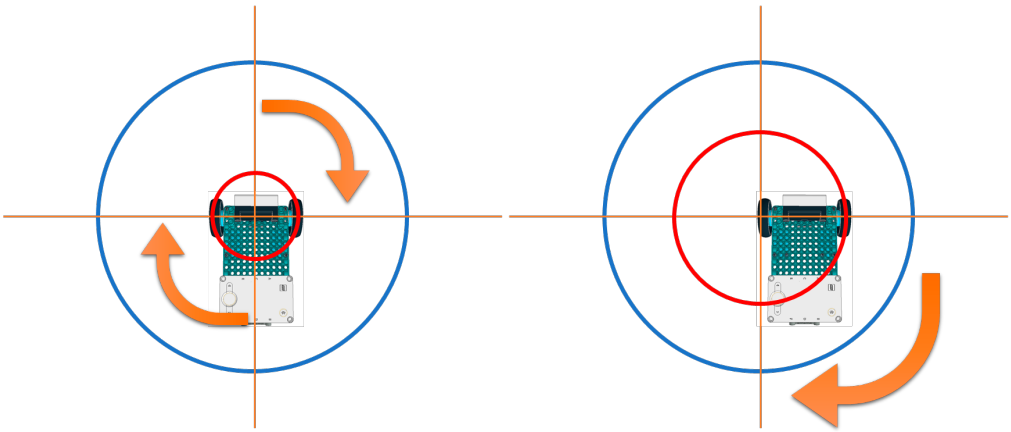


Abb. 4–10 // Unterschiedliche Drehalgorithmen bei fahrenden Robotermodellen

Der rote Kreis verdeutlicht die Strecke, die das jeweilige Rad zurücklegen muss, um eine entsprechende Drehung zu vollziehen. Bei einer Panzerdrehung ist somit eine geringere Drehbewegung eines Rads notwendig, da sich die zurückgelegte Strecke je Rad addiert. Die Strecke pro Rad berechnet sich aus dem Durchmesser des Drehkreises und der gewünschten Drehung – somit Durchmesser × π × ¼ für eine 90-Grad-Drehung.

Der Abstand der Räder im Beispielroboter, die den Durchmesser des Drehkreises bei der Panzerdrehung sowie den Radius des Drehkreises bei der Drehung mit einem Rad darstellen, beträgt circa 11 cm.

Aus der Überlegung der Streckenberechnung zuvor ergeben sich somit die folgenden Werte für die Berechnung einer 90-Grad-Kurve:

	Strecke für die Drehung	Motordrehung
Panzerdrehung	11 cm × π × ¼ = 8,64 cm	8,64 cm/(5,7 cm × π) = 0,48 = 174 Grad
Einraddrehung	2×11 cm × π × ¼ = 17,28 cm	17,28 cm/(5,7 cm × π) = 0,96 = 347 Grad

Mit diesem Ansatz sind die Strecken und Drehungen immer noch mit festen Werten programmiert. Umgebungsbedingte Rahmenbedingungen wie ein rutschiger Untergrund können immer dazu führen, dass die Strecke oder die Drehung ungenau erfolgt.

Um dies besser zu gestalten, ist ein alternativer Ansatz über weitere Sensoren ebenfalls möglich.

Ein Ultraschallsensor kann die Entfernung von einer Wand messen. Wenn somit immer eine Wand für die Entfernungsmessung vorhanden ist, kann der Unterschied zur letzten Messung für eine Fahrt von 20 cm genutzt werden.

Für die Drehung bietet sich der Kreiselsensor (Gyrosensor) im Hub an. Dieser misst die Winkelgeschwindigkeit und kann diese in einen Drehwinkel umrechnen. Auch hier ist die Fahrgeschwindigkeit von entscheidender Bedeutung für die Genauigkeit der Drehung. Dieser wird in späteren

Kapiteln intensiver genutzt, um die Drehung des Roboters unabhängig von seiner Bauart korrekt durchzuführen. Um die Komplexität in diesem Abschnitt geringer zu halten, wird erstmal darauf verzichtet.

4.3.1 Umsetzung mit Textblöcken

Eine Schleife bei den Textblöcken wird durch einen der folgenden Klammerblöcke repräsentiert. Diese befinden sich bei den orangen Elementen, die die Ablaufsteuerung im Programm darstellen.

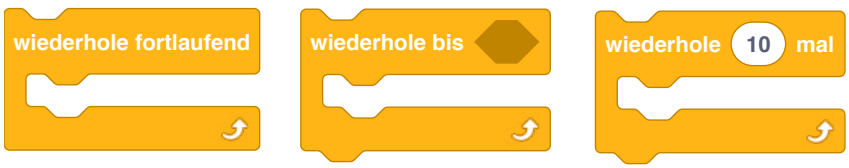


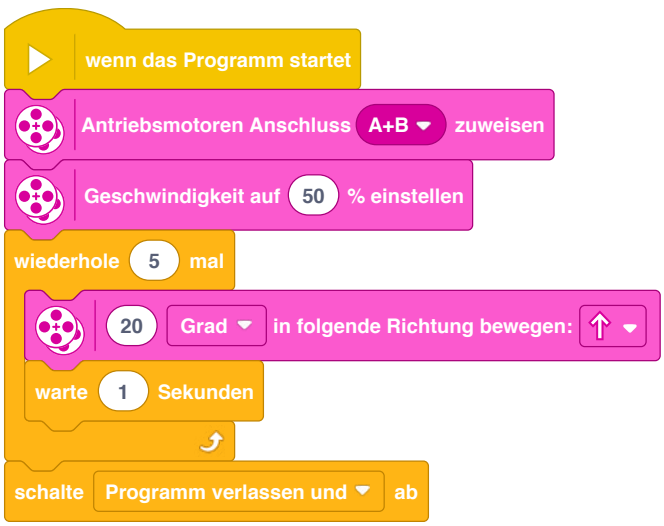
Abb. 4–11 // Unterschiedliche Blöcke für Schleifen als Klammerblöcke

Textblöcke unterstützen nur Schleifen mit vorgelagerter Abbruchbedingung. Somit wird die Programmsequenz innerhalb der Schleife bei Erreichen der Abbruchbedingung nicht mehr ausgeführt und die Abbruchbedingung immer zu Beginn eines Schleifendurchgangs geprüft. Wenn die Prüfung der Abbruchbedingung wahr (true) ist, wird die Schleife beendet. Solange sie falsch (false) ist, wird die Schleife erneut durchlaufen (»Führe die Schleife so lange aus, bis das Ereignis eintritt!«). Dies entspricht einer sogenannten Repeat-Until-Schleife. Dies ist sehr wichtig, da die Schleifenprüfung im Gegensatz zu der in diesem Buch diskutierten textuellen Programmierung umgedreht verstanden wird.

Die Abbruchbedingung ist bei den Textblöcken grundsätzlich beliebig möglich. Eine vorgefertigte Abbruchbedingung stellt einen Zähler dar und wird durch den Textblock »wiederhole x-mal« repräsentiert. Weiterhin gibt es eine sogenannte Endlosschleife, welche die enthaltene Programmsequenz unendlich oft ausführt (»wiederhole fortlaufend«). Ein wichtiger Aspekt ist die Abbruchbedingung aufgrund eines logischen Werts (wahr oder falsch). Damit lässt sich die Abbruchbedingung innerhalb der Schleife oder bei der Prüfung aufgrund einer Logik berechnen und als Abbruchbedingung nutzen. Dafür kann z. B. ein Vergleich mit einem Sensorwert genutzt werden, wie er in Aufgabe A433 notwendig wird.

Die Aufgabe A431 ist dabei über eine Schleife, die 4-mal durchlaufen wird, zu lösen und den Roboter immer ein Stück vorwärtsfahren und die 90-Grad-Drehung durchführen lässt. Hierbei müssen die ersten Fragen einer sinnvollen Drehung um 90 Grad beachtet werden, die in dem Fall durch einfache Versuche ermittelt werden können. Das folgende Programm liefert dabei ein passendes Ergebnis für den Beispielroboter. Die Abbruchbedingung ist dabei ein Zähler und wird nach dem vierten Durchlauf erreicht – die Schleife wird damit beendet.

Für die Aufgabe A432 wird die Vorwärtsbewegung des Roboters auf Grad umgestellt, wobei 20 Grad Motordrehung genutzt werden. Anschließend wird mit dem Warteblock eine Sekunde gewartet und dies mithilfe der Schleife 5-mal durchgeführt.



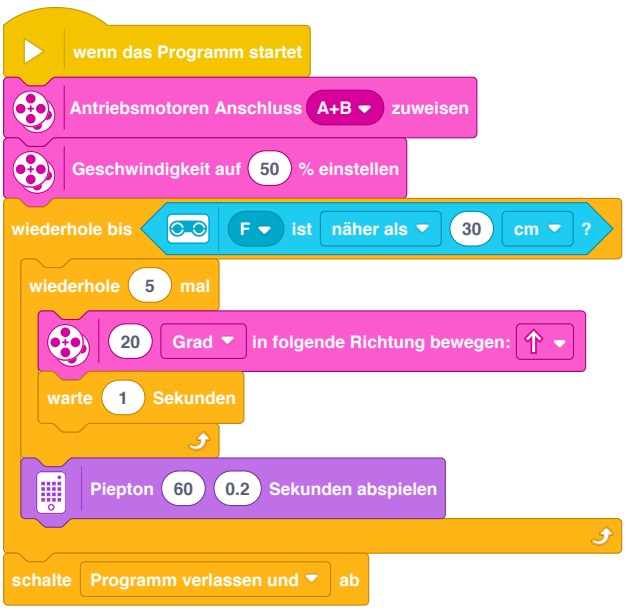
Die Aufgabe A433 ist etwas komplexer und erfordert verschachtelte Schleifen. Dabei gibt es eine innere Schleife, die das Programm aus der zweiten Aufgabe darstellt. Die äußere Schleife beinhaltet den Aufruf der inneren Schleife und zusätzlich den Tonblock zur Ausgabe des Tons. Die Abbruchbedingung der äußeren Schleife wird dabei auf die Entfernungsmessung des Ultraschallsensors umgestellt. Dabei soll die Schleife so lange laufen, bis die Entfernungsmessung weniger als 30 cm misst. Somit muss die Kontrolle auf kleiner 30 cm gestellt werden. Für die Aufgabe A414 wurde bereits die Abbruchbedingung für eine Farbe genutzt. Im türkisen Bereich findet sich auch

eine Ereignisabfrage für den Ultraschallsensor. Dabei lässt sich neben dem Port zusätzlich die Bedingung (»näher als«, »weiter als«, »genau bei«) sowie die Maßeinheit (»cm«, »%«, »Zoll«) wählen. Die Prozentangabe bezieht sich damit auf die maximale Entfernungsmessung von 200 cm.

Abb. 4–12 // Auswahl des Vergleichsoperators bei Textblöcken

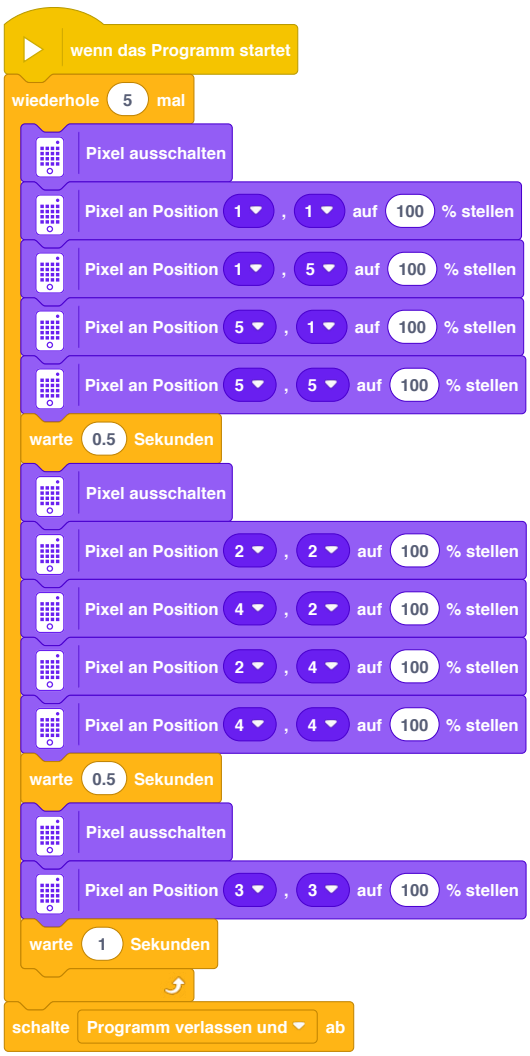
Für die Bedingung ist somit folgende Konfiguration notwendig.

Aufbauend auf diesen Angaben sieht eine mögliche Lösung der Aufgabe A433 folgendermaßen aus:



Die Abbruchbedingung wird immer beim Schleifenbeginn geprüft. Im Beispiel der dritten Aufgabe bedeutet dies, dass die Entfernungsmessung erneut erst nach Abspielen des Klangs stattfindet. Der Roboter bleibt nicht stehen, wenn er innerhalb der inneren Schleife (20-Grad-Schritte) die 30 cm Entfernung zu einer Wand unterschreitet. Dies ist eine wichtige Erkenntnis, da für eine permanente Kontrolle von Ereignissen andere Techniken notwendig sind (zum Beispiel parallele Ausführung von Abfragen oder Abfragen zu dedizierten Zeitpunkten).

Die Aufgabe A434 baut auf der Aufgabe A423 auf und zeigt eine kleine Animation auf der Lichtmatrix mithilfe einer Schleife. Dabei wird nach jedem Aktivieren bestimmter Pixel eine Wartezeit eingeplant, die Lichtmatrix gelöscht und anschließend der nächste Animationsschritt gezeigt. Dabei laufen die Punkte von außen nach innen und simulieren so einen Animationsablauf.



Eine Schleife kann bei den Textblöcken theoretisch auch abgebrochen werden. Dafür steht der Textblock zum Abschalten eines Stapels im orangen Bereich zur Verfügung. Dadurch kann eine Endlosschleife sofort bei Erreichen dieses Textblocks beendet werden.

Allerdings beendet dies den gesamten Stapel inklusive aller darin befindlichen Programmsequenzen sofort. Somit muss, vor allem bei parallelen Programmierpfaden, sehr darauf geachtet werden, wann ein Stapelabbruch erfolgt, da dies das Programm auch in einen undefinierten Zustand bringen kann, wenn die Blöcke in der Schleife nicht korrekt zu Ende gebracht wurden.

Ein kontrollierter Abbruch einer konkreten Schleife und Fortsetzen des Stapels nach der Schleife wird mit Textblöcken nicht unterstützt. Dies müsste mit einem parallel gestarteten Stapel (siehe Kapitel 4.7 und 5.2) umgesetzt werden.

Im grünen Bereich der Palette finden sich weitere Wahrheits- und Wertblöcke, die Bedingungen berechnen können, welche dann in Schleifen oder Schaltern genutzt werden können. Dabei geht es konkret um Vergleiche (größer, kleiner, gleich) von Werten, Bereichsprüfungen (liegt ein Wert zwischen x und y) sowie logische Operationen (UND-, ODER-Verknüpfungen). Diese werden im Kapitel 4.7 detaillierter betrachtet.

4.3.2 Umsetzung mit Python

In diesem Abschnitt wird das Schleifenkonstrukt von Python erläutert und verschiedene Einsatzmöglichkeiten vertieft. Eine Schleife stellt dabei eine grundlegende Anweisung in Python dar. Durch die Eigenschaft von Python, auch in sich verschachtelte Aufrufe zu ermöglichen, bietet Python eine wesentlich größere Bandbreite für Schleifenkonstrukte an, als es mit Textblöcken möglich ist. Im Rahmen dieses Abschnitts werden dabei die wichtigsten angesprochen, um auch die Aufgaben sinnvoll zu lösen.

Es werden in Python zwei Schleifenanweisungen unterschieden:

Python-Anweisungen	Kurzerläuterung
for ... in ... else	Schleife, welche über einen Inhalt iteriert
while ... else	Schleife mit einer generischen Abbruchbedingung

Die for-Schleife in Python unterscheidet sich von anderen textuellen Programmiersprachen, weil die for-Schleife keine Schleife für eine zählbare Wiederholung einer Programmsequenz darstellt. Vielmehr wird mit der for-Schleife über einen Inhalt (z. B. ein Feld, ein Text o. ä.) iteriert und ist

eher mit Iteratoren aus anderen Programmiersprachen vergleichbar. Der Inhalt stellt dabei eine Sequenz von Daten dar. Die folgenden Beispiele sollen dies grundsätzlich verdeutlichen. Beim ersten Beispiel wird jeder Buchstabe des Worts »LEGO« einzeln ausgegeben. Im zweiten Beispiel wird zuerst der Text »R2D2« und anschließend »C3PO« ausgegeben:

```
for c in "LEGO":
    print(c)

robots = ["R2D2", "C3PO"]
for f in robots:
    print(f)
```

Für die vorliegenden Aufgaben ist die for-Schleife von Python nur bedingt einsetzbar.

In Python ist auch eine andere Schleifenart möglich – die sogenannte While-Schleife. Diese stellt die generische und universell einsetzbare Schleifenart in Python dar, bei der eine Abbruchbedingungen notwendig ist, um die Schleife zu beenden. Eine While-Schleife wird durchlaufen, solange eine vorgegebene Bedingung erfüllt ist. Dabei unterstützt Python nur die vorgestellte Prüfung der Abbruchbedingung. Dies bedeutet, dass die Prüfung erfolgt, bevor der Schleifeninhalt ausgeführt wird.

Ein Beispiel verdeutlicht der folgende Programmcode, der aufsteigend eine Zahl ausgibt und die Schleife so lange durchläuft, solange der Zähler kleiner als 5 ist:

```
i=1
while i<5:
    print(i)
    i=i+1
else:
    print("ist größer als 5")
```

Dabei werden das erste Mal auch Variablen eingesetzt, die erhöht werden müssen. Die Details zur Verwendung von Variablen werden in Kapitel 4.5 näher erläutert.

Darüber hinaus wird eine Besonderheit in der Python-Sprache aufgezeigt. Python unterstützt einen else-Block bei einer while-Schleife. Dieser Block wird ausgeführt, wenn die Bedingung nicht mehr erfüllt ist. Der else-Block wird allerdings auch ausgeführt, wenn der Schleifenrumpf nie ausgeführt wurde.

Eine Abbruchbedingung muss den Wert wahr (true) erfüllen. Somit kann darin auch beliebiger alternativer Programmcode ausgeführt werden. Dies erweitert die Einsatzmöglichkeiten der While-Schleife, um zum Beispiel auf ein bestimmtes Ereignis des Hubs oder eines Sensors zu warten:

```
while colorSensor.get_color() != 'red':
    # do something
```

Bei den Textblöcken wird die Schleife so lange wiederholt, bis eine Bedingung erfüllt ist (»wiederhole bis«). In Python wird die Schleife so lange wiederholt, solange die Bedingungsprüfung wahr (true) liefert. Somit sind hier die Abfragen genau umgedreht zu den Textblöcken.

Die ersten beiden Aufgaben lassen sich nach diesen Ausführungen ohne weitere notwendige Erklärungen lösen.

Die Lösung für Aufgabe A431 sieht folgendermaßen aus:

```
motor_pair = MotorPair('A', 'B')
motor_pair.set_default_speed(50)
i = 0
while i < 4:
    motor_pair.move_tank(20, 'cm')
    motor_pair.move_tank(10, 'cm', -50, 50)
    i = i + 1
```

Die Lösung für Aufgabe A432 kann folgendermaßen umgesetzt werden:

```
motor_pair = MotorPair('A', 'B')
motor_pair.set_default_speed(50)
i = 0
while i < 5:
    motor_pair.move_tank(20, 'degrees')
    wait_for_seconds(1)
    i = i + 1
```

Für die Aufgabe A433, bei der die Entfernung des Ultraschallsensors gemessen werden soll, kommt erneut die Klasse `DistanceSensor` zum Einsatz. Nach Initialisierung mit dem korrekten Port kann der Entfernungswert ausgelesen und als Abbruchbedingung in der While-Schleife genutzt werden. Hierbei ist zu beachten, dass die Abbruchbedingung im Vergleich zu den Textblöcken umgedreht werden muss. Bei den Textblöcken wird eine Schleife solange ausgeführt, bis ein Ereignis eintritt und damit die Schleife beendet wird. Bei einer While-Schleife wird die Schleife solange durchgeführt, wie die Abbruchbedingung den Wert wahr (true) liefert. In der Beispiellösung muss somit die Prüfung in der While-Schleife auf eine Entfernung größer 30 cm erfolgen, da die Schleife solange ausgeführt werden soll. Es kann hierbei zu einem Programmfehler kommen, wenn der Ultraschallsensor keinen gültigen Wert liefert. Eine sinnvolle Umgehung dieses Problems wird konkret in Abschnitt 4.7.2 vorgestellt.

```
hub = MSHub()

motor_pair = MotorPair('A', 'B')
motor_pair.set_default_speed(50)
```