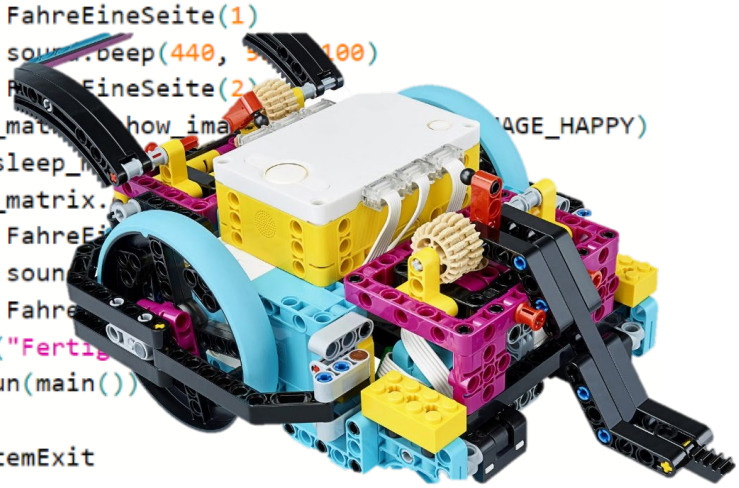
SWITZERLAND

PYTHON FÜR SPIKE PRIME

Grundlagen

```python
from hub import port, sound, light_matrix
import motor_pair, runloop, time

MP = motor_pair.PAIR_1
motor_pair.pair(MP, port.A, port.B)

async def FahreEineSeite(SEITENNUMMER):
    light_matrix.write(str(SEITENNUMMER))
    await motor_pair.move_for_degrees(MP, 360, 0)
    await motor_pair.move_for_degrees(MP, 210, 100)
    light_matrix.clear()

async def main():
    await FahreEineSeite(1)
    await sou beep(440, 5 100)
    await F EineSeite(2
    light_mat how_ima AGE_HAPPY)
    time.sleep_
    light_matrix.
    await FahreE
    await soun
    await Fahre
    print("Fertig
runloop.run(main())

raise SystemExit
```



Vera Hausherr
Version 1.0 / 13.09.2023

© 2021 World Robot Olympiad Schweiz
World Robot Olympiad Schweiz ist die offizielle Ausrichterin der World Robot Olympiad Wettbewerbe in der Schweiz.
World Robot Olympiad™ und WRO® sind eingetragene Markenzeichen der World Robot Olympiad Association.

Copyright

Dieses Lerndossier ist das geistige Eigentum der Autorin, Vera Hausherr, und der World Robot Olympiad Schweiz (WRO). Es wird kostenfrei an die Teilnehmenden von WRO-Workshops abgegeben und darf von diesen sowie ihren Team-Kolleg:innen oder Schüler:innen verwendet werden. Wir bitten alle Teilnehmenden, dieses geistige Eigentum zu respektieren und das Lerndossier nicht an weitere Personen weiterzugeben.

Dieses Werk darf nicht gegen Entgelt weitergegeben werden.

Markenzeichen

LEGO®, SPIKE™ Prime, LEGO® Education, SPIKE™ App und andere Begriffe, die mit LEGO zu tun haben, sind Markenzeichen. Um den Text leichter lesbar zu machen, werden die Markenzeichen-Symbole weggelassen.

Technischer Hinweis

Dieses Lerndossier wurde auf der Basis der Version 3 der SPIKE-App verfasst. Es kann kleine Abweichungen zwischen den Abbildungen in diesem Dossier und früheren (oder noch unveröffentlichten späteren) Versionen der SPIKE-App geben.

Lust auf mehr Robotik?

Dann mach mit am Wettbewerb der World Robot Olympiad! Alle Informationen findest du auf https://wro.swiss

Inhalt

Copyright 2
Inhalt 3
1. Grundlagen 4
1.1 Ein Projekt anlegen 4
1.2 Hallo Welt! 5
1.3 Kommentare in Python 7
1.4 Farben im Code 8
1.5 Einen Timer verwenden 8
2. Motoren 10
2.1 Ein einzelner Motor 10
2.2 Zwei Motoren, die nicht zusammengehören 11
2.3 Antriebsmotoren bei einem Fahr-Roboter 13
2.4 Fahren 14
2.4.1 Geradeaus fahren 14
2.4.2 Kurven und Drehungen 16
2.5 Eine bestimmte Strecke fahren 17
3. Sensoren 19
3.1 Der Farbsensor 19
3.1.1 Farben 19
3.1.2 Reflektiertes Licht 20
3.2 Der Abstandssensor 21
4. Schleifen und Bedingungen 23
4.1 Schleifen 23
4.1.1 Schleife mit einem Zähler 23
4.1.2 Endlosschleife 25
4.1.3 Schleife mit einer Stoppbedingung 25
4.2 Bedingungen 25
4.2.1 Einfache Bedingungen 25
4.2.2 Verschachtelte Bedingungen 25
5. Funktionen («MyBlocks») 25
4.2.3 Funktionen ohne Parameter 25
4.2.4 Funktionen mit Parameter 26
4.2.5 Funktionen mit Rückgabewert 26
Lösungen 30
Lösung zu Teil 2.2 Zwei Motoren 30
Lösung zu Teil 4.1.1. Schleifen 30

1. Grundlagen

1.1 Ein Projekt anlegen

Lernziele:

- ☐ Ich weiss, wie ich ein neues Projekt in Python anlege.

- Öffne die SPIKE App und gehe zum Bereich «SPIKE Prime».
- Klicke auf der Startseite auf **Neues Projekt** und lege dann ein neues Projekt an.
- Tipp: Stelle sicher, dass **Python** ausgewählt ist und klicke dann auf **Anlegen**.

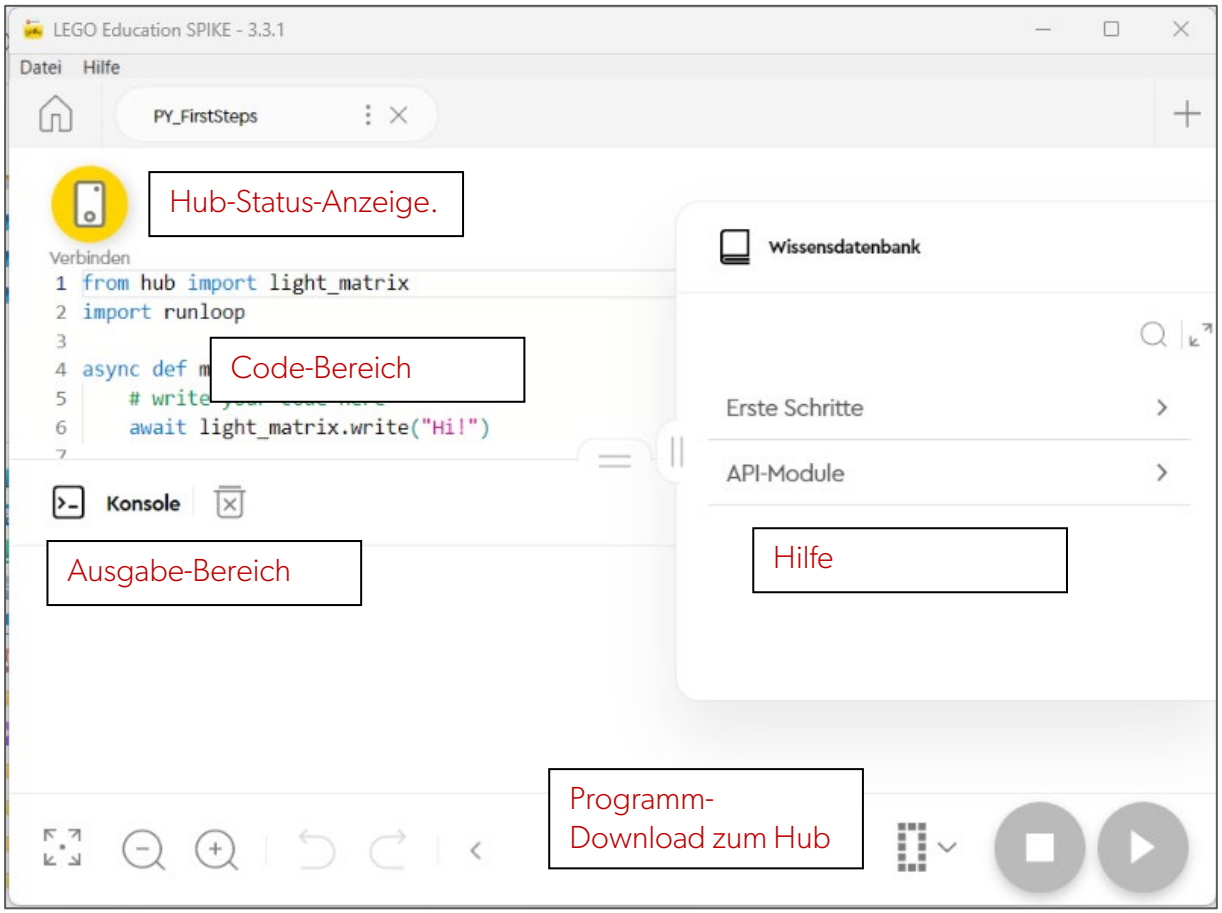
Tipp:

Gib dem Projekt einen Namen, der für dich Sinn macht, damit du es später leichter wieder findest.

Du kannst dir auch eine Regel angewöhnen: Fange Python-Projekte immer mit PY_ an, also z.B. PY_FirstSteps. Wenn du sie dann später suchst, findest du alle Python-Projekte am selben Platz.

Wenn du an einem grossen Projekt arbeitest, bietet es sich auch an, es immer wieder unter einem anderen Namen zu speichern, z.B. mit dem jeweiligen Datum, also PY_FirstSteps_05-09-23. So hast du dann noch ein Backup für den Notfall!

Du siehst jetzt diesen Bildschirm:



Den Hilfe-Bereich («Wissensdatenbank») sowie den Ausgabe-Bereich («Konsole») kannst du vergrössern oder verkleinern, um mehr Platz für den Code zu haben.

Wir werden diese Bereiche später erklären.

Aufgabe 1

- Lege ein Python-Projekt mit dem Namen PY_HalloWelt an!
- Lösche in diesem neuen Projekt alle Code-Zeilen, die automatisch erstellt werden. Damit werden wir weiter arbeiten.

1.2 Hallo Welt!

Lernziel

☐ Ich weiss, wie ein einfaches Programm aufgebaut ist.

Wir schreiben jetzt unser aller erstes Programm. Es ist eine gute Regel bei Programmierern, dass man, wenn man eine neue Programmiersprache ausprobieren will, als erstes ein so genanntes «Hallo Welt»-Programm schreibt, das genau das macht: Beim Start des Programmes wird auf der Lichtmatrix des Hubs der Satz «Hallo Welt» angezeigt. Das ist oft das einfachste mögliche Programm, und man kann daran schon ein bisschen sehen, wie die neue Programmiersprache funktioniert.

Lösche zunächst mal alles, was schon auf der Seite steht. Übrig bleibt jetzt nur noch eine leere Zeile, die schon nummeriert ist. Die Zeilen-Nummerierung wird automatisch erstellt und kann nicht gelöscht werden. Du wirst bei grösseren Programmen merken, dass sie sehr nützlich ist!

In die erste Zeile schreibst du jetzt diesen Text: (Achtung: Gross- und Kleinschreibung müssen genau so sein!)

```
from hub import light_matrix
```

Damit sagst du dem Programm, dass er die Programmbausteine zur Lichtmatrix verwenden soll. Manchmal brauchst du auch noch andere Bausteine, die du importieren musst.

Tipp:

Am Anfang scheint das mit dem «Import» vielleicht etwas verwirrend. Bei Scratch hat man schliesslich auch alle Bausteine gleich verfügbar!

Aber Python ist ein sehr mächtiges System. Stelle es dir wie eine riesige Bibliothek vor, in der man Bücher zu ganz vielen verschiedenen Themen findet: Bücher zur Geographie, zur Geschichte, zur Biologie. Wenn du jetzt eine Präsentation über Ameisen machen musst, brauchst du nicht die ganze Bibliothek. Sondern nur Bücher über Ameisen, die du im Bereich «Biologie» findest.

In unserem Code ist **`hub`** der Bereich, und **`light_matrix`** das Buch.

Aus Scratch kennst du übrigens ein ähnliches Prinzip: Du suchst aus, in welchem Farb-Bereich dein Textblock liegt, und schaust dir dann das Symbol am Anfang des Textblocks an. Für alles, was du mit der Lichtmatrix machst, brauchst du die violetten Blöcke mit dem Hub-Symbol:

Wenn du also ein Programm schreibst, musst du immer überlegen, in welchem Bereich du die entsprechenden Bausteine findest. Die Lichtmatrix ist auf dem Hub, als importierst du aus diesem Bereich.

Tipp:

In der Wissensdatenbank wird sowohl für den Bereich als auch für den Programmbaustein das Wort «**Modul**» verwendet.

In diesem Lerndossier bleiben wir bei den Begriffen **Bereich** und (Programm-) **Baustein**, damit es nicht zu Verwechslungen kommt. Das Wort **Modul** wird als Oberbegriff für beides verwendet.

Noch kann unser Programm gar nichts tun, Wir haben ja schliesslich erst gesagt, welcher Baustein verwendet werden soll, aber diesem Baustein noch keine Befehle zugeordnet.

Das machen wir mit einer weiteren Textzeile: (achte darauf, es exakt so zu schreiben, ohne Leerschlag zwischen den Wörtern und Satzzeichen!)

```
3 light_matrix.write('Hallo Welt!')
```

Du siehst, dies wird in Zeile 3 geschrieben. Zeile 2 bleibt leer. So trennen wir den Kopfbereich des Codes (mit den **`import`**-Befehlen) vom eigentlichen Code.

Tipp:

Code wird schnell lang und damit unübersichtlich. Gewöhne dir daher an, dass du ihn immer wieder durch sinnvolle Leerzeilen unterteilst, um dich leichter zurecht zu finden!

Schauen wir uns Code-Zeile 3 an:

Wir rufen zunächst den Baustein **`light_matrix`** auf, den wir vorher importiert haben. Dann sagen wir, was dieser Baustein tun soll: **`write()`** (=schreiben). In die Klammer kommt jetzt, was geschrieben werden soll. Damit das Programm weiss, dass es sich um Text handelt, machen wir einfache Anführungszeichen drumherum. Achtung: mit doppelten Anführungszeichen funktioniert es nicht!

Merke:

Wir haben eine Grundstruktur, die wir in Python immer wieder brauchen:

`baustein.befehl(parameter)`
Ein Parameter ist ein veränderbarer Wert. Du könntest statt **`'Hallo Welt'`** auch **`'Ciao Mondo'`** schreiben, oder **`'Hello World'`** oder irgendeinen anderen Text. Das ändert nichts am Programm an sich, sondern nur am Ergebnis.

Wenn du schon mit Scratch gearbeitet hast, kennst du Parameter gut: Sie werden immer in die ovalen Felder in den Textblöcken geschrieben.

Aufgabe 2

- Ändere den Code so, dass er dich begrüsst!

1.3 Kommentare in Python

Lernziel

☐ Ich kann meinen Code kommentieren und weiss, warum das nützlich ist.

Wenn dein Programm nur eine einzige Sache machen kann, und diese Sache bereits aus dem Projektnamen erkennbar ist, dann musst du deinen Code nicht unbedingt kommentieren. Aber oft werden Programme länger und sie können unterschiedliche Sachen machen.

Dann ist es sinnvoll, immer wieder zu schreiben, was das eigentlich bedeutet, was du gerade gecodet hast, und warum du es so programmiert hast.

Kommentare fangen immer mit dem **#**-Symbol an und gehen bis zum Ende dieser Zeile. Wenn du einen längeren Kommentar schreiben willst, muss jede Zeile mit dem **#** anfangen.

```
light_matrix.write('Hello, World!')
# So schreibt der Hub "Hallo Welt"
# Wenn der Kommentar sehr lang ist,
# teile ihn auf mehrere Zeilen auf
# und fange jede Zeile mit einem # an.
```

1.4 Farben im Code

- ☐ Ich weiss, was die wichtigsten Farben im Python-Code sind und kann das für mich nutzbar werden lassen.

Dir ist sicher schon aufgefallen, dass der Text in deinem Code automatisch in verschiedenen Farben formatiert wird. Das zeigt, dass unsere Programmierumgebung ziemlich schlau ist und mitdenkt.

Das sind die wichtigsten Farben.

`Blau` Schlüsselwörter, die unser Programm gliedern. Du kennst bereits **`from`** und **`import`**, aber auch **`if`**, **`else`** (wenn, dann, sonst) werden automatisch blau dargestellt, ebenso wie **`True`** und **`False`** (Wahr und Falsch; achte darauf, dass diese beiden grossgeschrieben werden müssen!)

`Pink` sind Text-Parameter, die du als festen Text programmierst, so wie **`'Hallo Welt'`** im Beispiel vorhin. Sie brauchen immer einfache Anführungszeichen.

`Orange` sind Zahl-Parameter. Die brauchst du später z.B., wenn du Entfernungen oder Geschwindigkeiten angeben willst.

`Grün` sind Kommentare. Sie haben keinen Einfluss auf das Programm, sondern sind eine Hilfe für dich, damit du auch später noch drauskommst.

`Schwarz` sind die Namen von Modulen (Bereiche und Bausteine) und Befehlen, sowie von Variablen und Funktionen. Auch Satzzeichen werden in der Regel schwarz formatiert, bis auf Klammern `()`, die sind dunkelgrün.

Tipp:

Hilfe, meine Farben sind falsch!

Sieht dein Code plötzlich so aus?

```
3 light_matrix.write(Hello, World!')
```

Dann ist da ein Fehler! Du weisst schliesslich, dass **`Hello, World!`** ein Text-Parameter ist und darum pink sein muss. Aber wenn du genau schaust, siehst du, dass du ein Anführungszeichen vergessen hasst! Wenn du das noch einfügst, sieht es wieder richtig aus.

Auch die roten Wellenlinien zeigen, dass in deinem Code ein Fehler ist. Das passiert auch, wenn du z.B. den Namen eines Befehls falsch schreibst:

```
3 light_martix.write('Hello, World!')
```

Findest du den Fehler?

1.5 Einen Timer verwenden

Lernziel

- ☐ Ich weiss, wie ich einen Timer verwende, um eine Aktion nach einer bestimmten Zeit zu beenden.

Manchmal wird eine Aktion für eine bestimmte Dauer benötigt. Es soll z.B. auf der Lichtmatrix für genau zwei Sekunden ein Herz angezeigt und dann wieder ausgeschaltet werden.

So sieht der Code aus:

```
1 from hub import light_matrix
2 import time
3 # wir brauchen ein Modul, das die Zeit messen kann. Es heisst time.
4 time ist ein alleinstehendes Modul, das keinen übergeordneten Bereich braucht.
5
6 light_matrix.show_image(1)
7 # show_image() ist der Befehl, um ein Bild aus der internen Datenbank anzuzeigen.
8 # Bild Nummer 1 ist ein Herz.
9
10 time.sleep_ms(2000)
11 # Das ist der Befehl zum Warten.
12 # Wir rufen das time-Modul auf, zu dem der Befehl sleep_ms() gehört.
13 # Der Parameter ist die Wartezeit in Millisekunden.
14 # Um 2 Sekunden zu warten, müssen wir 2000 eingeben.
15
16 light_matrix.clear()
17 # Mit diesem Befehl wird die Lichtmatrix wieder ausgeschaltet.
18 # Wir brauchen hier keinen Parameter, denn es gibt keine verschiedenen Varianten
von "aus".
```

Aufgabe 3

- Probiere aus, was für Bilder angezeigt werden, wenn du den Parameter in Zeile 6 änderst!
- Ändere den Parameter in Zeile 10. Was ist die kleinste Zahl, die du eingeben kannst, um das Bild noch zu sehen?

2. Motoren

Jetzt wird es interessant, wir benutzen Motoren! Schliesslich soll unser Roboter arbeiten und fahren können!

Wir fangen mit einem einzelnen Motor an, den man z.B. für ein Werkzeug braucht, mit dem ein Objekt eingesammelt werden kann.

2

2.1 Ein einzelner Motor

Lernziel

- ☐ Ich kann einen einzelnen Motor verwenden und kenne die wichtigsten Befehle und Parameter.

Wir wollen einen einzelnen Motor verwenden.

Schliesse dafür einen mittleren Motor am Port A an und bringe an der blauen Drehscheibe des Motors einen anderen Legostein so an, dass du Drehungen leichter beobachten kannst.

Wir wollen jetzt Code schreiben, mit dem sich dieser eine Motor dreht. Und zwar soll er 3 ganze Umdrehungen machen, mit einer Geschwindigkeit von 2 Umdrehungen pro Sekunde (U/s)

Zunächst müssen wir die benötigten Module importieren:

```
import motor
from hub import port
```

Der Baustein **`motor`** enthält alle Befehle für einen einzelnen Motor. Der Baustein **`port`** ist zuständig, um die Anschlüsse (Ports) zu kontrollieren.

Jetzt kommt noch der Befehl: Er hat die Form

```
motor.run_for_degrees(Port, Grad, Geschwindigkeit)
```

Tipp:

Dieser Befehl hat drei Parameter. Wenn hier im Dossier in Code-Beispielen deutschsprachige Wörter verwendet werden, dann sind dies Hilfs-Bezeichnungen für Parameter, die ausgewählt werden sollen. Dies hilft dabei, sich zu merken, was wir brauchen. Um dies noch leichter sichtbar zu machen, dass das keine echten Code-Wörter sind, werden sie im Dossier zusätzlich auch ***`kursiv`*** gedruckt.

- Den ***`Port`*** geben wir so an: **`port.A`** oder **`port.B`**, je nachdem, welchen Port wir brauchen.
- ***`Grad`*** zeigt, wie weit sich der Motor drehen soll. Dabei müssen wir vielleicht etwas rechnen: Eine Umdrehung sind 360 Grad (=360°). Wenn der Motor also drei ganze Umdrehungen machen will, ist unser zweiter Parameter `1080`.
- Die ***`Geschwindigkeit`*** gibt an, wie schnell der Motor sich drehen soll. Die Geschwindigkeit wird in Grad pro Sekunde (°/s) angegeben. Da wir wissen, dass eine Umdrehung 360° sind, können wir also auch ausrechnen: 1 U/s = 360 °/s, und für uns relevant: 2 U/s = `720` °/s.
- **Alle Parameter werden ohne Einheitenzeichen eingegeben!**

Unser Code sieht jetzt also so aus:

```
import motor
from hub import port

motor.run_for_degrees(port.A, 1080, 720)
# Beachte die drei Parameter!
```

Aufgabe 4

- Experimentiere mit unterschiedlichen Parametern aus. Benutze für die Parameter **Grad** und Geschwindigkeit auch negative Zahlen, abwechselnd und gleichzeitig z.B. `-360`. Was passiert jetzt?

2.2 Zwei Motoren, die nicht zusammengehören

Lernziel

- ☐ Ich kann zwei unabhängige Motoren steuern und weiss, wie ich sie gleichzeitig oder nacheinander laufen lasse.

Jetzt versuchen wir etwas anderes: Wir hängen zwei Motoren gleichzeitig an! Der erste Motor bleibt am Port A, der zweite Motor kommt an Port B. Die beiden Motoren sollen unabhängig voneinander funktionieren, weil sie z.B. für zwei verschiedene Werkzeuge verwendet werden. Sie sind also nicht die beiden Antriebsmotoren eines Fahr-Roboters.

Der neue Code sieht so aus:

```
import motor
from hub import port

motor.run_for_degrees(port.A, 360, 720)
motor.run_for_degrees(port.B, 360, 720)
```

Was geschieht?

Eigentlich könnte man ja vermuten, dass zuerst der Motor an Port A eine Umdrehung macht, und anschliessend der Motor an Port B. Aber beide Motoren bewegen sich gleichzeitig!

Merke:

`Run_for_degrees()` ist ein Befehl, der nicht erst abgeschlossen sein muss, bevor der nächste Befehl gestartet wird. Sondern er wird gestartet, und unmittelbar nach dem Start wird auch der nächste Befehl gestartet. Da der Zeitabstand so klein ist, sieht es für das menschliche Auge so aus, als wäre es genau gleichzeitig, in Wirklichkeit sind ein paar Millisekunden Abstand.

Befehle, die auf diese Art funktionieren, die also nicht erst abgeschlossen sein müssen, nennt man **abwartbare Befehle**. Das bedeutet: In der Regel laufen diese beiden Befehle gleichzeitig ab, du kannst aber auch abwarten, bis der eine vollständig ausgeführt ist, bevor der nächste Befehl an die Reihe kommt.

Schwer zu verstehen? Stelle es dir im Alltag vor: Du möchstest ein Lied auf dem Handy hören, und du möchtest eine Runde Candy Crush auf dem Handy spielen. Du kannst beides gleichzeitig machen: Während du Candy Crush spielst, läuft auch die Musik. Aber du kannst auch zuerst das Lied ganz hören und erst danach das Spiel spiele.

Wie können wir es jetzt machen, dass die beiden Befehle nacheinander ausgeführt werden, dass sich also zuerst der Motor an Port A dreht, und dann der an Port B?

Das geht. Wir brauchen zunächst für unser Programm einen Rahmen:

```
1 import runloop
2
3 async def main():
4     # Schreibe hier dein Programm
5
6 runloop.run(main())
```

Zeile 1: Als erstes importieren wir zusätzlich zu den anderen benötigten Modulen den Baustein **`runloop`**. Er enthält die notwendigen Schlüsselwörter, mit denen wir steuern können, ob die abwartbaren Befehle gleichzeitig oder nacheinander ausgeführt werden sollen.

Zeile 3: Als nächstes definieren wir eine so genannte Funktion. Ein komplexes Programm wird oft in mehrere Teile unterteilt, die man als Funktionen bezeichnet. (Was Funktionen genau sind, schauen wir unten im Kapitel 5 an Für den Moment brauchst du nur die Funktion **`main()`**).

Die Funktion, die wir hier brauchen, heisst **`main()`** und bekommt das Schlüsselwort **`async`**, was «nicht gleichzeitig» bedeutet. Dieses Schlüsselwort zeigt uns an, dass in der Funktion abwartbare Befehle sind, die nacheinander ausgeführt werden sollen. Das Schlüsselwort **`def`** zeigt an, dass der folgende Code die Definition der Funktion ist, das heisst, hier nach dieser Zeile kommen alle Befehle dieser Funktion.

Den Inhalt der Funktion schreiben wir dann eingerückt an die Stelle, wo im Moment der Kommentar
`    # Schreibe hier dein Programm`
steht.

Zeile 6: Als letztes rufen wir die Funktion auf, d.h. wir sagen dem Programm, dass sie jetzt ausgeführt werden soll. Dieser Aufruf ist nicht mehr eingerückt.

Der komplette Code sieht jetzt so aus:

```
1  import motor
2  import runloop
3  from hub import port
4
5  async def main():
6      await motor.run_for_degrees(port.A, 360, 720)
7      # Mit dem Schlüsselwort await wird der Code so lange blockiert,
8      # bis diese Zeile fertig ausgeführt ist.
9      await motor.run_for_degrees(port.B, 360, 720)
10 runloop.run(main())
```

Zeile 6: Hier siehst du das Schlüsselwort **`await`**. Dieses friert den Code ein, bis der Befehl dieser Zeile vollständig durchgeführt wird. Erst wenn Motor A sich fertig gedreht hat, beginnt die Drehung von Motor B.

Aufgabe 5

- Ergänze jetzt den Code so, dass die beiden Motoren zuerst nacheinander und dann noch einmal gleichzeitig laufen. (Lösung am Ende des Dossiers)

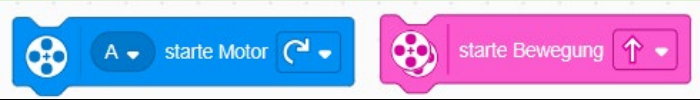
2.3 Antriebsmotoren bei einem Fahr-Roboter

Lernziele

☐ Ich weiss, was Antriebsmotoren sind, und kann sie richtig festlegen.

Wenn wir einen Fahrroboter haben, brauchen wir zwei Motoren, die fest zusammenarbeiten. Dazu müssen wir einen anderen Bereich importieren, das **`motor_pair`** (Motorenpaar). Jetzt werden diese beiden Motoren als eine Einheit betrachtet und können mit einem gemeinsamen Befehl gesteuert werden.

Tipp:
Wenn du bisher mit Scratch gearbeitet hast, kennst du den Unterschied bereits: Ein Einzelmotor wird mit blauen Blöcken gesteuert, ein Motorenpaar mit pinken Blöcken.



Jetzt müssen wir unsere Antriebsmotoren festlegen. In Scratch sah das so aus:

In Python sieht das so aus:

```
1 from hub import port
2 import runloop
3 import motor_pair
4
5 async def main():
6     # Verknüpfe die Motoren an den Anschlüssen A und B zu einem Motorenpaar:
7     motor_pair.pair(motor_pair.PAIR_1, Port_Linker_Motor, Port_Rechter_Motor)
8     # Was auch immer das Motorenpaar jetzt machen soll
9
10 Runloop.run(main())
```

Die wichtige Zeile ist **Zeile 7**. Sie zeigt an, welche beiden Ports wir brauchen. Schauen wir uns diese Zeile Stück für Stück an:

`motor_pair.pair()`: Dies ist der Befehl, dass zwei Motoren gekoppelt werden und von jetzt an als eine Einheit gelten. Der Befehlt hat drei Paramenter:

- **`motor_pair.PAIR_1`**: Dieser Parameter bleibt immer gleich. Die einzige Ausnahme ist, wenn dein Roboter mehr als zwei Räder hat, mit denen er fahren könnte, und zwischen den verschiedenen

Möglichkeiten umgeschaltet werden soll. Dann könntest du auf **PAIR_2** und **PAIR_3** noch weitere Paare festlegen. (Dieser Fall wird aber in der Realität eher selten vorkommen. Trotzdem verlangt SPIKE Python von dir, jedes Mal, wenn du das Motorenpaar verwendest, zu schreiben, dass es das **PAIR_1** ist.)

- *Port_Linker_Motor*: Gib hier den Port an, an dem der in Fahrtrichtung linke Motor eingesteckt ist. Ports werden, wie wir bereits früher gesehen haben, immer in der Form **port.A** angegeben.
- *Port_Rechter_Motor*: Gib hier den Port an, an dem der in Fahrtrichtung rechte Motor eingesteckt ist.

Aufgabe 6

- Baue einen ganz einfachen Roboter, der ausser dem Hub noch zwei Motoren an den Anschlüssen A und B hat, an jedem Motor ein Rad, und am anderen Ende das türkise Kugelrad als Stütze. Die Kursleitung hat eine Vorlage, wenn du es nicht alleine schaffst.
- Lege ein neues Projekt an und nenne es **PY_Fahrschule**.
- Importiere die notwendigen Module.
- Definiere eine **async** Coroutine namens **main()**.
- Lege innerhalb der Coroutine das Motorenpaar fest.
- Mit diesem Code werden wir im nächsten Abschnitt weiter arbeiten.

2.4 Fahren

Lernziel

- ☐ Ich weiss, wie ich meinen Roboter vorwärts und rückwärts fahren lasse.
- ☐ Ich kann lenken.
- ☐ Ich kenne verschiedene Methoden, um anzuhalten.

2.4.1 Geradeaus fahren

Um den Roboter zu fahren, gibt es mehrere Möglichkeiten.

Die einfachste Möglichkeit ist, wenn ich für eine bestimmte Zeitdauer, z.B. 2 Sekunde, geradeaus fahren will. Wenn ich nichts anderes angebe, fährt der Roboter mit der Standardgeschwindigkeit der Motoren. Die ist ziemlich langsam!

```
1 from hub import port
2 import runloop
3 import motor_pair
4
5 async def main():
6     # Kopple zwei Motoren an den Anschlüssen A und B
7     motor_pair.pair(motor_pair.PAIR_1, port.A, port.B)
8     # Kurz warten, bevor der Motor startet, hilft bei der genauen Beobachtung.
9     await runloop.sleep_ms(2000)
10
11    # Starte die Bewegung mit Standardgeschwindigkeit (ziemlich langsam)
12    motor_pair.move(motor_pair.PAIR_1, 0)
13
14    # Stoppe den Roboter nach 2 Sekunden
15    await runloop.sleep_ms(2000)
16    motor_pair.stop(motor_pair.PAIR_1)
17
```

```
18 runloop.run(main())
19
20 raise SystemExit
```

Schauen wir uns jetzt wieder einzelne Zeilen genauer an. Ausnahmsweise fangen wir ganz am Schluss an.

Zeile 20: Wenn es dich bisher genervt hat, dass das Programm nicht beendet wird, solange du nicht auf den Knopf drückst, kannst du mit dieser Zeile das Programm ausschalten. Es ist sinnvoll, dies als die letzte Zeile von jedem Programm zu haben, damit der Roboter nach dem Ausführen der Befehle automatisch wieder in den Standby-Modus wechselt und weniger Akku braucht. Diese Zeile darfst du von jetzt an am Schluss jedes Programms haben, auch wenn es nicht extra erwähnt wird!

Zeile 12: Der Befehl **`motor_pair.move(`*`Motorenpaar`*`, `*`Lenkung`*`)`** hat zwei obligatorische Parameter:

- **`Motorenpaar`** ist das vorhin definierte Motorenpaar.
- *`Lenkung`* ist hier orange geschrieben, weil dieser Wert immer eine Zahl zwischen `-100` und `100` ist. Diese Zahl gibt an, ob und wie der Roboter sich dreht. Wenn der Parameter den Wert 0 hat, fährt der Roboter geradeaus. Details zum *`Lenkung`* -Parameter lernst du im nächsten Abschnitt.

Tipp:

Wenn du zum ersten Mal etwas neues ausprobierst, dann verwende kleine Zahlen: Kurze Strecken und kurze Zeiten bei niedrigen Geschwindigkeiten. Oder lass deinen Roboter auf dem Boden fahren! So verhinderst du, dass dein Roboter vom Tisch fällt.

Ist dir das etwas zu langsam? Dann ändern wir jetzt die Geschwindigkeit!

Wir müssen dafür Code-**Zeile 12** ändern. Wir müssen in den **`move()`**-Befehl noch einen weiteren Parameter für die Geschwindigkeit angeben. Aus

```
motor_pair.move(motor_pair.PAIR_1, 0)
```

wird

```
motor_pair.move(motor_pair.PAIR_1, 0, velocity=Tempo)
```

Wie auch beim Einzelmotor wird die Geschwindigkeit in Grad pro Sekunde (°/s) angegeben. Die Höchstgeschwindigkeit ist beim Mittleren Motor 1100 °/s, das ist ein bisschen mehr als drei ganze Umdrehungen pro Sekunde. Wenn du die türkisen Standard-Räder verwendest, entspricht das 53.5 cm/s. In zwei Sekunden fährt dein Roboter bei Höchstgeschwindigkeit also über einen Meter weit!

Aufgabe 7

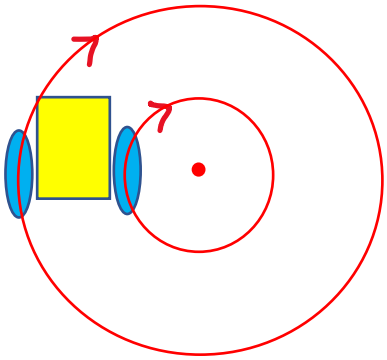
- Probiere den Code mit verschiedenen Geschwindigkeiten aus. (Beachte: höher als 1100 geht technisch nicht und führt zu einem Fehler!)
- Beobachte dabei, wie präzise der Roboter geradeaus fährt. Fährt er immer gleich gerade, oder eher in einem Bogen oder einer Wellenlinie? Gibt es eine Idealgeschwindigkeit, bei der er am besten geradeaus fährt?
- Beobachte auch, wie der Roboter sich bei den unterschiedlichen Geschwindigkeiten beim Bremsen verhält.
- Gib jetzt auch einmal negative Zahlen (zwischen 0 und -1100) ein. Was passiert?

2.4.2 Kurven und Drehungen

Jetzt wollen wir nicht immer nur geradeaus fahren. Wir wollen Kurven fahren und uns drehen. Dazu müssen wir wir im Code von vorhin in Zeile 12 noch den *Lenkung*-Parameter, wo bisher eine `0` steht, ändern. Er gibt an, wie stark die Drehung ist.

Dazu brauchen wir ein kleines bisschen Theorie. Es gibt nämlich drei Arten von Kurven bzw. Drehungen.

Kurve: Du kannst eine Kurve fahren (die eigentlich ein Teil eines Kreises ist:



In so einem Fall brauchen wir für den *Lenkung*-Parameter eine Zahl zwischen 0 und 50. Im Beispiel nehmen wir 30.

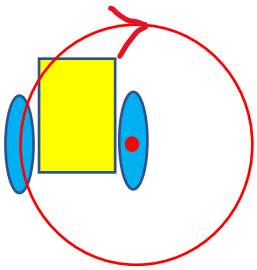
```
motor_pair.move(motor_pair.PAIR_1, 30, velocity=360)
```

Was hier genau geschieht: Der eine Motor bewegt sich etwas schneller als der andere. Durch diese ungleiche Bewegung entsteht die Drehung.

Aufgabe 8

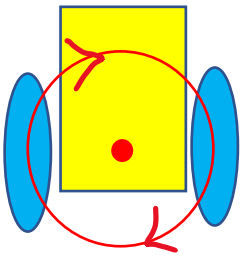
- Probiere aus, was passiert, wenn du die Zahl im ersten Einstellungsfeld grösser oder kleiner machst!
- Gib jetzt auch negative Zahlen zwischen 0 und -50 ein. Was geschieht?

Zirkeldrehung: Eine spezielle Form der Kurve ist die «Zirkeldrehung». Sie funktioniert genau wie ein Zirkel in der Geometrie: Ein Rad bleibt an der Stelle, das andere fährt.



Für eine Zirkeldrehung muss der *Lenkung*-Parameter 50 oder -50 sein (je nachdem, in welche Richtung wir drehen wollen).

1. **Kreiseldrehung**: Bei einer Kreiseldrehung dreht der Roboter sich um die Mitte zwischen seinen beiden Antriebsrädern.



Diese Art von Drehung benutzen wir am liebsten. Der Roboter braucht am wenigsten Platz, und wir können am genauesten festlegen, wie er nach der Drehung stehen soll. Der *Lenkung*-Parameter ist hierbei `100` oder `-100`.

Technisch drehen sich bei dieser Art von Drehung beide Räder genau gleich schnell, aber in die entgegengesetzte Richtung.

Für alle Arten von Drehung kannst du übrigens wie beim Geradeaus-Fahren auch die Drehung nach einer bestimmten Zeit beenden und das Motorenpaar stoppen.

2.5 Eine bestimmte Strecke fahren

Bisher ist dein Roboter für eine bestimmte Zeitdauer gefahren. Das geht, ist aber nicht immer sehr praktisch. Vor allem, wenn wir genau wissen, wie lange eine Strecke ist, die er zurücklegen soll, möchten wir gerne eine andere Möglichkeit haben.

Wir brauchen dafür statt dem einfachen **`move()`**-Befehl den Befehl `move_for_degrees()`.

Der Befehl sieht so aus:

```
motor_pair.move_for_degrees(motor_pair.PAIR_1, Grad, Lenkung, velocity=Tempo)
```

Wichtig ist hier vor allem das zweite Parameter *Grad*: Es zeigt an, wie viele Grad die Motoren sich drehen sollen. Dabei gilt wieder, dass
1 Umdrehung = 360 Grad sind.

Beispiel: Der Roboter soll genau 45 cm weit geradeaus fahren und dann stoppen. Wie kommen wir jetzt von 45 cm auf eine Grad-Zahl?

Hier brauchen wir ein Massband und müssen ein bisschen rechnen:

Mit dem Massband messen wir den Umfang der Räder, die an den Fahrmotoren sind. Wenn du die Standard-Räder aus der Spike-Box verwendest, dann brauchst du nicht zu messen: Der Umfang beträgt 17.5 cm. Bei anderen Rädern musst du aber auf jeden Fall messen, denn je nach Grösse der Räder kommt etwas ganz anderes heraus!

Jetzt musst du rechnen: Wie oft passt der Radumfang 17.5 cm in die Strecke von 45 cm?

$$\frac{45cm}{17.5cm} = 2.571$$

Und als zweiten Schritt:

$$2.571 * 360° = 925.56°$$

Da Python als Gradzahl aber nur eine ganze Zahl akzeptiert, müssen wir auf die nächste ganze Zahl runden, also 926°. (Und beim Schreiben des Codes das Grad-Zeichen weglassen.). Der Code sieht also jetzt so aus:

```
motor_pair.move_for_degrees(motor_pair.PAIR_1, 926, 0, velocity=360)
```

Tipp:

Wenn du eine Kurve oder Drehung machst, funktioniert das genauso. Der Parameter `Grad` gibt dabei an, wie weit der schnellere Motor sich drehen soll. Man kann den zwar mathematisch ganz genau ausrechnen, aber im Alltag ist man oft schneller, wenn man etwas rumprobiert. Mit der Zeit entwickelt man auch eine gewisse Erfahrung, welche `Grad`-Werte welche Drehung ergeben.

3. Sensoren

3

3.1 Der Farbsensor

Lernziel:

- ☐ Ich weiss, wie ich die Werte an einem Farbsensor auslesen und anzeigen lassen kann.

3.1.1 Farben

Der Farbsensor kann zwei verschiedene Arten von Werten messen bzw. erkennen: Farben und reflektiertes Licht. Schauen wir zunächst die Farben an:

Tipp:

Der Sensor kann nur die folgenden Farben erkennen: Rot, Grün, Blau, Magenta, Gelb, Orange, Türkis, Hellblau, Schwarz, Weiß. Wenn er andere Farben vor sich hat, oder mehrere Farben gleichzeitig, kann er es nicht eindeutig erkennen. Daher ist die Position des Sensors im Verhältnis zum Objekt sowie die Umgebungsbeleuchtung sehr wichtig. Farberkennung gehört zu den Teilen im Code, die am häufigsten getestet werden müssen, um Fehler zu vermeiden.

Auch sind manche Farben für den Sensor grundsätzlich schwer zu erkennen. Insbesondere Orange, Türkis und Magenta werden nur schlecht erkannt. Versuche also nach Möglichkeit, diese Farben zu vermeiden.

So lange der Hub mit dem Computer/Tablet verbunden ist, können wir in der Status-Anzeige sehen, welche Farbe der Farbsensor gerade erkennt.

In diesem Screenshot erkennt er auf Port C Blau. Wie du siehst, hat Blau den Zahlwert 3. Diesen Zahlwert brauchst du später, wenn du z.B. in einer Bedingung eine bestimmte Aktion ausführen willst, wenn der Sensor Blau erkennt.

Beispiel: Wir schreiben jetzt ein ganz einfaches Programm, mit dem der Roboter immer eine Farbe erkennt, ihren Zahlwert in den Ausgabe-Bereich (die Konsole) schreibt, zwei Sekunden wartet (damit du Zeit hast, ein anderes Objekt vor den Sensor zu halten) und dann von vorne anfängt. Das Programm wird so lange ausgeführt, bis du es manuell stoppst.

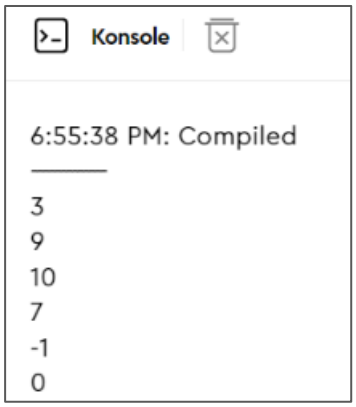
```
from hub import port
import color_sensor
import runloop
import time

async def main():
    while True:
        # Scanne die Farbe des Objektes und schreibe den Wert in die Konsole.
        print(color_sensor.color(port.C))
        # Warte zwei Sekunden und wiederhole.
        time.sleep_ms(2000)

```

```
13 runloop.run(main())
```

Zeile 2: Hier müssen wir den Baustein **`color_sensor`** importieren.

Zeile 7: `while True` leitet eine Endlosschleife ein. Wir werden Schleifen im nächsten Kapitel genau kennenlernen.

Zeile 9: Hier wird der Sensor-Wert erkannt und in die Konsole geschrieben. Das Ergebnis sieht beim Ausführen dann z.B. so aus:



Das Ergebnis ist so zu verstehen: Jede Zeile ist ein Messdurchgang. Die Zahl ist der Zahlwert der jeweils erkannten Farbe; **`-1`** bedeutet, dass keine Farbe erkannt wurde.

Diese Werte sind möglich:

- 0=schwarz
- 1=pink (magenta)
- 2=violett
- 3=blau
- 4=hellblau oder türkis
- 6=grün
- 7=gelb
- 8=orange
- 9=rot
- 10=weiss

Im Screenshot oben hat der Sensor also in dieser Reihenfolge die Farben Blau, Rot, Weiss, Gelb, keine Farbe, Schwarz erkannt.

Achtung: Manche Farben werden nur schlecht erkannt und führen oft zu einem Fehler, obwohl das Programm richtig ist. Dazu gehören insbesondere die Mischfarben Orange und Turkis. Am zuverlässigsten werden neben Schwarz und Weiss die Farben Rot, Gelb und Grün erkannt.

3.1.2 Reflektiertes Licht

Als nächstes benutzen wir den Farbsensor, um das reflektierte Licht abzufragen. Wir schreiben es wieder in die Konsole.

Wenn der Sensor das reflektierte Licht misst, gibt er Werte von `0` bis `100` zurück. Diese kann man so interpretieren: Je grösser die Zahl, desto mehr Licht wird reflektiert und desto heller ist das Objekt. Je kleiner die Zahl ist, desto dunkler ist es. Wenn im Messfeld des Sensors sowohl weiss als auch

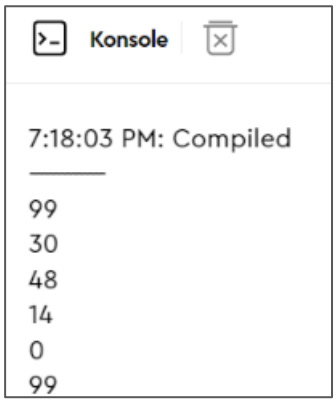
schwarz vorkommst, misst der Sensor den Mittelwert. Wenn du den Sensor direkt über die Kante zwischen etwas Weissem und etwas Schwarzem hältst, bekommst du einen Wert von ungefähr 50. Diese Werte machen wir uns z.B. bei einem Linienverfolger zunutze.

Wir müssen aus dem Code zur Farberkennung vom vorherigen Abschnitt nur eine einzige Zeile ändern:

```
9        print(color_sensor.reflection(port.C))
```

Halte nun beim Testen den Sensor vor etwas Schwarzes, etwas Weisses, etwas, das sowohl Schwarz als auch weiss enthält, sowie über verschiedene Graustufen.

Das Ergebnis könnte so aussehen:



3.2 Der Abstandssensor

Lernziel

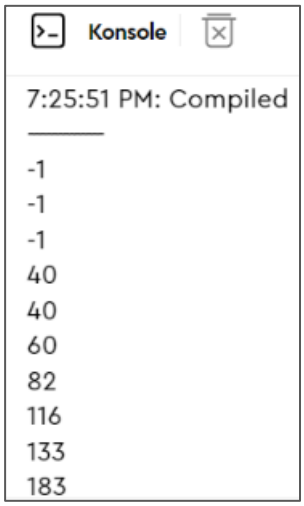
- ☐ Ich kann mit dem Abstandssensor auslesen, wie weit der Sensor von einem Hindernis entfernt ist.

Das Prinzip bei dieser Aufgabe ist dasselbe wie beim Farbsensor.

Beispiel: Wir lesen den Abstand aus und schreiben ihn in die Konsole. Wir wählen dieses Mal aber einen kürzeren Zeitabstand, (200ms = 0.2s), da wir den Sensor einfach in der Hand halten und bewegen können.

```
1  from hub import port
2  import distance_sensor, runloop, time
3
4  async def main():
5      while True:
6          # Messe den Abstand zum Farbsensor und schreibe den Wert in die Konsole.
7          print(distance_sensor.distance(port.D))
8          # Warte eine 0.2 Sekunden und wiederhole.
9          time.sleep_ms(200)
10
11 runloop.run(main())
```

Zeile 7: Hier wird der Abstand, den der Sensor am Port D misst, in die Konsole geschrieben.



Wenn hier ein Wert von -1 angezeigt wird, bedeutet das, dass der Sensor zu weit von seinem Ziel entfernt ist oder zu nah dran ist. Erlaubte Entfernungen liegen zwischen 5 cm und 2 m. Ein Ergebnis von -1 kann auch dann kommen, wenn der Sensor in einem so merkwürdigen Winkel zum Objekt ist, dass der Sensor nicht weiss, an welcher Stelle er messen soll.

4. Schleifen und Bedingungen

4

4.1 Schleifen

Lernziel

- ☐ Ich kenne die verschiedenen Arten von Schleifen und weiss, wann und wie ich sie verwende.

Merke:

Die goldene Regel für Programmierer lautet:

Schreibe nicht etwas zweimal,
wenn du es auch einmal schreiben kannst!

4.1.1 Schleife mit einem Zähler

Die einfachste Art einer Schleife ist es, wenn du angibst, wie oft etwas geschehen soll.

Beispiel: Dein Roboter soll ein Quadrat fahren, dessen Seitenlänge genau 17.5 cm beträgt (Das ist der Umfang der SPIKE-Standardräder, also kannst du genau eine Motorumdrehung = 360 Grad angeben. Auf der Lichtmatrix soll er jeweils die Seiten-Nummer anzeigen. Der Code dafür sieht so aus: Da du weisst, dass ein Quadrat genau 4 gleich lange Seiten und genau 4 rechte Winkel hat, kannst du das mit einer Schleife machen, in der eine Seite und ein Winkel 4x wiederholt werden.

```
1  from hub import light_matrix
2  import motor_pair
3  from hub import port, sound
4  import runloop
5
6  async def main():
7
8      motor_pair.pair(motor_pair.PAIR_1, port.A, port.B)
9
10     i = 0
11     while i < 4:
12         light_matrix.write(str(i + 1))
13         await motor_pair.move_for_degrees(motor_pair.PAIR_1, 360, 0, velocity=360)
14         await motor_pair.move_for_degrees(motor_pair.PAIR_1, 210, 100, velocity=180)
15         i += 1
16     sound.beep(440, 500, 100)
17
18 runloop.run(main())
```

Schauen wir uns diesen Code genauer an. Du kennst die import-Zeilen schon, da wir sie bereits bei den früheren Beispielen verwendet haben. In Zeile 3 werden aus dem Bereich **hub** zwei Bausteine importiert.

Zeile 8: Wir legen die Antriebsmotoren an den Ports A und B fest.

Zeile 10: Wir bestimmen einen **Zähler** und den Startwert, den dieser Zähler vor Beginn der Schleife hat. Diesen nennt man oft `i` oder einen anderen einzelnen Buchstaben. Der Zähler ist eine Variable. Wir werden Variable später noch genauer anschauen. Als Startwert für den Zähler gibt man in der Regel `0` an.

Tipp:
Der Name `i` für solche Zähler stammt noch aus den Anfangszeiten des Programmierens, wo man versucht hat, alles so kurz zu halten wie möglich, weil Speicherplatz teuer war. Er ist eine Abkürzung des englischen Wortes «integer», welches «ganze Zahl» bedeutet, also eine Zahl wie 0, 1, 256 oder -13. Denn beim Zählen zählt man üblicherweise in ganzen Zahlen, nicht in Bruchzahlen.

Wenn du wie in diesem Beispiel nur einen einzigen Zähler hast, dann kannst du `i` verwenden. Wenn du aber mehrere Dinge zählen musst, kommt man da schnell durcheinander. Manche Programmierer nehmen für den zweiten Zähler im Programm dann einfach den Buchstaben `j`. Aber das wird mit der Zeit unübersichtlich! Deshalb ist es sinnvoll, sich dann einen wirklich einprägsamen Namen auszudenken. Im Beispiel können wir den Zähler **`SEITENNUMMER`** nennen. Der Code sieht dann so aus:

```
SEITENNUMMER = 0
while SEITENNUMMER < 4
    # Hier ist der Code für die Seite und die Drehung
    SEITENNUMMER += 1
```

Zeile 11: Hier sagen wir, wie oft die Schleife durchlaufen werden soll. In unserem Beispiel wollen wir ein Quadrat fahren, deshalb wird die Schleife 4x durchlaufen. Wenn, wie es die Regel ist, der Startwert des Zählers 0 ist, dann müssen wir bis 3 zählen (0, 1, 2, 3). Das ist nicht ganz logisch für den Alltag, wird aber beim Programmieren generell so gemacht. Im Code drücken wir das so aus:

`while i < 4`, oder auf Deutsch: Solange `i` kleiner als `4` ist.

Zeile 12 bis 15: Hier steht der Code, der in der Schleife wiederholt werden soll. Um genau zu sehen, was in der Schleife ist, rücken wir diesen Code ein. Danach geht es nicht eingerückt wieder weiter.

Beachte: Die Gradzahl in Zeile 14 (im Beispiel 210) ist davon abhängig, wie dein Roboter konstruiert ist. Massgeblich sind der Radabstand sowie die Grösse der Räder.

Tipp:

Du kennst das bereits von Scratch: Da steht das, was wiederholt werden soll, in der Klammer des C-Blocks. Danach geht es mit etwas anderem weiter.

Zeile 14: Beachte, dass Parameter mit der Gradzahl (im Codebeispiel `210`) davon abhängig ist, wie dein Roboter konstruiert ist. Je enger die beiden Räder beieinander sind, desto kleiner ist die Zahl, je weiter sie auseinander sind, desto grösser ist die Zahl. Auch die Grösse der Räder hat einen Einfluss darauf. Am besten ausprobieren!

Zeile 15: `i += 1`
Am Ende setzen wir den Zähler um eins hoch. Dadurch verändert sich der Wert von `i`.

Diese seltsame Schreibweise ist eine Kurzform von
`i = i + 1`

und bedeutet ausgeschrieben: Der Zahler i soll jetzt den Wert des bisherigen Zählers plus 1 haben. Das heisst: Wenn i zuletzt 3 war, dann ist es ab sofort 4.

Zeile 16: Die Schleife ist vorbei. Zur Kontrolle spielen wir einen kurzen Piepton. Dieser Code ist nicht mehr eingerückt, denn er ist nicht mehr Teil der Schleife.

Aufgabe

- Modifiziere den Code für das Quadrat so, dass er nach jeder Drehung einen Piepton abspielt. Nach Abschluss des Quadrats soll er auf der Lichtmatrix für eine Sekunde ein Quadrat anzeigen. (Lösung am Ende des Dossiers)

4.1.2 Endlosschleife

4.1.3 Schleife mit einer Stoppbedingung

4.2 Bedingungen

Lernziel

- ☐ Ich kann einfache und verschachtelte Bedingungen richtig verwenden.

4.2.1 Einfache Bedingungen

4.2.2 Verschachtelte Bedingungen

5. Funktionen («MyBlocks»)

Lernziel

- ☐ Ich weiss, was eine Funktion ist, und wofür sie nützlich ist.
- ☐ Ich kann Funktionen ohne und mit Parametern erstellen und aufrufen.
- ☐ Ich kann eine Funktion mit Rückgabewert erstellen und auf diesen Wert zugreifen.

4.2.3 Funktionen ohne Parameter

Erinnerst du dich noch, wie wir die Code-Zeile `async def main()` kennengelernt haben? Da heisst es: «Ein komplexes Programm wird oft in mehrere Teile unterteilt, die man als Funktionen bezeichnet.» Jetzt ist der Moment gekommen, Funktionen genauer kennenzulernen.

Erinnerst du dich auch noch an die goldene Regel für Programmierer? Hier ist sie noch mal:

Merke:

Die goldene Regel für Programmierer lautet:

Schreibe nicht etwas zweimal,
wenn du es auch einmal schreiben kannst!

Wenn eine Aktion (oder eine Gruppe von Aktionen) mehrmals direkt hintereinander ausgeführt werden soll, nimmt man dafür am besten eine Schleife. Was macht man aber, wenn zwischen den beiden Durchgängen dieser Aktion etwas anderes passiert, das nicht wiederholt wird? Muss man dann den Code doch zweimal schreiben, mit allen Nachteilen, die dies mit sich bringt?

Nein, genau dafür gibt es **Funktionen**. Mit einer Funktion lagert man einen Programmteil aus, so dass man ihn später beliebig oft mit einer einzigen Code-Zeile aufrufen kann.

Beispiel: Der Roboter soll ein Quadrat fahren. Nach jeder Ecke soll er eine andere Aktion ausführen, z.B. einen unterschiedlichen Text auf der Lichtmatrix anzeigen oder einen unterschiedlichen Ton abspielen.

```
from hub import port, sound, light_matrix
import motor_pair, runloop, time

MP = motor_pair.PAIR_1
motor_pair.pair(MP, port.A, port.B)

async def FahreEineSeite(Motorenpaar):
    await motor_pair.move_for_degrees(Motorenpaar, 360, 0)
    await motor_pair.move_for_degrees(Motorenpaar, 210, 100)

async def main():
    await FahreEineSeite(MP)
    await sound.beep(440, 500, 100)
    await FahreEineSeite(MP)
    light_matrix.show_image(light_matrix.IMAGE_HAPPY)
    time.sleep_ms(1000)
    light_matrix.clear()
    await FahreEineSeite(MP)
    await sound.beep(500, 500, 100)
    await FahreEineSeite(MP)
    print("Fertig!")
runloop.run(main())

raise SystemExit
```

Zeile 4 und 5: Als erstes bestimmen wir Kurznamen und Variablen, die im gesamten Programm gebraucht werden. In unserem Beispiel ist das der Kurzname **`MP`** für das Motorenpaar, mit dem der Roboter fährt, und legen auch den Wert dafür fest. Es ist **`motor_pair.PAIR_1`**. Egal an welcher Stelle des Codes wir jetzt einen Fahrbefehl geben wollen können wir mit dem Kurznamen MP auf dieses Motorenpaar zugreifen.

Dies nennt man eine **globale Variable**. Wenn du eine Variable innerhalb einer Funktion definierst, dann kannst du sie nur in dieser Funktion selbst verwenden.

Zeile 7 bis 9: Hier definieren wir unsere Funktion **`FahreEineSeite()`**. Da wir in dieser Funktion abwartbare Befehle zum Fahren brauchen, muss am Anfang das Schlüsselwort **`async`** stehen, so wie bei **`main()`** auch. Den Code innerhalb der Funktion kennst du schon von der Schleife.

Zeilen 12, 14, 18 und 29: Hier wird vier Mal unsere Funktion aufgerufen. Du siehst, du brauchst nur eine einzige Zeile für all das, was in der Funktion steht, egal wie lang die Funktion tatsächlich ist.

Wichtig ist, dass du vor den Funktionsaufruf jeweils das Schlüsselwort **`await`** einfügst, damit die Funktion erst vollständig ausgeführt wird, bevor die nächste Codezeile kommt.

4.2.4 Funktionen mit Parameter

4.2.5 Funktionen mit Rückgabewert

Es gibt auch Funktionen, die einen Rückgabewert haben. D.h. sie enthalten nicht, oder nicht nur, eine Aktion, sondern sie haben ein Ergebnis, das an **`main()`** zurückgegeben werden und dort weiter verwendet werden kann.

Beispiel: Du erinnerst dich sicher noch an die komplizierte Berechnung, mit der du die Länge der Strecke, die du fahren willst, in Grad umrechnest, die die Motoren sich drehen müssen:

$$\frac{45cm}{17.5cm} = 2.571$$

Und als zweiten Schritt:

$$2.571 * 360° = 925.56°$$

Wir wollen jetzt eine Funktion schreiben, die uns diese Berechnung automatisch ausführt.

Als Vorüberlegung wandeln wir die Berechnung in eine Formel um, deren Teile wir als Variablen verwenden.

$$\frac{STRECKE\_IN\_CM}{RADUMFANG\_IN\_CM} = Zahl\ 1$$

Zahl 1 ist ein Zwischenergebnis, das wir in der zweiten Formel brauchen. Diese lautet:

$$Zahl\ 1 * 360° = GRAD$$

Wir können das in eine Formel zusammenfassen:

$$GRAD = 360° * \frac{STRECKE\_in\_CM}{UMFANG\_IN\_CM}$$

Und wir wissen, dass der Umfang der SPIKE-Standard-Räder 17.5 cm beträgt. Das heisst, wir können im Code dann statt der Variablen **`UMFANG_IN_CM`** direkt diesen Wert verwenden, oder wir haben ganz am Anfang des Codes eine globale Variable, der wir diesen Wert zuweisen. Das hat den Vorteil, dass wir im Falle eines späteren Umbaus des Roboters mit anderen Rädern den Code einfach weiterverwenden können und nur diese eine Zeile ändern müssen.

Schauen wir uns an, wie wir das als Funktion schreiben können.

```
1  from hub import port
2  import motor_pair, runloop
3
4  MP = motor_pair.PAIR_1
5  motor_pair.pair(MP, port.A, port.B)
6  UMFANG_IN_CM = 17.5
7
8  def BerechneGradAusCm(STRECKE_IN_CM):
9      GRADZAHL = 360 * STRECKE_IN_CM / UMFANG_IN_CM
10     GRADZAHL = round(GRADZAHL)
11     print(GRADZAHL)
12     return GRADZAHL
13
14 async def main():
15     motor_pair.move_for_degrees(MP, BerechneGradAusCm(45), 0)
16
17 runloop.run(main())
```

Zeile 4 bis 6: Wir definieren globale Variablen, damit der Code nachher schön übersichtlich bleibt.

Zeilen 8 bis 12: Das ist die Funktion. Sie hat einen Parameter **`STRECKE_IN_CM`**, den sie von **`main()`** bekommt.

Zeile 9: So sieht die Formel von oben im Code aus.

Merke:

Du kannst im Code jede Rechnung machen, die du auch im Matheunterricht machen kannst. Manches wird einfach etwas anders geschrieben. Dies sind die wichtigsten Operatoren (Rechenzeichen):
+ plus
- minus
*** mal**
/ geteilt
^ hoch (Das brauchen wir, weil man Zahlen im Python-Code nicht als Hochzahlen schreiben kann. 3^2 wird im Code so geschrieben: 3^2

Zeile 10: Jetzt runden wir das Ergebnis noch auf die nächste ganze Zahl, da wir überall, wo wir Grad brauchen, ganze Zahlen benötigen. Dezimalzahlen führen zu einem Fehler.

In Zeile 9 haben wir gesagt, dass die Variable `GRADZAHL` den Wert haben soll, der durch die Formel berechnet wird. Bei 45 cm sind das 925.6. Dies ist am Ende von Zeile 9 jetzt der neue Wert von `GRADZAHL`.

In Zeile 10 verändern wir den Wert der Variablen erneut: Dazu runden wir den letzten bekannten Wert.

Hinweis:

Code wie in Zeilen 9 und 10 ist keine Gleichung in der Mathematik! Deine Mathelehrperson wäre gar nicht begeistert, wenn du eine Gleichung so lösen würdest, weil links und rechts vom Gleichheitszeichen gar nicht dasselbe steht.

Das liegt daran, dass wir hier gar keine Gleichung haben. Sondern eine Variable, der wir einen Wert zuweisen. Und dieser Wert kann sich im Laufe des Codes, manchmal von Zeile zu Zeile verändern!

Ein Beispiel dafür haben wir bereits bei den Schleifen gehabt, wo wir ein Quadrat programmiert haben und dafür einen Zähler (`i` oder `SEITENNUMMER`) verwendet haben. Auch diesen Zähler haben wir verändert, wenn wir ihn um eins vergrössert haben.

Zeile 11 ist eigentlich nur eine Hilfszeile. Mit ihr schreiben wir den Wert von GRADZAHL in die Konsole. So können wir überprüfen, ob alles richtig ist. Diese Zeile kann später problemlos gelöscht werden.

Zeile 12: Hier wird festgelegt, dass `GRADZAHL` der Ausgabewehr ist, den die Funktion an `main()` zurückgibt. Wir brauchen dafür das Schlüsselwort `return`.

Zeile 15: In dieser Zeile kommt unsere Funktion zum Einsatz. Den Befehl
`motor_pair.move_for_degrees(MOTORENPAAR, GRAD, LENKUNG)`
kennen wir bereits.

- Für ***MOTORENPAAR*** verwenden wir die Kurzbezeichnung MP, die wir am Anfang des Codes in Zeile 4 definiert haben.
- Um ***GRAD*** zu bekommen, rufen wir jetzt die Funktion `BerechneGradAusCm` mit dem Parameter 45 auf, weil wir schliesslich 45 cm weit fahren wollen. Wenn das Programm ausgeführt wird,

springt es in die Funktion, rechnet das Ergebnis aus, und setzt dann als ***GRAD*** den Rückgabewert der Funktion ein, nämlich 926.

- Da wir geradeaus fahren wollen, ist ***LENKUNG*** 0.

Lösungen

In diesem Bereich findest du mögliche Lösungen für die Aufgaben, wo du selbst einen Code schreiben sollst.

Lösung zu Teil 2.2 Zwei Motoren
Lösung zu Teil 4.1.1. Schleifen